Self-Adaptive Machine Learning-based Systems

Maria da Loura Casimiro

CMU-S3D-24-112

December 2024

Software and Societal Systems Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Dr. David Garlan, Chair, Carnegie Mellon University
Dr. Paolo Romano, Co-Chair, IST, University of Lisbon
Dr. Christian Kästner, Carnegie Mellon University
Dr. Bruno Martins, IST, University of Lisbon
Dr. Grace Lewis, Software Engineering Institute
Dr. Valeria Cardellini, University of Roma Tor Vergata

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Software Engineering.

Support for this research was provided by Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) through the Carnegie Mellon Portugal Program under Grant SFRH/BD/150643/2020.

Keywords: Machine Learning, Self-Adaptive Systems, Model Retrain, Model Fine-tune, Misprediction

Palavras-chave: Aprendizagem automática, Sistemas auto-adaptáveis, Re-treino de modelos, Afinação de modelos, Previsões incorretas

For my parents

Abstract

Machine learning (ML) models are now commonly used as components in systems. Due to their black-box and data-driven nature, ML components can produce erroneous outputs (in the form of mispredictions) that may critically impact the system's quality of service. Such mispredictions may be caused by component changes, environment changes, or due to ML components' inherent uncertainty and inaccuracy. In the face of these changes, and to cope with mispredictions, self-adaptation arises as a natural solution: systems that monitor and adapt themselves at run time to optimize their system utility.

This thesis provides a repertoire of ML adaptation tactics, and a framework that generates policies specifying *when* to apply each tactic to adapt ML components such that overall system utility is optimized. The development of this framework raises two main challenges: (i) estimating the expected costs and benefits due to the execution of an adaptation tactic; (ii) evaluating the impact of the improved ML predictions on overall system utility. To address the first problem we build predictors that learn to estimate the expected benefits of each adaptation tactic. To solve the second problem we leverage probabilistic model checking methods and instantiate a formal model of the system, capturing the key dynamics of ML components and their impact on expected system utility.

The techniques proposed in this thesis are evaluated via two use-cases: credit card fraud detection and machine translation systems. We show that: the self-adaptive ML-based systems built leveraging the proposed framework achieve better system utility than that achievable when employing simpler baselines to guide adaptation, such as periodic or reactive adaptation schemes; the proposed framework is suitable for run-time adaptation in non-critical domains; the framework can be extended to account for multiple adaptation tactics; and that it can be leveraged to plan for the long term when to adapt ML models.

Resumo

Atualmente, modelos de Aprendizagem Automática (AA) são amplamente utilizados em diversos sistemas. Estes modelos são propícios a erros de previsão devido à sua dependência dos dados e à sua natureza opaca. Estes erros podem comprometer a qualidade do serviço oferecido e são causados, por exemplo, por alterações no próprio componente de AA, nos pedidos recebidos, ou pela incerteza inerente ao próprio modelo de AA. Para lidar com estas alterações, e de forma a prevenir erros, exploramos a ideia de sistemas de AA auto adaptáveis: sistemas capazes de se auto monitorizar e auto adaptar durante a execução de maneira a otimizar a qualidade de serviço que oferecem.

Nesta tese propomos um conjunto de táticas para adaptar componentes de AA e uma abordagem para gerar políticas que determinam *quando* utilizar cada tática para otimizar a qualidade de serviço. Há dois desafios principais, nomeadamente: (i) estimar os custos e benefícios esperados devido à execução de cada tática; (ii) avaliar o impacto esperado da adaptação na qualidade do serviço. Para resolver o primeiro desafio, criamos modelos que estimam os benefícios esperados associados à execução de cada tática. Para o segundo, usamos métodos de análise formal, criando um modelo formal do sistema de maneira a capturar as dinâmicas principais associadas ao componente de AA, à influência que cada tática exerce sobre ele, e ao impacto esperado da adaptação na qualidade do serviço.

Os métodos propostos são avaliados recorrendo a dois casos de estudo: sistemas de deteção de fraude e de tradução automática. Mostramos que: os métodos propostos permitem melhorar a qualidade de serviço, comparando com estratégias simples, como adaptações periódicas ou aleatórias; o tempo necessário para decidir como e quando adaptar é propício a adaptar sistemas não-críticos em tempo real; esta abordagem consegue analisar várias táticas e planear como adaptar considerando horizontes de planeamento longos.

Acknowledgments

It seems like it was only yesterday that I got on a plane to Pittsburgh to start my PhD journey. And what a journey it was... full of first times: first Thanksgiving – thanks Cat (and family!) for taking me in as a King, sharing your traditions with me, and showing me what Thanksgiving feels like; first international conference – thanks Pedro, for the good times we shared and the fun we had while working; first road-trips in American National Parks – thanks Diogo, Catarina, Daniel, Hugo, and Paulo for all the good times hiking; first Broadway show – thank you Catarina, for introducing us to a world of wonder.

However, I wouldn't have gotten this far without my support network. So if I am writing this today, it is all thanks to my family, advisors, and friends.

I want to start by thanking my advisors, professors Paolo Romano and David Garlan, for taking me in as their student and teaching me how to be a researcher; for continuously pushing me and for always being there to discuss ideas and discuss my questions. Second, I want to thank professor Luís Rodrigues, my unofficial advisor. Thank you for the opportunity to join the Distributed Systems Group and for your continuous support throughout. I like to think you wouldn't attend my weekly meetings unless you actually believed in my work. Having you there not only made the work better, but also motivated me to keep going when everything seemed to be going wrong.

The fun times and de-stressing are thanks to my friends, both at INESC – Cláudio, Rafael, Mafalda and João, Pedro, Coimbra – and CMU. Thank you Cat and Travis for teaching me about all things America, for always being there for me, for the road-trips to Canada, and for countless other good times we shared and that are too many to list here. Thank you Simon, for being the best office-mate I could have asked for. I would always look forward to our weekly runs and yoga, lunches and strolls with Tater. Thank you Parv, for the Sunday coffees and conversations, for the good times at 5801, for teaching me how I hate lifting weights, and for making me laugh at you. Thank you Bárbara, Pedro, and Lourenço, for always being close even when we were an ocean apart, for always being available to hang out and play board games. Finally, I thank the best house-mates I could have asked for, my "portuguese gang": Carolina, Daniel, Hugo, Paulo, and especially Catarina, for all the complicity and for being a sister to me. Our trips to the Benedum center, the house parties and dinners, shopping trips, road trips, late nights working, thesis proposal practice runs and so much more were invaluable.

Finally, and although there are not enough words, I thank my family, in particular my mom, dad, and Diogo, who are always there to hear me complain, to celebrate my wins, and to push me to always give the best of me. Thank you dad, for saying what I need to hear, even when I don't want to hear it. Thank you mom, for taking things out of the rational plane and bringing them into the emotional plane, for always understanding how I'm feeling and putting yourself in my shoes. Thank you Diogo, for being my rock.

Contents

List of Figures

List of Tables

Acronyms

AI artificial intelligence

AID adaptation impact dataset

AIP adaptation impact predictor

ATC average thresholded confidence

CB-ATC class-based-average thresholded confidence

FPR false positive rate

LA-AID look-ahead adaptation impact dataset

LA-AIP look-ahead adaptation impact predictor

LEC learning enabled component

LES learning enabled system

MAE mean absolute error

ML machine learning

MLC machine-learning component

MLS machine-learning-based system

MT machine translation

MDP markov decision process

NN neural network

PCC pearson correlation coefficient

SAS self-adaptive system

SLA service level agreement

TL transfer learning

TNR true negative rate

TPR true positive rate

Chapter 1

Introduction

Machine learning (ML) based systems, which are systems composed of both ML and non-ML components, are pervasive nowadays [157]. Examples of ML based systems include robots [84], medical diagnosis systems [62], autonomous vehicles [18, 40], and fraud detection systems [21, 64]. These systems' ML components are inherently subject to make mistakes – in the form of mispredictions – that may critically impact the quality of the services they provide. For instance, machine translation systems, which rely on (large) language models, can mistranslate sentences, for example because of misused punctuation marks, or lack of pronouns [5]. Such mistranslations may have catastrophic impacts, e.g., in social or ethical contexts [89]: a Palestinian man was arrested by Israeli police due to a mistranslation of a greeting in Arabic into hate speech in Hebrew [161].

Mispredictions may be due to an ML component's intrinsic (in)accuracy (e.g., when a linear model is used to predict the behavior of a component that has non-linear dynamics) or caused by (i) changes to other system components or (ii) environment changes. First, changes to any system component may ultimately impact an ML component. On the one hand, the ML component may start receiving new or different inputs. For instance, suppose a new sensor has been introduced in a system. The values measured by this sensor may constitute a new input feature to the ML component, which may require an update to ensure it can deal with the new feature. On the other hand, if downstream components relative to the ML component influence the data that is later processed by the ML component (e.g., via feedback loops within the system), then changes to these downstream components may lead to ML mispredictions.

Second, ML components are usually built based on the assumption that the data seen in their operational environment will be similar to the data that they were trained on. However, when this assumption does not hold, or changes over time, and the underlying environment under which the system operates is different than expected, the ML component may mispredict. For example, consider the scenario in which a fraudster devises new strategies to commit credit card fraud. The fraudulent transactions will thus be characterized by a set of features (e.g., transaction amount, zip-code) that the ML component cannot recognize as fraudulent. These phenomena are generally referred to as dataset shift [149].

In light of such changes, and as a means to cope with mispredictions, self-adaptation [52] arises as a natural solution: systems that can continuously monitor and adapt themselves such that their quality (often referred to as *system utility*) is optimized. Typical self-adaptive systems rely on a control layer containing a planner that, based on a repertoire of actions (termed *tactics*), dynamically computes an adaptation strategy (i.e., sequence of adaptation actions) to execute such that system utility is optimized [68, 90]. Hence, by rendering an ML-based system self-adaptive, it would be capable of detecting when ML components are negatively impacting system utility, for example through mispredictions, and of optimizing system utility through adaptation. Adaptation could be achieved by executing tactics such as model retrain or fine-tune, or by replacing the offending component with another component (ML or non-ML) whose performance is sub-par but predictable for the specific environmental context.

While the self-adaptive systems literature is extensive it has thus far typically targeted adaptation of non-ML systems [26, 28, 52, 84], or leveraged ML as part (but not as the target) of the adaptation process [72, 95, 156]. Our work takes a different approach by instead considering ML components as the *target* of the adaptation. To extend existing approaches to cope with adaptation of ML-based systems, the following challenges arise:

- Determining whether the ML component is mispredicting or will do so in the future;
- Understanding the adaptation space for ML-based systems, i.e., the repertoire of tactics available for the system to adapt itself in response to ML mispredictions;
- Understanding the pre- and post-conditions for each adaptation tactic, including its expected costs, benefits, and how these and their uncertainty impact systemic properties of the system (e.g., availability, adaptability, and scalability) and, ultimately, system utility;
- Synthesizing adaptation strategies for long-term optimization, i.e., for the following N time periods, such that expected system utility is optimal at the end of that horizon and reasoning about the periodicity with which those strategies should be reevaluated, i.e., how often the system should analyze the need for adaptation.

So far, researchers have been mostly concerned with the following problems: (i) understanding on which data ML components are more likely to mispredict [47]; (ii) demonstrating how improved ML components (i.e., retrained ML components [104] or ML components with optimized hyper-parameters [114, 154, 155]) offer benefits at the ML component level, such as increased model accuracy or lower training cost; and (iii) leveraging ML techniques to enhance the adaptive capabilities of non-ML-based systems (e.g., leverage ML to improve a planner's capabilities and explore the space of adaptation tactics more efficiently) [72]. In contrast, this thesis focuses on the problem of determining *when and how* to adapt an ML-based system in order to optimize system utility. To address this problem, we are faced with a multitude of challenges associated with predicting the execution trade-offs of ML adaptation tactics, which are dependent on a number of factors, including: (i) cost [31, 154, 194], (ii) benefits [154], and (iii) opportunity [114, 115, 154, 155]. The problem of predicting the costs of executing an ML adaptation tactic has been addressed by the literature for tactics such as hyper-parameter tuning [31, 155, 194] and transfer learning [85]. Additionally, the cost of more-complex ML adaptation tactics, such as unlearning,

can be quantified via their execution latency [35], or via storage costs [20]. This thesis thus focuses on the problem of estimating the benefits of adaptation.

Regarding predicting the benefits of executing a tactic, and for instance considering the model retrain tactic, the benefits of its execution will be affected by: (i) the quality and quantity of the data available for the retrain (e.g., how representative of the new environment it is); (ii) the duration of the retrain process (e.g., longer retrains are more costly but may offer better accuracy); (iii) the hyper-parameters of the ML component and/or its architecture (both the hyper-parameters and/or the model's architecture can be updated as part of the retrain process or they may remain the same) [114, 154, 155].

Finally, in terms of opportunity, it is necessary to understand how the ML component's performance impacts system utility. Consider a system for which system utility is defined in terms of the costs incurred (e.g., monetary penalties due to violations of system level objectives) and the revenue obtained. For such a system, if revenue and cost are directly affected by ML performance (e.g., the higher the accuracy of the ML component, the higher the system's revenue and the lower its costs), determining whether ML adaptation is necessary and what its impact on system utility is only requires being able to estimate the costs and benefits of the adaptation. However, when system utility is not directly dependent on the ML component's predictive performance, determining whether it should be adapted requires a more complex system-level analysis, since an increase in accuracy will not necessarily increase revenue or decrease costs. For example, when system utility depends on how long the system takes to respond to user requests, if retraining does not improve an ML component's latency during inference (i.e., when the ML component is used online to respond to a user request), then the tactic's execution will not impact system utility. Yet, if as part of the retrain process the ML component's architecture or hyper-parameters are modified, then the latency during inference might be reduced, thus improving system utility.

Ideally, a self-adaptive ML-based system should be able to determine when to execute an adaptation tactic or sequence of tactics such that system utility is improved. To build such a system, it would be ideal to have access to a framework to guide this development process. This framework would serve as a model for the implementation of self-adaptive ML-based systems, and would be: *formal* – to enable precise and rigorous mathematical formulations and analyses of the synthesized adaptation strategies; *generic* – such that it can be applied to different system contexts (e.g., fraud detection, machine translation) and types of ML components (e.g., classifiers, regressors); *tractable* – in the sense that it is suitable for online adaptation of systems; *extensible* – as in being able to cater for the addition of adaptation tactics to its reasoning process as such tactics become available.

1.1 Thesis Statement

In this dissertation, I demonstrate that system utility of ML-based systems can be improved through adaptation of their ML components, by enabling self-adaptation of ML-based systems, as expressed in the following thesis statement:

We can engineer self-adaptive ML-based systems such that they are capable of optimizing system utility in the presence of environment changes that lead ML components to mispredict, by (a) reasoning about the tactics available to adapt ML components; and (b) developing a formal, generic, tractable, and extensible framework to construct self-adaptive ML-based systems that can plan for long-term adaptations.

As a starting point, this thesis begins by eliciting, from state-of-the-art research on ML [29, 104, 154, 191], some of the main adaptation tactics available to adapt ML components. The repertoire of ML adaptation tactics paves the way for the remainder of this thesis, which focuses on the development of the framework to determine *when* to adapt ML-based systems. The key idea of the proposed framework is to decouple the problems of (i) estimating the expected ML predictive performance improvement due to adaptations and (ii) estimating the impact of ML improved predictions on overall system utility. The first problem is addressed by relying on predictors that learn to estimate the expected benefits of each adaptation tactic. The second problem is tackled by leveraging model checking methods that, given a formal model of the system and a system utility definition, identify an expected optimal adaptation strategy. Despite the fact that model checkers can intrinsically reason about long-term predictions, the challenge in planning for the long term lies in incorporating the predictors to estimate the benefits of adaptation tactics into the formal model of the system. Since existing model checkers are unable to query external, more complex predictors (e.g., deep random forests) while model checking, the adaptation benefits predictors need to be simple enough such that they can be integrated in the formal model of the system in such a way that they still provide accurate estimates of the benefits.

The formality, generality, tractability, and extensibility of the framework developed in this thesis are evaluated first by instantiating the framework for a credit-card fraud detection system (a binary classification problem), and then by demonstrating its applicability for a machine translation system (a sequence-to-sequence problem). The quality of the adaptation strategies determined by the framework are evaluated by comparison with the adaptation strategies determined by the following commonly used baselines: optimal adaptation strategy, periodic system adaptation, reactive adaptation (adapt upon system utility constraint violations), and random adaptation.

Regarding *generality*, this thesis shows how the framework can be leveraged to instantiate an additional use case, beside fraud detection, namely ML-based machine translation systems. While the fraud detection system relies on classifier ML components (i.e., their output is restricted to a predefined set of discrete classes), the machine translation systems need to solve a sequence-to-sequence task, which is inherently more complex. These two use cases represent two distinct and prominent types of ML component, and highlight how the framework can be specialized to better fit the unique characteristics of each. For instance, while the fraud detection system has to comply with explicit service level agreement, the machine translation system is instead concerned with maximizing translation quality.

Formality is achieved by leveraging probabilistic model checking methods to synthesize (expected) optimal adaptation strategies. By relying on formal specifications, the framework provides a separation of concerns, which contributes to the framework's generality and extensibility. Further, the rigorous mathematical formulations which underpin formal

models allow for a stricter analysis of the problem at hand, the system requirements and assumptions that are accounted for. This analysis is stricter in the sense that it is more explainable – given that the modeling is governed by precise rules so there is no ambiguity in what the model is specifying – and certain – in the sense that mathematics is an exact science. This guarantees that interpretations regarding the model's tasks and goals are not subjective, thus making it viewer-independent and ensuring that all stakeholders agree on what the model is implementing.

The framework's *tractability* is evaluated by measuring the latency of the decision making process. In online adaptation scenarios, there typically is a window of opportunity during which the decision making process is allowed to run. This period varies depending on the domain: for example, real-time safety-critical systems such as autonomous vehicles require much faster decision making than movie recommender systems. This thesis shows that the proposed framework is tractable for domains such as recommender systems, enterprise systems, and Internet of Things (IoT) systems, but not necessarily for real-time safety-critical systems.

Finally, to extend a self-adaptive ML-based system and cater for additional tactics, three main modifications are required: (i) the specification of a tactic-specific module in the system's formal model, (ii) the inclusion of an actuator in the system itself to carry out the tactic's execution, and (iii) the ability to predict the tactic's costs and benefits. Thus, the framework's *extensibility* (i.e., how easily the framework allows for additional adaptation tactics to be accounted for when deciding the optimal adaptation strategy) is evaluated by: augmenting the set of tactics available for adaptation considered by the self-adaptive credit card fraud detection system, and testing the machine translation system with a tactic not applied to the fraud detection system.

1.2 Contributions

This thesis makes the following contributions:

1. **Repertoire of ML Adaptation Tactics:** A catalogue of existing tactics for adapting ML components together with guidelines on when they are likely to be of benefit (Chapter 4);

   The relevance of this contribution is reflected in the following publications:

   > Maria Casimiro, Paolo Romano, David Garlan, Gabriel A. Moreno, Eunsuk Kang, and Mark Klein. "Self-Adaptation for Machine Learning Based Systems." In *European Conference on Software Architecture (Companion)*. 2021.
   >
   > Maria Casimiro, Paolo Romano, David Garlan, Gabriel A. Moreno, Eunsuk Kang, and Mark Klein. "Self-adaptive machine learning systems: Research challenges and opportunities." In *European Conference on Software Architecture*, pp. 133-155. Cham: Springer International Publishing, 2021.

2. **Framework to Engineer Self-Adaptive ML-enabled Systems:** A generic, tractable, and extensible framework for adapting ML components of ML-based sys-

tems (Chapter 5), evaluated on two use cases: fraud detection system (Section 6.2) and machine translation system (Section 6.3);

The relevance of this contribution is reflected in the following publications:

> Maria Casimiro, David Garlan, Javier Cámara, Luís Rodrigues, and Paolo Romano. "A probabilistic model checking approach to self-adapting machine learning systems." In *International Conference on Software Engineering and Formal Methods*, pp. 317-332. Cham: Springer International Publishing, 2021
>
> Maria Casimiro, Paolo Romano, David Garlan, and Luís Rodrigues. "Towards a framework for adapting machine learning components." In *2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pp. 131-140. IEEE, 2022.
>
> Maria Casimiro, Diogo Soares, David Garlan, Luís Rodrigues, and Paolo Romano. "Self-Adapting Machine Learning-based Systems via a Probabilistic Model Checking Framework." *ACM Transactions on Autonomous and Adaptive Systems* (2024).

And by the following manuscript under preparation for submission:

> Maria Casimiro, Paolo Romano, José Souza, Amin M Khan, and David Garlan. "FLEXICO: Sustainable Machine Translation via Self-Adaptation", 2025

Artifacts:

The source code for the self-adaptive fraud detection system is publicly available, for replication and re-use by the community, at the following GitHub repository:

`https://github.com/cmu-able/ACSOS22-ML-Adaptation-Framework`

3. **Long-Term Planning of When to Adapt:** An approach to enable the framework to plan for the long term (Section 5.5.1), evaluated on the fraud detection system use case.

   The work associated with this contribution is currently under submission:

   > Maria Casimiro, Valentim Romão, David Garlan, Luís Rodrigues, and Paolo Romano. "Ripple: A Long-Sighted Self-Adaptation Approach to Retrain Machine-Learning-Enabled Systems", *ACM International Conference on the Foundations of Software Engineering (FSE)*, 2025

1.3 Outline

The remainder of this document is organized as follows: **Chapter 2** provides an overview of relevant related work and background. It introduces self-adaptive systems, machine-learning-based systems, and how ML has been recently leveraged in the field of self-adaptive systems. Additionally, it introduces the concept of data shift and the most common types of shift that typically affect ML components, possibly leading them to mispredict. **Chapter 3** overviews the proposed framework resorting to an illustrative example. **Chapter 4** introduces the repertoire of ML adaptation tactics, along with guidelines of when to apply

each tactic, their cost/benefits, and potential pros and cons. **Chapter 5** presents the framework proposed in this thesis, demonstrating how the framework can be leveraged to perform long-term planning of when to adapt ML-enabled systems (Section 5.5). The framework is evaluated in **Chapter 6** based on the claims detailed in Section 6.1 and via two distinct use-cases: fraud detection systems (Section 6.2) and machine translation systems (Section 6.3). Finally, **Chapter 7** provides a discussion on how to incentivize and facilitate the wide-spread adoption of the proposed framework (Section 7.1), discusses the limitations of the thesis (Section 7.2), overviews promising avenues for future work (Section 7.3), and concludes this document (Section 7.4).

Chapter 2

Background and Related Work

This chapter introduces the areas of the literature most relevant to this thesis. Specifically, Section 2.1 introduces the concept of self-adaptive system (SAS) and provides background on probabilistic model checking, a set of techniques typically employed in the self-adaptive systems literature [26, 28, 98] to generate optimal adaptation strategies. Then, Section 2.2 introduces the notion of machine-learning enabled systems, describing their architecture, goals, and current research challenges and directions. This section also describes common issues that may affect the predictions of ML models, thus hindering overall system utility. Finally, Section 2.3 describes the state-of-the-art regarding self-adaptation of ML systems, highlighting efforts along the following key dimensions: monitoring and analysis for detection of ML failures, planning of adaptation strategies to cope with ML failures, and current research efforts towards understanding the repertoire of tactics applicable to ML components, as well as how to estimate the costs and benefits of these tactics.

2.1 Self-Adaptive Systems

Self-adaptive systems (SASs) are systems capable of reacting to unpredictable changes in the environment in which they are operating and of adjusting themselves to their new environment [42, 52]. Adaptation can assume different forms, but the most common is reactive adaptation [96]. This corresponds to settings in which there are constraint/threshold violations that trigger adaption. Other types of adaptation correspond to proactive adaptation [125, 126] and homeostatic adaptation [70]. While the former triggers adaptation whenever the system expects something bad will happen, the latter triggers adaptation whenever system utility improvements from such adaptation are expected, to maintain stability and functionality in changing environments.

To self-adapt, these systems usually rely on a framework known as the MAPE-K control loop [90] that is divided into five components: Monitor, Analyze, Plan and Execute over a knowledge base K, as shown in Figure 2.1. Each of these components performs a critical task: *Monitor* is responsible for collecting relevant data from the managed system. One way for the *Monitor* to collect this data is through the use of sensors. *Analyze* is in charge of determining whether the system is behaving as desired or whether it needs to adapt to

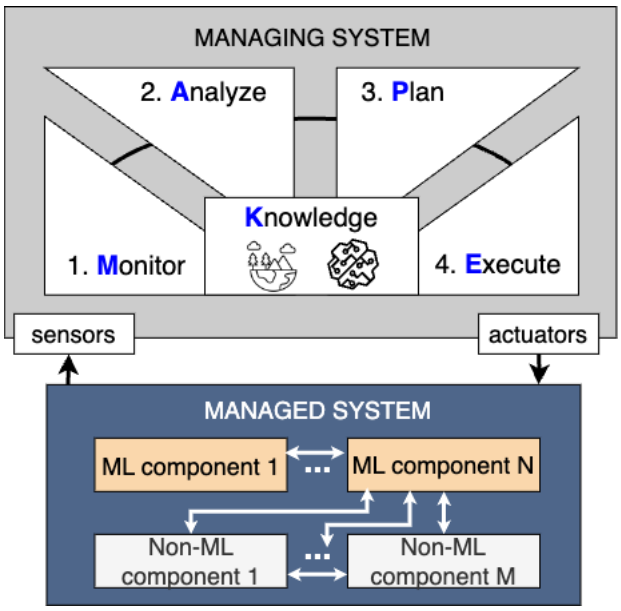


Figure 2.1: MAPE-K loop over an ML-enabled system.

improve its quality. *Plan* is responsible for determining what is the best course of action for the system to improve its utility, i.e., its quality. To do this, *Plan* has a set of actions that the system can enact to improve, and which are called *tactics*. *Plan* then reasons about the costs and benefits of each tactic and chooses an optimal adaptation strategy to execute. *Execute* is then in charge of carrying out the selected tactic and changing the managed system. *Knowledge* reflects what is known about the environment as well as about the managed system, i.e., what is monitored and actuated on. In practice, to build SASs and to generate optimal adaptation strategies, a common approach extensively used in the literature [26, 28, 98] is to use probabilistic model checking techniques.

Probabilistic model checking. Probabilistic model checking is a set of methods for reasoning about and analyzing systems that exhibit probabilistic and uncertain behavior. In order for probabilistic model checking tools to generate (optimal) adaptation strategies, it is necessary to instantiate a formal model of the system under adaptation, and to specify an adaptation goal in the form of a property (written as a temporal logic formula), which the model checker can verify. The result of the verification process is intrinsically dependent on the property, which can be of different types. The following are examples of types of properties that can be specified: verify whether some event occurs (e.g., whether there are deadlocks in a model); compute probabilities (exact, maximum, and minimum) of occurrence of an event (e.g., probability of message loss); estimate minimum/maximum (i.e., optimal) rewards of paths of a model. These techniques are a natural fit for planning adaptations in self-adaptive systems since they support proactive adaptation schemes such as *look-ahead* [125, 126]. This consists of having the model checker, via the formal model of the system, simulate the possible future states of the environment and of the

system to synthesize an adaptation strategy (sequence of adaptation tactics to execute) that maximizes system utility in the long term.

To specify the properties that the model checker verifies for optimality, one may resort to different logics, such as probabilistic computation tree logic (PCTL) [78], which is an extension of computation tree logic (CTL) [48]. CTL formulas are used by model checkers to verify system properties, typically expressed in terms of liveness and/or safety. PCTL extends CTL allowing the definition of formulas that account for probabilities associated with state transitions. By exploiting the fact that it is possible to associate rewards with specific states or state transitions in the formal model, and by specifying reward-based properties as a function of constraints of the system (e.g., system level objectives), the model checker generates optimal strategies that are expected to lead the system to a state that complies with the constraints.

This thesis leverages the PRISM model checker [99], which is a popular probabilistic model checker [28, 98, 127]. We define the formal models as markov decision processs (MDPs) [147], which model systems' dynamics through a set of states with probabilistic transitions and in discrete time-steps. More formally, a MDP is defined by a 4-tuple $(\mathcal{S}, \mathcal{A}, P_a, R_a)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, P_a is the probability matrix that gives the probability of transitioning to state s' from state s by executing action a at time t. R_a is the reward associated with this transition. Typically, there is a state in which the actions that the MDP can execute are the tactics available for adaptation. These tactics all have the same probability of executing but will lead the system to different states. It is the job of the model checker to test all possible paths, and hence all adaptation tactics, when verifying a given property (for example maximizing system utility). This entails that the model checker will output an execution trace that specifies the sequence of adaptation tactics that verify the property. We can thus extract the expected optimal adaptation tactic to execute from the sequence of adaptation tactics output by the model checker.

2.2 Machine-Learning-Enabled Systems

As defined by Lewis et al., an ML-enabled system is a "software system that relies on one or more ML software components to provide required capabilities" [105]. These systems leverage ML to adapt to changes, make predictions, or provide customized outcomes without the need for hard-coded rules. Differently, the behavior of traditional software components is explicitly defined via rules or algorithms. Thus, ML components are influenced both by the data available for their training and by their environment of operation. In fact, the environment has a significant impact in the ML component, as the ML model's predictive quality is influenced by the data it receives during inference (i.e., when it is queried in a production environment).

Additionally, the methods employed by the framework to enable ML adaptation borrow ideas from lifelong learning (LL), also known as *continual learning* (CL) in the Deep Learning community [109]. LL has been defined by Chen and Liu [41] as: "At any time point, the learner has learned a sequence of N tasks, $T_1, T_2, ..., T_N$. When faced with the

$(N+1)$th task T_{N+1}, the learner can leverage the knowledge in the knowledge base (KB) to help learn T_{N+1}. KB maintains and accumulates the knowledge learned from the previous N tasks. Once T_{N+1} has been learned, KB is updated with the knowledge gathered during the learning process." For instance, open-world learning[1] can be considered a form of lifelong learning. Ideally, in this setting, a learner would be able not only to learn continuously while it is operating in the environment, such that it can improve its performance on the tasks it has already learnt, but also recognize new tasks as they emerge and learn them.

Similarly, when leveraging the proposed framework to self-adapt ML-based systems, they become capable of continuously learning and modifying themselves according with the environment in which they are operating.

This section introduces and discusses environmental changes that lead ML models to mispredict, as well as typical sources for these changes.

2.2.1 Causes of Machine Learning Component Misprediction

ML approaches rely on a dataset composed of multi-dimensional input data and, in the case of (semi-)supervised models, labels. Since this dataset is used for training the ML model, the environment from which these data points are collected is usually known as the training environment. Then, once the ML model has been trained, it can be used by the system to make predictions[2] at run-time. This is typically considered the testing environment[3] as the model has never seen the current data points. We will use these notions of training and testing environments throughout this section to introduce typical causes of degradation of ML components.

It is generally assumed that the joint probability distribution $P(y, \mathbf{x})$ of labels y and multi-dimensional input data $\mathbf{x}$ (used to train the model) does not change between training and testing environments. Yet, when this assumption does not hold, and hence the prior distribution $P(\mathbf{x})$, the posterior distribution $P(y)$, or any of the conditional distributions $P(y|\mathbf{x})$ or $P(\mathbf{x}|y)$ changes, one may be in the presence of a problem commonly known as dataset shift (also known as data shift) [65, 149, 201]. We discuss the differences between these types of shift in more detail below.

Research efforts on detecting dataset shift [65, 135, 149, 150, 195, 201] are orthogonal to the problem we aim to solve, which focuses on determining when to adapt an ML component through the estimation of the costs and benefits of different ML adaptation tactics. However, the techniques to detect the different types of shift proposed by these works [65, 135, 142, 149, 150, 201] are valuable to our research as they can inform the decision process carried out by our adaptation framework. Specifically, we can use the

[1]When the data with which the ML model is trained is not available all at once, prior to deployment, and is instead made available continuously during run-time, online learning approaches are able to incorporate this new data into the ML model.

[2]We consider a prediction to be the output of an ML model. For example, for classification tasks, ML predictions are the probabilities of the input sample belonging to each possible class.

[3]This should not be confused with the testing environment in software engineering contexts: environment that replicates the production system and that is used to test the system before deploying it to production.

outputs of these methods, which monitor and detect changes in the ML components, as inputs to our decision process. These might constitute valuable features for deciding when to adapt an ML component since if these tools detect that the ML model's behavior has changed according to what they perceive to be "normal" behavior, then this might indicate a change in the underlying environment that calls for adaptation.

The following sections describe the most common types of data shift, and introduce typical sources of drift. These correspond to the typical causes of ML misprediction we are targeting with the proposed framework for ML adaptation.

Types of Data Shifts

The literature on ML has investigated several types of dataset shifts that have different characteristics. These different characteristics influence the impact that each type of shift has on a system, and also how easy it is to deal with/detect the shift. Specifically, problems such as anomaly detection, novelty detection, open set recognition, out of distribution detection, and outlier detection [195] are specific instances of the most common types of shift. We argue that the different types of shift described below are general enough to be representative of most of the issues addressed by the existing ML literature.

Co-variate shift. When the distribution of the inputs to a model changes, such that it becomes substantially different from the distribution on which the model was trained, we find ourselves in the presence of a problem commonly known as co-variate shift. That is, the distribution $P(\mathbf{x})$ changes but the conditional distribution $P(y|\mathbf{x})$ remains the same. More formally, $P(\mathbf{x})_{train} \neq P(\mathbf{x})_{test}$ and $P(y|\mathbf{x})_{train} = P(y|\mathbf{x})_{test}$ [129]. This type of shift is usually analyzed to evaluate how a model generalizes and how robust it is when the feature space is altered at test time, i.e. while the system is executing.

Prior probability shift (label shift). Differently, when we are in the presence of prior probability shift, also known as label shift, the distribution $P(y)$ of the labels/outputs has changed, i.e., in a classification task the class proportions differ between training and test. More formally, $P(y)_{train} \neq P(y)_{test}$ and $P(\mathbf{x}|y)_{train} = P(\mathbf{x}|y)_{test}$ [129]. This can be seen as the inverse of co-variate shift in the sense that the distribution of the labels changes, while the conditional distribution of x given y remains unchanged. Dealing with this type of shift is particularly challenging when the new distribution $P(y)_{test}$ is unknown.

Concept shift. Finally, concept shift corresponds to a change in the relationship between input and output distributions, although both distributions remains the same. More formally, when this type of shift occurs, we can have $P(y|\mathbf{x})_{train} \neq P(y|\mathbf{x})_{test}$ and $P(\mathbf{x})_{train} = P(\mathbf{x})_{test}$ or $P(\mathbf{x}|y)_{train} \neq P(\mathbf{x}|y)_{test}$ and $P(y)_{train} = P(y)_{test}$ [129].

Although these are the most common types of shift, it is also possible that other types of shift occur, for instance when both the conditional distribution and the features/labels distribution changes. Formally, this would correspond to $P(y|\mathbf{x})_{train} \neq P(y|\mathbf{x})_{test}$ and $P(\mathbf{x})_{train} \neq P(\mathbf{x})_{test}$ or $P(\mathbf{x}|y)_{train} \neq P(\mathbf{x}|y)_{test}$ and $P(y)_{train} \neq P(y)_{test}$ [129]. These types

of shift are typically less investigated in the literature since they are not so common in real world applications and also because they are extremely difficult to detect and deal with.

Sources of Data Shift

During regular system operation, shift in the data can occur due to: the passing of time, incorrect data or sample selection bias. Next, we provide details on these sources of shift.

Natural drift due to time. An effect of the natural passing of time is that people's tastes and behaviors change [19, 111]. For example, due to the passing of time a user may become interested in books/shows/music they were not interested in before. A static ML model, which is never adapted and does not account for these changes, will gradually start producing worse predictions. In such a scenario, we will be in the presence of concept shift.

Incorrect data. This problem arises when there are samples in the model's training set that are incorrectly labeled [189] or when test data is tampered with, thus leading the model to mispredict for inputs with specific characteristics. The former can happen for instance when unsupervised techniques are used to label examples in order to bootstrap the training set of a second supervised model [189]. Incorrect data can also make their way into a model's training set due to attackers that intentionally pollute it (e.g., by maliciously altering some of the input features) so as to cause the ML component to incorrectly predict outputs for certain inputs [74, 81]. Finally, noise and uncertainty, due to sensor errors or due to errors from upstream components, may also change the input data to a ML model, possibly causing label shift and mispredictions.

Sample selection bias. This occurs when selecting data points for a training set or when performing data cleaning[4]. When selecting data points, there may be environmental factors that cause some inputs or labels to be sampled more often. For example, when selecting participants for a survey, steps must be taken to ensure that the population of interest is accurately represented. Similarly, when performing data cleaning, for instance for a digit recognition task, less clear digits may be thrown away. However, this may prevent the model from learning that some digits are intrinsically harder to write than others [149]. During these, arguably critical, phases of model construction, sample selection bias will cause the training distribution to follow a different distribution than the test distribution, leading to data shift and to potential drops in ML predictive quality. Sample selection bias leads to co-variate shift [149].

2.3 Self-Adaptive Systems and Machine-Learning

This section presents existing work on the areas of machine learning and self-adaptation. We start by introducing works that call for the need to bridge the gap between these two

[4]In ML, data cleaning corresponds to the process of identifying and correcting errors in a dataset that may negatively impact a predictive model.

areas of research (Section 2.3.1), for example by evaluating how the variations in the accuracy of an ML model affect overall system utility [181]. Then, we introduce research efforts tackling the problems of monitoring ML models, detecting shifts and mispredictions (Section 2.3.2), followed by a discussion of works that test different ML adaptation tactics, such as retrain, and hyper-parameter tuning (Section 2.3.3). Section 2.3.4 overviews state-of-the-art approaches most closely related to our work. Finally, we conclude with a discussion of recent efforts in the area of sustainability (Section 2.3.5).

2.3.1 Bridging the Gap Between Self-Adaptive Systems and ML

For the past decade, researchers on ML and SAS have been arguing for the need to bridge the gap between these two research areas. For instance, Wagstaff [181] discusses how contributions in the ML field should be evaluated differently, so that real impact in the world can be measured. Specifically, they argue for the need to evaluate new approaches/algorithms in their real target domain of application, accounting for domain specific characteristics, and also that the evaluations should measure the approach's/algorithm's impact in terms of metrics such as lives saved, money spent/saved. In fact, the work by Bernardi et al. [14] is a great example of how system utility is affected by ML performance and how practitioners in a real setting go about evaluating their ML models. This is aligned with our definition of system utility and with our goal of adapting ML to optimize it.

Similar in spirit to Wagstaff, Bures argues for a new self-adaptation paradigm in which self-adaptation and artificial intelligence (AI) benefit from and enable one-another [24]. This is aligned with the vision for continual AutoML [56], which advocates for architectures that can manage ML systems and adapt them in the face of adversities (e.g., uncertainty, dataset shift, outliers). Finally, on-the-job learning [109] is another paradigm that is characterized by three main tasks, namely having an agent (e.g., ML-based system): discovering new tasks, gathering new data, and incorporating that data into its knowledge base and actions without interrupting the application.

Differently from ML research, which does not seem to be taking advantage of SAS research, the latter has been steadily taking advantage of ML techniques to improve the self-adaptation capabilities of systems [72, 84, 148, 156, 185, 186, 197]. Specifically, ML has been used in the adaptation manager to improve the Analyze and Plan components of the MAPE-K loop and to: update adaptation policies, predict resource usage, update run-time models, reduce adaptation spaces, predict anomalies, and collect knowledge [72]. The field of AIOps can be seen as an application of self-adaptation principles to improve systems in run-time, automating IT operations and systems [54]. These works demonstrate the relevance of our work on self-adaptive ML, and the need to bring together the ML and SAS research areas.

2.3.2 Monitoring Machine Learning Models

Regarding monitoring of ML models, below we overview related work along two key dimensions: research efforts targeting the problem of detecting data drifts, which may lead to ML mispredictions; and work that addresses the problem of performance estimation with

unlabeled samples (i.e., computing the predictive performance of an ML model without access to ground-truth labels).

Monitoring for drifts. Recently, a significant number of research efforts address the challenge of monitoring ML models, to detect data drifts and ML mispredictions. For instance, Zhou et al. [199] propose a framework to monitor ML models via data shift detection for data streams by leveraging information theory approaches to compare the distributions of key ML model features over weeks. Similarly, Pinto et al. [142] also propose an information theory-based method to detect shift in data streams, showing that it can accurately detect when an input signal has changed. In fact, detecting different types of shift is an active area of research in the ML domain [150, 195]. For instance, Rabanser et al. [150] evaluate the efficiency of different dimensionality reduction techniques along with different techniques for statistical hypothesis testing to evaluate the best combination to detect different types of data shift. These statistical and information theory based features form an initial set of informative features that can be leveraged both to detect the need for adaptation and to characterize the benefits of adaptation.

Recent work that also brings together ML and self-adaptation has: looked at monitoring deep neural networks (NNs) [192]; proposed a domain meta-model that instantiates the components required for monitoring supervised ML models and drift [95]; presented a framework that enables the creation of behavior oracles for learning enabled systems (LESs)[5], such that these oracles can then be used at runtime to predict an expected behavior category (e.g., correct prediction, misprediction) for a LES [102]; proposed a framework to monitor learning enabled components (LECs) of LES [103] by leveraging the LES behavior oracles [102]. This monitoring process allows LESs to reason about whether an LEC is expected to comply with the functional requirements. Ultimately, this allows stakeholders to understand how the system is expected to perform under varying environmental conditions. Recent work has also explored and proposed approaches to debug ML models and characterize the type of data for which they are expected to mispredict [47].

These works can thus be leveraged to detect when ML models are mispredicting, or are expected to mispredict in the near future, such that adaptation can be triggered. Our work goes beyond detecting mispredictions given that we aim to reason on which adaptation tactic to execute, and when, such that system utility is optimized.

Monitoring for real time predictive performance. Another relevant aspect of monitoring ML models is being able to compute the ML model's real time predictive performance. This may not be as straightforward, particularly in contexts for which ground truth labels may take a non-negligible time to become available. Without these labels, we only have access to a set of unlabeled samples, model predictions, and delayed (outdated) labeled samples, thus rendering it challenging to estimate a model's predictive quality in real-time. To address this challenge, the ML literature has proposed numerous approaches to estimate the quality of models' predictions [34, 67, 88]. Existing methods can be coarsely

[5]Learning enabled systems (LESs) rely on learning enabled components (LECs) to perform specific tasks – e.g., an object detector trained by camera images. LESs can thus be seen as our ML-based systems.

classified according to the type of ML models that they target (e.g., deep neural networks vs generic ML predictors) and to their ability to estimate aggregate quality on an entire (unlabeled) test set [76] vs identify misclassified test inputs [67].

These approaches typically leverage the probability distribution that a model outputs[6] to gain insights into the level of uncertainty associated with each prediction. This uncertainty thus constitutes a meaningful proxy to estimate model performance/error. For instance, the approach by Jiang et al. [88] targets deep neural networks and estimates the error on unlabeled test sets by measuring the disagreement rate of instances of the same network trained with a different run of Stochastic Gradient Descent (SGD). Another approach is to develop self-trained ensembles of deep networks that predict which inputs will be misclassified by the classifier based on (known) errors in the training dataset [34]. Other works, like average thresholded confidence (ATC) [67], take a model-agnostic approach and can estimate the correctness of individual model predictions — and then use the point-wise predicted labels to compute aggregate estimates of a model's predictive quality (e.g., class error rate) as well as of ground truth class probabilities.

By leveraging such techniques the framework proposed in this thesis can be applied to a wide range of domains without requiring assumptions on ground truth label availability, increasing its applicability and generalizability. In fact, Section 5.4 presents an extension of ATC [67], called CB-ATC, which is integrated in the framework and evaluated, along with ATC, in Section 6.2.2. Yet, the framework's accuracy is ultimately dependent on the accuracy of both (i) the predictors for estimating the benefits of executing each adaptation tactic and (ii) the approaches employed to estimate the current quality of the ML model.

2.3.3 Machine Learning Adaptation Tactics

The ML literature offers a plethora of work presenting approaches that can be regarded as ML adaptation tactics. Specifically, there have been works studying the effect of hyper-parameter tuning on ML models and trying to estimate whether specific models in a given system should be tuned given the expected benefits of doing so [114, 115, 154, 155, 176]. However, these works do not look into the problem of continuous adaptation, and are instead concerned only with determining whether a single, initial execution of hyper-parameter tuning is worthy given the expected improvement in ML accuracy. These approaches rely on classifiers that output a binary decision: perform or do not perform hyper-parameter tuning. However, to determine whether tuning is worthy, it is necessary to evaluate its impacts on overall system metrics, such as duration of the whole process. Some of these works discuss these trade-offs but do not decide to tune or not tune the models based on a general notion of system utility.

Research on ML has also proposed approaches to retrain[7] ML models [191] and research on SASs has also investigated the impact of retraining and incrementally training ML

[6]The model predicts a probability of each class being the correct class for the sample.

[7]Different flavors of ML model retrain can be considered, ranging from simply training the model from scratch with new data (this is what we typically consider as model retrain throughout this document), or more complex approaches that account for model architecture and hyper-parameter updates. All these variants yield different benefits to the system and have different costs.

models on training accuracy and training time [36]. This type of study has also been performed for the specific case of fraud detection systems [12, 104]. Finally, work on transfer learning [85, 137, 184] and machine unlearning [20, 29, 35] can also be leveraged as adaptation tactics for ML models.

However, differently from the work on hyper-parameter tuning and as far as we know, there are no approaches to estimate the benefits of tactics such as retraining, incremental learning, transfer learning, unlearning, or replacing the ML component by a non-ML component (e.g., rule-based model, geometric-based model for object detection). Indeed, to enable self-adaptation of ML models it is crucial to develop ways to estimate the benefits of the different tactics. In this work, we take a first step in this direction by estimating the benefits of retraining ML models.

2.3.4 Self-Adaptive Machine Learning

In the domain of self-adaptive systems, pioneering work by Chen explored the impact of retraining and incrementally training ML models via an empirical study, testing: methods under distinct domains, key performance indicators, and ML algorithms [36]. A particular relevant finding was that no method (retrain vs incremental train) was strictly better than the other. This highlights the need for the work developed in this thesis, in particular the need for approaches capable of estimating the benefits of different ML adaptation tactics.

Chen also presented a framework, LiDOS, for performing lifelong planning [37]. The base idea of this work is that the adaptation manager is continuously generating new adaptation plans that are tested on a twin of the real system. This twin is assumed to be an exact replica of the real system operating in the production environment, that is subject to the same (real) operational environment. Whenever there is an environmental change, the twin is notified and re-configures itself. By comparing the configurations of both the real and twin systems, it is possible to tell which configuration is most suitable for the current operational environment. The twin will keep re-configuring itself until a pre-defined number of alternative configurations has been tested. When this threshold is reached, the real system will be re-configured with the best configuration found. This approach seems to be applicable to ML-based systems despite only being evaluated with non-ML-based systems. Yet, having a replica of the deployed system and continuously testing different adaptation options before actually adapting the real system introduces costs and overheads, both computational and monetary, that may simply be too large to be acceptable in realistic settings. Should the cost be bearable in practice, this approach could provide benefits since the outcomes of the adaptation will be fully known (no uncertainty), thus leading to optimal adaptation decisions.

A similar research effort by Gheibi et al. [71] proposed a lifelong-learning architecture for SASs whose managing component relies on ML. The overall idea is to have a lifelong learning loop act as a meta-manager of the managing component. The meta-manager monitors and updates the ML component of the self-adaptation manager, preparing it to cope with concept shift. Throughout the lifelong learning loop, the ML components of the self-adaptation manager are updated whenever a new task is discovered. This work focuses on how to learn new tasks efficiently, by leveraging knowledge of previously learnt

tasks. Differently, we focus on the problem of learning *when* to adapt an ML model trained for a given task, that may be negatively impacting system utility due to data-shift. Theoretically, the framework proposed in this thesis could be leveraged to improve the lifelong-learning architecture of Gheibi et al. [71], thus only updating the self-adaptation manager's ML components when our framework deemed it beneficial.

More recently, researchers have proposed approaches that deal with adapting ML components of ML-enabled systems via model switching [97, 116], and explored how large language models can be incorporated into the adaptation loop, re-imagining self-adaptation [58]. However, most relevant to the work developed in this thesis are the works of Kulkarni et al. [97] and Marda et al. [116] that propose and instantiate, respectively, AdaMLS, an ML load balancer that dynamically switches the ML model being utilized, as a way to maximize system utility. Specifically, they assume the existence of a pool of ML models that have been analyzed offline, such that they can be grouped into clusters of similar performance. Then, whenever the monitored key performance indicators decrease below the desired values, the analyzer and plan stages search in the repository for new ML models that can bring the key performance indicators back to the desired levels. These works represent a step towards engineering self-adaptive ML enabled systems and constitute evidence of the success of ML adaptation. The work of this thesis complements these works by engineering self-adaptive ML-enabled systems for other domains and applications (fraud detection and machine translation), with different adaptation tactics (model retrain and model fine-tune). More importantly, this thesis proposes a generic and extensible framework that enables the creation of self-adaptive ML-enabled systems.

2.3.5 Sustainable Machine Learning

With the increased application and usage of ML-enabled systems, and especially since the advent of generative AI, we have seen a growing concern regarding the sustainability of AI approaches. Thus, research on sustainable AI has gained traction recently, highlighting the role that ML plays in global warming and calling for more accountability and transparency in reporting the environmental costs of model training [7, 13, 79, 86, 100, 165, 170, 179, 180]. Towards this end, research has focused on: understanding how to decrease AI's footprint by increasing its efficiency [180], studying the energy consumption associated with training/fine-tuning ML models [13, 165, 180], creating tools for self-reporting and evaluating the energy consumption and carbon emissions of training ML models [7, 79, 100] researching green AI adaptation tactics [86], creating systems that account for the trade-offs in ML model energy consumption when selecting which ML model to use [170], and adapting MLOps pipelines [15] with the goal of increasing their sustainability.

The work developed in this thesis complements these efforts since the proposed adaptation framework aims to prevent unnecessary adaptations of ML models via, for instance, model retrain or fine-tune by trading off the expected benefits of ML adaptation with its costs. Furthermore, we believe the proposed framework can be complemented by these works since it can be extended not only to leverage such green AI adaptation tactics [86] but also to consider sustainability-oriented system utility definitions [170].

2.4 Summary

This chapter presented related work along two main research areas, namely self-adaptive system (SAS) and machine learning (ML)-based systems. Specifically, we introduced the concept of SAS and probabilistic model checking, a technique commonly employed in state-of-the-art SAS systems to generate optimal adaptation plans. We then presented ML-enabled systems, discussing typical causes of ML misprediction, the most common types of data-shift, and providing examples of typical sources of data-shift.

We also overviewed existing work that calls for the need to bridge the gap between self-adaptive systems and ML such that both research areas benefit and leverage one another, provided background on existing efforts to monitor ML models and to improve them. These two research areas are valuable for the work done in this thesis as the techniques proposed can be employed to improve the adaptation loop by enabling more timely adaptations. Then, we discussed existing work that bridges the gap between SAS and ML. This discussion highlighted a gap in the literature: currently there is no work that analyzes the benefits and costs of ML adaptation tactics to decide whether adaptation of ML components is worth it. This literature gap, along with increasing concern regarding ML sustainability, inspired this thesis work.

The following chapter overviews the proposed framework for engineering self-adaptive ML-based systems, highlighting how it enables ML adaptation, leading to an optimization of system utility, and how it decides when and how to adapt the ML component.

Chapter 3

An Overview of Self-Adaptation for ML-based Systems

This thesis leverages formal methods to instantiate the problem of reasoning about the need for adaptation at a general architectural level. This section provides an overview of the framework proposed in this thesis (Chapter 5) to decide *when* and *how* to adapt ML components of ML-enabled systems. Specifically, the framework leverages (i) probabilistic model checking methods to generate optimal adaptation strategies and (ii) predictors to estimate the expected benefits of executing each tactic available to adapt the ML component. We start by introducing a simple use case to demonstrate the intended functionality of the framework, and then describe the modules that are instantiated in the formal model required by the model checker.

Running Example. Consider a system that receives jobs and has to select a platform for them to execute. As it is often the case in practice, we assume that the execution time of a job on a given platform depends on the job's characteristics, i.e., it may execute faster on a platform than on another [6, 31]. Each time a job completes, the system receives a fixed reward. As such, in a given period, the system will strive to complete as many jobs as possible by selecting the platform that can execute each incoming job in the shortest amount of time, so as to accrue the maximum benefits possible. For example, the system could receive different data analytic jobs with diverse characteristics (e.g. neural network (NN) training, data stream processing) [6, 31]. The distribution of jobs in the environment thus has an intrinsic uncertainty that the ML scheduler needs to navigate.

The system then relies on an ML component to decide the best platform for a specific job to execute in. For instance, the training of a neural network can be offloaded to GPUs or CPUs. While both platforms allow the system to complete its task (i.e., execute the job) one platform may be more efficient (lower latency) than the other, thus allowing the system to complete more jobs in a given horizon. We are interested in scenarios in which the type of job generated by the environment is altered, for example due to dataset shift [149], thus leading the ML model to offer worse predictive quality [94].

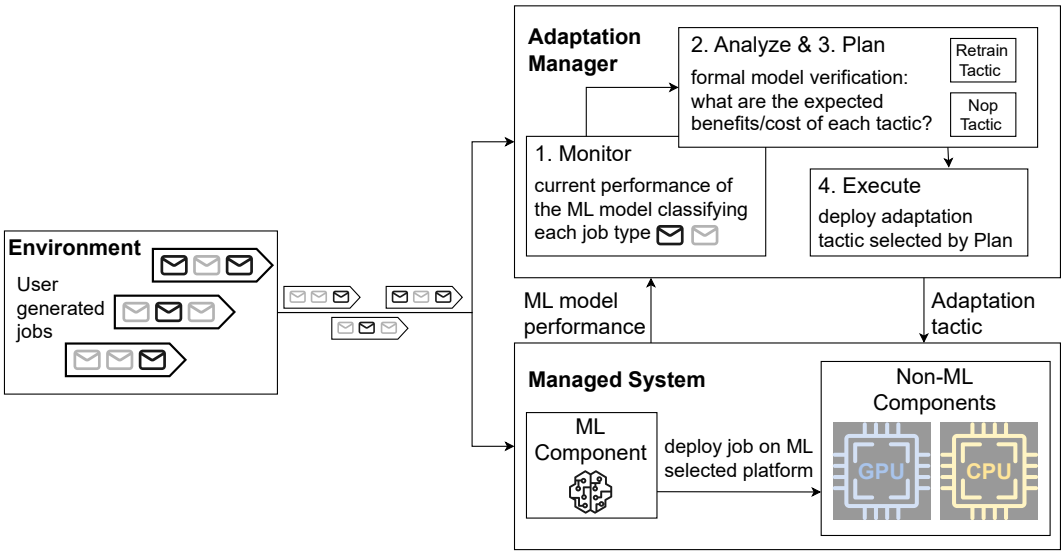


Figure 3.1: Self-adaptive scheduler system. Different job types are represented by the black and gray messages/envelopes.

Machine Learning Adaptation. To equip the system with adaptation capabilities, so that it can deal with environment changes and accuracy fluctuations of the ML model, for instance due to unknown jobs (e.g., unseen neural network topology), we consider that each time a new job arrives, the adaptation manager can decide between simply querying the current ML model (i.e., no adaptation – tactic *nop*) or adapting it (e.g., via tactic *retrain*) to increase its predictive quality and maximize the likelihood of executing the job in the preferred platform. Figure 3.1 provides a visual representation of the self-adaptive scheduler system. The *retrain* tactic is one example of ML adaptation tactics that can be considered (Section 4.1 presents a repertoire of tactics for ML adaptation) and consists of incorporating additional training data in a new version of the model [190]. However, the execution of this tactic has non-negligible latency (its effects in the system are not immediate), and it has a monetary cost (e.g., if retrain is performed in the cloud) [6, 31]. As such, for this use-case, we consider system utility as the sum of the benefits of completing jobs minus the cost of executing a tactic (tactic *nop* has no cost).

Formally Modeling the ML-based system

To decide *how* to adapt the ML model, the proposed framework leverages probabilistic model checking methods to decide which adaptation tactic will optimize system utility the most. By employing probabilistic model checking methods, we can not only account for the intrinsic uncertainties in the system's environment of operation, and in the prediction of the expected benefits of each adaptation tactic, but also reason at a general architectural level about the expected impact of ML predictions on overall system utility.

However, the model checker needs a formal model of the system being verified, so we first need to create its formal model. Then, we need a logic property that encodes the

goal of the system, i.e., its system utility definition. The model checker then verifies which execution paths lead the system to optimize the property (i.e., system utility). This process will ultimately lead to finding the adaptation tactic that optimizes system utility.

Creating the formal model requires thinking about the properties of the system that impact system utility. For instance, the jobs that are generated by the environment and are sent to the system will influence system utility. Thus, this has to be modeled. Similarly, since the framework adapts ML components, and that the predictive quality of the ML model influences the platform where the jobs are executed, this will also have to be modeled. Finally, the job execution also needs to be modeled because we need to simulate that a job has finished executing to collect the benefits of its execution. Below, we briefly describe how we model each of these components.

Environment. We model the environment as generating two types of job (J1 and J2) according to probability *pJob1* (or *pJob2*=1-*pJob1*). Although it could be trivially extended to generate more job types, having only two is enough for the purpose of illustrating reasoning about whether to adapt the ML component.

Adaptation manager. The adaptation manager (i.e., the managing system, cf. Section 2.1) is responsible for triggering adaptations. In this simple running example, two tactics are available: *nop* (no operation) and *retrain*. When the system receives a new job generated by the environment, the pre-condition for the tactics' execution becomes true. At this point, the model checker, when asked to synthesize optimal adaptation strategies, decides between *retrain* and *nop* and triggers the corresponding tactic in the ML component. The latency of the *retrain* tactic is accounted for by the ML component during tactic execution. The tactic's cost is subtracted from the system's rewards when the job completes its execution.

Non-Machine-Learning component. A non-ML component is responsible for simulating the execution of the jobs. When the environment generates a new job, and after the adaptation manager has selected the adaptation tactic to execute and adaptation has taken place, the executor will deploy the job on the selected platform. To simulate the job's execution, the non-ML component which has two platforms at its disposal (GPU or CPU), needs to know the latency of the job. As there is intrinsic uncertainty in determining job execution latency, we assume the existence of historical data which can be used to construct distributions of possible execution latencies. These distributions can be discretized (as shown in Table 3.1) and encoded in PRISM so that model checking is tractable and this uncertainty is explicitly modeled. Whenever a job completes, the utility of the system is updated by adding the job completion reward and subtracting the tactic execution cost.

Machine Learning component. In this use case, since there are two possible execution platforms and two input job types, we represent the predictive quality of the ML model via two binary confusion matrices: one for each type of job. Additionally, to account for the new data that the ML model is continuously gathering (e.g., after a job completes,

Table 3.1: Discretized distribution of job latency for each job type, in each platform, and corresponding likelihood. Each cell in each matrix has the probability of the corresponding PRISM transition. That is, in Table 3.1a with probability 18% the predicted latency for a job is 3 on platform 1 (P1) and 8 on platform 2 (P2). If, for this situation, the ML component is very accurate, it will select platform 1 to deploy the job, since P1 has the lowest latency. Note that the ML model has different predictive quality for each job type.

(a) Job latency depends on the platform in which it is executed. Thus, in this case, ML accuracy has an impact on system utility.

P1 \ **P2**	lat.	6	8	10
lat.	prob.	20%	30%	50%
3	60%	12%	18%	30%
5	30%	6%	9%	15%
7	10%	2%	3%	5%

(b) The diagonal accounts for more than half of the probability, thus a job's latency will likely be the same regardless of the execution platform. Hence, ML accuracy should have little impact on system utility.

P1 \ **P2**	lat.	3	5	7
lat.	prob.	15%	70%	15%
3	15%	2.25%	10.5%	2.25%
5	70%	10.50%	49.0%	10.50%
7	15%	2.25%	10.5%	2.25%

the ML model has an extra sample from which it can learn), we count the inputs of each job type that are received and use this count as a proxy for the amount of information encoded in these new inputs. This count is increased whenever a new job arrives and reset whenever the *retrain* tactic is executed. This information then contributes to the impact of the *retrain* adaptation tactic on the ML component.

Specifically, we consider a simple model that aims to capture the changes to the confusion matrix upon the execution of a *retrain* tactic. The intuition behind this model is that the larger the number of new samples seen since the last training (i.e., new data), the larger should be the expected reduction in the misclassification rate. The confusion matrix should be updated as follows. The diagonal is incremented by a factor $\delta = (100 - cell_{ii}) * new\_data * impact\_factor$ that is proportional to the model's loss and to the amount of new data (e.g., number of new samples). The *impact_factor* adds flexibility to the update model to account for different types of retrain (e.g., when the hyper-parameters of the model are also updated, the benefits may be higher). The remaining cells in the same row should then be updated as $cell_{ij} = cell_{ij} - \delta cell_{ij}/(1 - cell_{ii})$. The non-diagonal cells (which correspond to scenarios of mispredictions of the ML component) are thus reduced proportionally to δ while ensuring that no cell gets a value lower than 0, and that the total reduction on non-diagonal cells is equal to δ. Table 3.2 provides an example of a confusion matrix being updated.

Optimizing system utility. Since the system receives a fixed reward whenever it completes a job, its goal is to maximize the number of jobs completed in a given time period, while simultaneously minimizing the costs spent on retraining. This requires seeking an

Table 3.2: Visualization of an update to a confusion matrix (CM). We assume $new\_data = 6$ and $impactFactor = 0.1$ which yields $\delta = (100 - 95) \times 6 \times 0.1 = 3$. P1 and P2 stand for Platform 1 and Platform 2, respectively.

Pred. \ **Real**	P1	P2
P1	95	5
P2	5	95

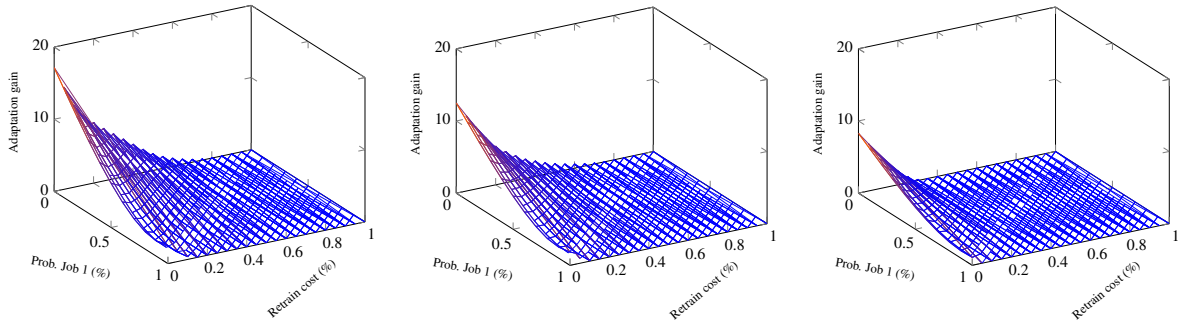
(a) Initial CM.

P \ **R**	P1	P2
P1	95 + 3	5 - $\frac{3\times5}{(100-95)}$
P2	5 - $\frac{3\times5}{(100-95)}$	95 + 3

(b) CM update.

P \ **R**	P1	P2
P1	98	2
P2	2	98

(c) Final CM.



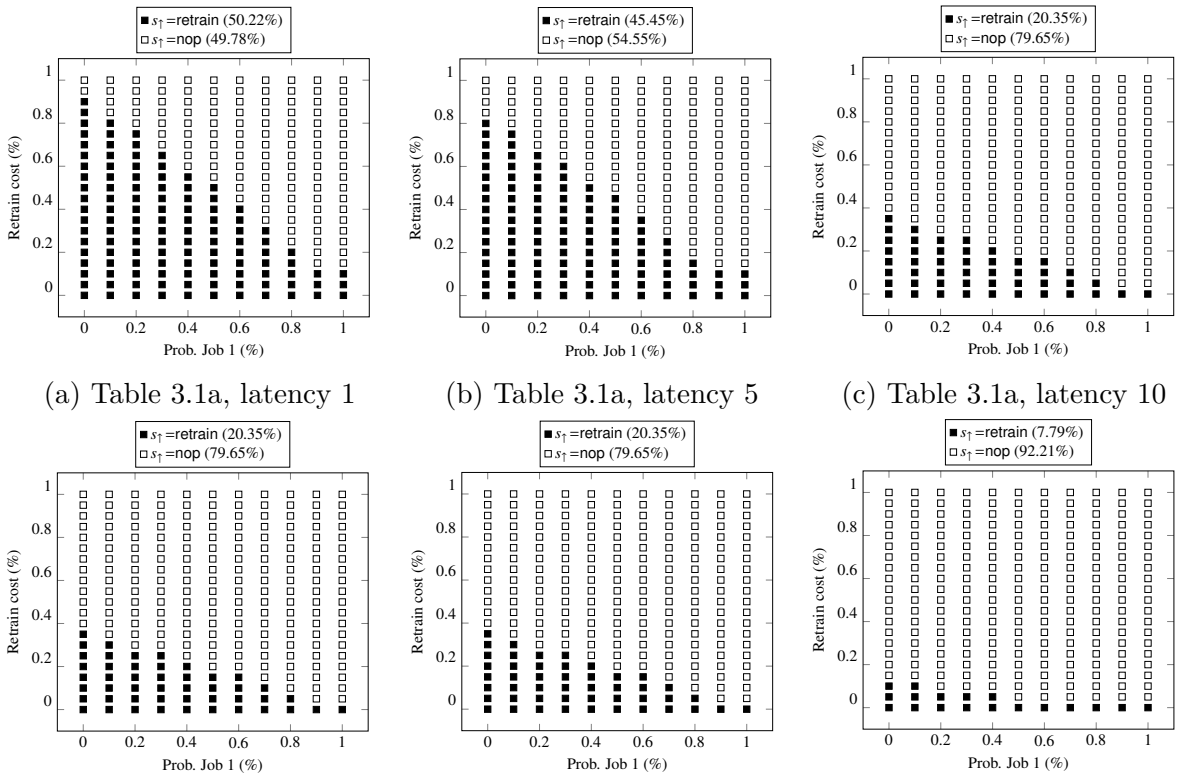
(a) Latency 1 (b) Latency 5 (c) Latency 10

Figure 3.2: Utility gains achievable due to ML adaptation when the system operates in an execution context in which ML accuracy impacts system utility.

adequate trade-off between investing time and resources to retrain the model and reasoning about the expected impact of the current accuracy on system utility. To do so, we encode in the model checker a logic property that ensures that system utility is maximized (in expectation) when the state "end" is reached, i.e., when the time period expires.

Below we analyze the system utility improvements obtained with the proposed framework via two research questions:
RQ1. What are the estimated utility gains achievable through ML adaptation?
RQ2. Under what conditions does the framework determine that ML adaptation improves overall system utility?

Experimental settings. For the experiments we vary the: (i) retrain cost; (ii) retrain latency (1, 5, 10); and (iii) the probability (from 0 to 1 with 0.1 increments) of the environment generating each job type. Our goal is to model how the system reacts to environment changes, so we assume that the ML model has better predictive quality for one type of job than for the other: 95% accuracy for type 1 jobs and 50% accuracy for type 2 jobs (these are symmetric confusion matrices). Thus, when the environment generated jobs change,



(a) Table 3.1a, latency 1 (b) Table 3.1a, latency 5 (c) Table 3.1a, latency 10

(d) Table 3.1b, latency 1 (e) Table 3.1b, latency 5 (f) Table 3.1b, latency 10

Figure 3.3: Areas of the space in which ML adaptation improves overall system utility. Black squares correspond to configurations in which it is worth adapting. White squares correspond to configurations in which having the option to adapt does not improve utility. The top row corresponds to the execution context of Table 3.1a and the bottom row to the execution context of Table 3.1b.

the system is affected. The impact factor is set to 0.1 for both job types. Job latency and uncertainty in each platform are set according to Table 3.1.

Results. Figure 3.2 shows the utility gains achievable due to ML adaptation, for an execution context in which ML accuracy affects system utility. Each sub-plot considers a different latency for executing the retrain tactic. The results show that for the retrain latency values considered, there are always system utility gains to be accrued due to ML adaptation. Next, we show how the framework can distinguish when to execute each adaptation tactic, seeing as determining *when* to adapt is a critical aspect of the framework.

The plots in Figure 3.3 consider different execution contexts and retrain latency, and show the conditions under which ML adaptation improves overall system utility. We now focus on Figure 3.3a and analyze the impact of the retrain cost and job probability variables. As expected, when retrain cost is low, the framework determines that adaptation is always worth it. Yet, as the cost increases, if the environment is more likely to generate jobs of type 1, retrain is no longer the optimal action. This is due to the fact that the ML model

has a good predictive quality for jobs of type 1. Thus, retrain costs outweigh the benefits. Differently, when the environment generates more jobs of type 2 (prob. Job 1 $< 50\%$), the tolerated cost of the model retrain increases. As tactic latency increases (Figures 3.3b and 3.3c), we see that model retrain is worth it only for lower adaptation costs.

The difference between the figures in the top row (Figures 3.3a, 3.3b,3.3c) and the figures in the bottom row (Figures 3.3d, 3.3e,3.3f) is the execution context of the system, that is, the latencies of the jobs in each platform and their probabilities (which are set according to Table 3.1). For the bottom row it is more likely that a job has the same latency regardless of the platform (Table 3.1b). In such a situation, having an inaccurate ML model has little impact on system utility. The comparison between top and bottom rows demonstrates this effect: for the bottom row plots, adaptation pays off only in very few scenarios and only when tactic cost is low. In fact, we see that when latency is high (Figure 3.3f), the cost of retrain has to be close to zero for adaptation to provide benefits.

Overall, these results confirm that ML adaptation improves overall system utility (RQ1) and that the proposed framework distinguishes *when* and *how* to adapt the ML model, trading-off the expected costs and benefits of each adaptation tactic (RQ2).

3.1 Summary

This chapter provided an overview of our approach to self-adaptation for ML-based systems, along with preliminary evidence showcasing the benefits in terms of system utility that may be expected by leveraging informed adaptation strategies.

Although the simple running example considered in this chapter used a simplistic linear model to predict the effects of retrain, which ignores the characteristics of the data available to execute each tactic and only looks at how much new data is available for retrain, Chapter 5 will present more complex predictors which are later evaluated in Chapter 6.

However, before delving into complex predictors, we focus on a key building block for enabling self-adaptation of ML-based systems: *how to adapt*. Thus, the following chapter presents: (i) a non-exhaustive repertoire of adaptation tactics for ML models; (ii) a description of each tactic along with examples of when each can be applied and a brief discussion of their advantages and disadvantages; and (iii) a discussion of the research challenges and opportunities that arise in each of the MAPE-K loop stages when engineering self-adaptive ML-enabled systems.

Chapter 4

Self-Adaptation for Machine Learning-based Systems

In recent years, machine-learning-based systems (MLSs) have become ubiquitous in domains such as enterprise and cyber-physical systems. MLSs are composed of one or more machine-learning components (MLCs) whose behavior is derived from training data [103]. MLCs are then embedded into a larger system containing traditional computational entities such as web services, databases, and operator interfaces. Examples include: fraud detection, which uses a classifier to detect fraudulent transactions [21]; medical diagnosis, which relies on ML for classifying types of diseases of patients [62]; self-driving cars, which use ML to determine whether they should brake to avoid collision with moving objects (e.g., cars, pedestrians) [10, 113]; robots, which rely on ML models to predict the amount of remaining battery power [84]; targeted advertisement services, which rely on recommender systems to show users items that they may find interesting [43]; and smart homes/buildings, that rely on MLCs for tasks such as face and voice recognition [77, 166] or occupancy prediction for proactive heating/cooling [63, 82].

Despite their widespread use, and similarly to non-ML components, MLCs can fail to perform as expected [47, 103, 193], thus reducing system utility. For example, changes in a system's operating environment can introduce drifts in the input data of the MLCs, or attacks may attempt to subvert the intended functionality of the MLC [74].

While the literature offers no immediate solution to guarantee the behavior of MLCs under changing environments [103], the large body of works on self-adaptive systems (SASs) has already investigated a number of methods to tackle analogous problems for non-ML enabled systems. The work developed in this thesis aims to bridge this gap by using self-adaptation to deal with environment changes and faults in the context of Machine-Learning Enabled Systems and machine-learning components.

In fact, there is a large number of emerging techniques that have been developed by the ML community for improving and correcting supervised ML models and that could be used as adaptation tactics in a SAS. These range from off-line, full model retraining and replacement, at one extreme, to incremental approaches performed *in-situ*, at the other [29, 81, 122, 142, 150, 190].

Unfortunately, determining *when* and how to take advantage of such tactics to perform adaptation is not trivial. First, there is a myriad of possible adaptation tactics that could potentially be applied to an ML component, but not all approaches work with all forms of ML models. For example, when a new family is moving into a new smart home, all the MLCs used by the smart home system should be fine-tuned to work as expected for that family [19]. Transfer learning [136] could be leveraged in this setting to speed up the process of fine-tuning these models. Yet, this tactic would yield the expected benefits only if the families are similar [130].

Second, the value of investing in improving the accuracy of a ML component is strongly context-dependent — often depending both on domain and timing considerations. For example, while a medical diagnosis system may support model retraining at run time, the latency of this tactic may make it infeasible for self-driving cars, which might rely instead on swifter tactics (such as replacing the ML component entirely) to match real-time response requirements. In a different mode of operation, however, both types of tactics may be available, e.g., if the self-driving car is stopped (parked mode of operation), it may be feasible to retrain an under-performing model without compromising safety.

Third, calculating the costs and benefits of these tactics is difficult, particularly in a whole-system context, where improving a specific component's performance may or may not improve overall system utility. Costs include time, resources (processing, memory, power), and service disruption. Benefits derive for instance from increased accuracy or fairness of the ML component, which can in turn lead to better performing down-stream components and support overall business goals (e.g., by improving advertisement revenue). Yet, both costs [31, 118] and benefits [36] can be hard to quantify, thus making it challenging to reason about whether executing an ML adaptation tactic will improve system utility.

We argue that to harness the potential of the rich space of ML adaptation mechanisms, it is necessary to: i) investigate what tactics are available to adapt MLCs and ii) develop methods to: i) identify which tactics, among the ones available to adapt a MLC, are the most effective in a given context to maximize system utility, and ii) integrate them into modern self-adaptive systems architectures. Note that the repertoire of ML adaptation tactics grows as new ML methods are developed and ML models can be modified and improved with alternative approaches and targeting different quality attributes (e.g., fairness, sustainability).

Thus, in this chapter we start by investigating what tactics are available to adapt ML components that are deteriorating system utility (Section 4.1). Then, in Section 4.2, with the aid of a use case from the fraud detection systems' domain, we provide examples of the different causes of system utility degradation presented in Section 2.2.1 and the applicability of the tactics introduced in Section 4.1. Finally, in Section 4.3, we discuss which changes/updates to the popular MAPE-K loop must be enacted in order to engineer self-adaptive ML-based systems.

Table 4.1: Examples of general adaptation tactics for ML-based systems with their strengths ('+') and weaknesses ('−').

Tactic	Description	Properties
Component Replacement	Replace an under-performing component by one that better matches the current environment	+ Fast and inexpensive, when possible – Alternative components may not be available in all scenarios – Alternative estimators, when available, may be more robust but less precise
Human-based Labeling [122]	Rely on a human to classify incoming samples or to correct the labeling of samples in the training set	+ Accuracy of human-based labels expected to be high – Expert knowledge may be expensive to obtain and/or introduce unacceptable latency
Transfer Learning [136]	Reuse knowledge gathered previously on different tasks/problems to accelerate the learning of new tasks	+ Less data-hungry than plain retrain – Effectiveness dependent on the similarities between old and new tasks/data – Computationally intensive process
Unlearning [29]	Remove samples that are no longer representative from the training set and from the model	+ Fast when ratio between data to forget and data-set size is small – Cost/latency for identifying examples to unlearn can be large and context-dependent
Retrain [190]	Retrain with new data and maybe choose new values for the ML model's hyper-parameters	+ Generic and robust method – Computationally intensive process – Accuracy and latency of the retrain process may vary significantly – Effective only once a relatively large number of instances of the new data are available

4.1 Adaptation Tactics

Inspired by the machine learning (ML) literature [29, 122, 136, 163, 190], in this section we provide a non-exhaustive list of ML techniques that can be leveraged as adaptation tactics when engineering self-adaptive ML-enabled systems. Specifically, Table 4.1 contains a collection of tactics that can be used to deal with under-performing ML-based components. These performance issues can be caused by the different types of shift previously introduced (see Section 2.2.1). Next, we provide a high-level description of the tactics presented in the table, discussing their costs and benefits, and motivating them in Section 4.2.

Component replacement. This tactic assumes the existence of a repository of components and respective meta-data that can be analyzed to determine if there exists a component that is better suited for the current system state, i.e., that is expected to lead to a higher system utility. If such a component exists, this tactic will replace the under-performing component by the one that is expected to be better. A benefit of this tactic, whenever it is available, is to enable a swift reaction to dataset shifts. Its main cost depends

on the latency and resources used for the analysis of the candidate components available in the repository. Additionally, alternative components may not always be available and, when they do exist, they may be less precise albeit more robust.

Human-based labeling. Humans are often able to recognize patterns, problems, and objects more accurately than ML models [122]. Thus, depending on the domain, humans may play a role in correcting these components or giving them correct samples [122, 183]. For example, when an ML component is highly uncertain about a specific input, it may rely on a human to provide a label. Similarly, if incorrect data is found on a model's dataset, a human may be asked to correct those samples. While this tactic may provide high benefit if the human is an expert, it also has a high cost, since humans are expensive. Additionally, if there is a significant amount of samples to label, the latency of the process may be unacceptable. It is also possible to rely on the actual system users (e.g., drivers of self-driving cars, or holders of credit cards), prompting them for an answer in particular situations (similar in spirit to captchas). However, this burdens the users who may become frustrated at the system and become less willing to use it.

Transfer learning. transfer learning (TL) techniques leverage knowledge obtained when performing previous tasks that are similar to the current one so that learning the current task becomes easier [110, 136]. For this tactic to be applicable, it is necessary to evaluate the similarity between the source and target tasks/domains. Transferring knowledge between dissimilar tasks will likely not provide benefits. In order to compute this similarity, metrics such as LEEP [130] can be used. The advantages of this tactic are that it requires less data than is needed to (re-)train a model from scratch, thus allowing for a quicker model initialization phase. However, the process of TL, similarly to a model (re-)train, is also computationally intensive.

Unlearning. This tactic corresponds to unlearning data that no longer reflects the current environment/state of the system and its lineage, thus eliminating the effect of that data on current predictions [29], while avoiding a full model retrain. A key problem that stands in the way of the execution of this tactic is the identification of incorrect labels. In scenarios in which the identification of incorrect samples is not readily available, one may leverage automatic techniques, such as the one described in [30], which are faster but typically less accurate than relying on humans. As such, the cost and complexity of this adaptation tactic vary depending on the context. Then, after identifying the incorrect samples, the model must be updated to accurately reflect the correct data. The advantage of unlearning techniques with respect to a typical full model retrain is the time savings (up to several orders of magnitude [29]) that can be achieved.

Retrain and/or hyper-parameter optimization. This is a general tactic that involves retraining the model with new data reflecting recent dataset shifts. There are many types of retraining, ranging from a simple model refresh (incorporate new data using old

hyper-parameters), to a full retrain (including hyper-parameter optimization, possibly encompassing the search for different model types/architectures [60]). These imply different computational costs and lead to different benefits in terms of model accuracy improvements. In the presence of dataset shifts, when there is new data that already incorporates the new input distribution, this tactic often represents a simple, yet possibly expensive, approach to deal with this problem. Yet, this tactic usually requires a substantial amount of data to yield highly accurate models and is computationally intensive. Its benefits are dependent on the type of retrain process and on the quality of the new data. As for its cost, if retraining is performed on the cloud, it can be directly converted to the economic cost of the virtual machines. Several techniques exist to predict such costs [6, 31, 118, 194].

4.2 Adaptation of ML-based Enterprise Systems

We now motivate the need for self-adaptive machine-learning enabled systems through an example from the enterprise systems domain. We provide examples of situations in which each type of shift can occur and of scenarios in which each repair tactic can be applied.

4.2.1 Fraud Detection System Use Case

Consider a fraud detection system that relies on ML models to determine whether credit/debit card transactions are legitimate or fraudulent. These ML models typically attribute a score to each transaction, which corresponds to the likelihood of the transaction being fraudulent [142]. The score attributed by the ML model is then used by a rule-based model to decide whether transactions are legitimate or fraudulent. Typical clients of companies that provide fraud detection services are banks and merchants. In this setting, system utility is typically defined based on attributes such as the cost of losing clients due to incorrectly declined transactions, fairness (no user sees their transactions declined more often than others) [50] and the overall cost of service level agreement (SLA) violations (these systems have strict SLAs to process transactions in real time, e.g., at most 200ms on the 99.999th percentile of the latencies' distribution [21]).

While cost and revenue are directly affected by the ML model's mispredictions, response time is affected by model complexity, i.e., more complex ML models may introduce higher prediction latencies that compromise SLAs. Thus, when adapting an ML component in this domain, it may be necessary to account for the impact of increased ML model complexity on the fulfillment of the SLAs. Further, the impact of ML component mispredictions varies not only from client to client, with whom different SLAs may have been agreed upon, but also in time, since during specific periods, e.g., Black Friday, the volume of transactions substantially increases [111]. During busy days such as these, since there is an increase in the number of legitimate transactions and the spending patterns are altered, it is crucial that ML models are less strict and reduce false alarms. At the same time, having less strict models may lead to more fraudulent transactions being accepted. Hence, mispredictions come at huge penalties and a delicate trade-off is required to ensure an acceptable system

utility. Finally, these systems are subject to constantly evolving fraud patterns, to which the ML components must adapt [8].

4.2.2 Causes of Degradation of ML Components' Accuracy

This section illustrates each type of data-shift with examples based on the fraud detection system use case.

Co-variate shift. In a fraud detection system, co-variate shift occurs when patterns of legitimate transactions change, for instance due to busy shopping days like Black Friday and Christmas [8]. Although the actual features used for classification may not change, their distribution does. For example, suppose a user usually purchases items online from shop **A**. The distribution of fraud given the feature *online shop* is 10% for shop **A**. The user then discovers that shop **B** sells the same items but at a cheaper price, so they start purchasing from shop **B**. Additionally, the distribution of fraud given feature *online shop* is also 10% for shop **B**. In this scenario, the distribution of the feature *online shop*, given by $P(\mathbf{x})$ is altered, but the distribution of fraud given the feature, $P(y|\mathbf{x})$, remains the same, i.e., the amount of fraud in shops **A** and **B** is the same regardless of where the user buys. In a case such as this, the fraud detection system may suspect the change in the user's behavior to be fraud because it learned that the user typically buys from shop **A**.

Prior probability shift (label shift). In the context of fraud detection systems, this type of shift occurs for example when the proportion of fraudulent transactions in the training set is different than for the test set [111], i.e., $P(y)_{train} \neq P(y)_{test}$. This type of shift requires assuming that the distribution of input data given fraud, $P(\mathbf{x}|y)$, does not change between training and testing environments. This corresponds to a mathematical abstraction over the power of an adversary that is capable of generating fraudulent transactions that follow the same pattern as legitimate transactions. Since the model has learned the typical distribution of fraud, it's predictions will follow that distribution, which is no longer representative of the system's environment.

Concept shift. This is the most common type of dataset shift in the fraud detection domain and occurs when fraudsters adapt their strategies and new fraud patterns emerge, such that the ML model is no longer able to effectively distinguish fraudulent from legitimate transactions [111]. In this case, the distribution of fraud given input data $P(y|\mathbf{x})$ changes while the distribution of input data $P(\mathbf{x})$ remains the same.

4.2.3 Repair Tactics

This section motivates the applicability of each repair tactic by providing examples of their usage when different causes of degradation have affected the system's ML component.

Component replacement. When the volume of transactions changes, for instance during special days such as Black Friday, ML models may consider the increased frequency of transactions as an indicator of fraud and erroneously flag legitimate transactions as fraudulent. Such mispredictions can lead to significant financial losses [21], thus requiring timely fixes that render the use of high latency tactics infeasible (note that in this context transactions need to be accepted/rejected within a few hundreds milliseconds [21]). As such, only low latency tactics can be applied. An example is to replace the under-performing models with rule-based models, e.g., developed by experts for specific situations, and/or to switch to previously trained models that are known to perform well in similar conditions.

Human-based labeling. Whenever the ML component suspects a transaction of being fraudulent it can automatically block that transaction. Then, the user can be informed of the decision and asked whether the transaction should be authorized or declined in the future. Another possibility is to add humans to the loop when adding samples to the ML component's training set. In this scenario, an expert can be asked to review the most uncertain classifications so as to improve the quality of the training samples.

In the former scenario, the benefits are easily quantifiable, since the risk of accepting a possibly fraudulent transaction can be measured via its economic value. However, users may get annoyed if their transactions are canceled too often, to the extent that they may stop purchasing using that credit card provider. As for relying on experts to review uncertain classifications, having an on-demand expert performing this task is expensive and the latency of the manual labeling process may be unacceptable[1].

Transfer learning. Suppose that: (i) a fraud detection company has a set of clients (such as banks), (ii) the company has a unique ML model for each client, so that it complies with data privacy regulations, and (iii) one of its clients is affected by a new attack pattern, which is eventually learned by that client's model. In this scenario, TL techniques can be used to improve other clients' models so that they can react to the same attack pattern. In fact, since privacy is paramount in this domain, there are techniques that can be used to deal with the problem of ensuring data confidentiality and anonymity in information transfer between clients [66, 110] instead of typical TL techniques that do not provide this assurance [136]. Estimating the benefits of executing this tactic for a given client boils down to estimating the likelihood that this client may be targeted by the same attack, which comes at an added cost and time. Yet, the execution of this tactic typically implies high computational costs (e.g., if cloud resources are used) and non-negligible latency, which may render this tactic economically unfavorable, or even inadequate, e.g., if the attack on a different client is imminent and the TL process is slow.

Unlearning. In the domain of fraud detection, if after a specific amount of time (e.g., 1 month) the fraud detection system does not receive complaints about a set of transactions, these will be labeled as legitimate. However, since users typically take a long time to

[1]Fraud detection systems normally rely on a fixed set of humans at any given time. This determines a maximum load of transactions that can be processed with a human in the loop.

review their statements and to complain when they do not recognize some transactions, it is possible that there are incorrectly labeled transactions in the dataset. In this scenario, and if a model has been trained with incorrect samples, it is possible to leverage this tactic to remove the incorrect samples from the model without requiring a full model retrain.

Retrain and/or hyper-parameter optimization. Full model retraining can be leveraged for example when there is a new fraud pattern for which there is already enough data for the model to learn from. By retraining the model with this data, the model will likely detect an increased amount of fraudulent transactions, thus also increasing system utility. However, as this is a slow tactic, throughout its execution system utility is likely to either drop or remain as unsatisfactory as it was prior to the execution of the tactic. To prevent such situations, hybrid planning approaches can be leveraged [138] to execute a swift tactic that slightly improves system utility while the slow tactic is executing.

4.3 MAPE-K Loop for ML-based Systems

In SAS, the MAPE-K loop typically actuates over a system composed of traditional components, i.e., non-ML components. However, as illustrated in Figure 2.1, ML-enabled systems generally encompass both non-ML and ML components. We argue that the MAPE-K loop should be revised in order to cope with the unique issues described in Section 2.2.1 that affect MLCs by effectively leveraging the adaptation tactics presented previously. In the following, we discuss the research challenges and opportunities that arise in each of the MAPE-K loop stages when engineering self-adaptive ML-based systems.

Monitor

The *Monitor* stage has to keep track of the inputs received by the ML components because shifts of the input distributions may affect the predictions. For instance, the detection of out-of-distribution inputs may mean that there has been a change in the environment and thus the model used by some ML component may no longer be representative of the current environment. The challenge here is not only detecting the occurrence of shifts in a timely and reliable fashion, but also how to effectively characterize them — since different types of shifts require different reaction methods.

As in other SAS, attributes that contribute to the system's utility (e.g., latency, throughput) or the satisfaction of required system properties must be monitored. In addition to these, the *Monitor* stage must also gather the outputs of the ML component to account for situations in which changes in the inputs go by unnoticed, perhaps because they are too slow, but that manifest themselves faster in the outputs [198]. Examples of outputs that should be monitored are, for instance, shifts in the output distribution, model's predictive performance (e.g., accuracy) and error — obtained by comparing predictions with real outcomes. A relevant challenge here is that often real outcomes are only known after a long time, if ever. For instance, in fraud detection, false negatives (i.e., undetected real fraud) are known only when users file complaints. Approaches such as

those proposed in [142, 196, 198] provide a good starting point for the implementation of a *Monitor* for self-adaptive machine-learning enabled systems.

Challenges. Monitoring input and output distributions requires keeping track of a multitude of features and parameters, which would otherwise be disregarded. This is already challenging due to the amount of data that needs to be stored, maintained, and analyzed. Finding suitable frequencies to gather these data and adapting them in the face of evolving time constraints is an even bigger challenge in time-critical domains [11, 142].

Analyze

The *Analyze* stage is responsible for determining whether degradations of the prediction quality of ML components are affecting (or are predicted to affect) other system components and system utility to such an extent that adaptation may be required. To accomplish this, one can leverage techniques developed by the ML community to detect possible issues in the inputs and outputs of the ML component [142, 149, 150, 198], errors in its training set [2] and the appearance of new features relevant for prediction [139]. These techniques must then be adjusted for each system, which includes adapting them to different ML models and tasks.

Challenges. Estimating the impact of an MLC on other system components and on system utility can be challenging because often (mis)predictions affect the system's utility/dependability in ways that are not only application- but also context- dependent. For instance, during periods with higher transaction volumes, such as on Black Friday, mispredictions have higher impact on system utility, since during these periods it is more critical to accurately detect fraud while maximizing accepted transactions. Architectural models can capture the information flows among components, but the challenge is to estimate how the uncertainty in the output of the ML components propagates throughout the system.

Plan

The *Plan* stage is responsible for identifying which adaptation tactics (if any) to employ to address issues with ML components that are affecting overall system utility. As with other self-adaptation approaches, this reasoning should consider the costs and benefits of each viable tactic. Further, most of the ML adaptation tactics described have a non-negligible latency, which needs to be accounted for as in latency-aware approaches [128]. An additional concern is that some of these tactics may require a considerable use of resources to execute, either in the system itself or offloaded. This requires *Plan* to account for this impact or cost. For MLSs that rely on multiple MLCs, whenever a system property is (expected to be) violated or when system utility decreases, fault localization may be required to understand which component is under-performing and should be adapted [45].

Challenges. Although there are several approaches [6, 31, 194] that attempt to predict the time/cost of training ML models, this is a complex problem that is strongly influenced by the type of ML model considered, its hyper-parameters and the underlying (cloud) infrastructure used for training. These techniques represent a natural starting point to estimate the costs and benefits of adaptation tactics such as the ones presented.

One interesting direction is to exploit techniques for estimating the uncertainty [135] of ML models to quantify both the likelihood of models' mispredictions as well as the potential benefits deriving from employing corrective adaptation tactics. While some ML models can directly estimate their own uncertainty [133], others require additional techniques (e.g., ensembles [22]) to obtain uncertainty estimations. Still, existing techniques can suffer from significant shortcomings in practical settings [135].

Finally, tactics that modify MLCs are typically computationally expensive (e.g., non-negligible latency). Thus, *Plan* must have mechanisms to ensure that the tactic is executed without compromising other components/properties, or even the entire system.

Execute

To execute a given adaptation tactic, the *Execute* stage must have access to mechanisms to improve or replace the ML component and/or its training set. As in the conventional MAPE-K loop, we require implementations of adaptation tactics that: are efficient to execute, have predictable costs/benefits, and are resilient to run-time exceptions.

Challenges. A key challenge is how to enhance the predictability of the execution of ML adaptation tactics, which often require the processing of large volumes of data (e.g., to retrain a large scale model) possibly under stringent timing constraints. We argue that the community of SAS would benefit from the availability of open-source software frameworks that implement a range of generic adaptation tactics for MLCs. These frameworks would allow to mask complexity, promote interoperability and comparability of SAS. Further, this would also provide an opportunity to assemble, in a common framework, techniques that have been proposed over many years in different areas of the AI/ML literature.

Knowledge

Finally, the *Knowledge* module is responsible for maintaining information that reflects what is known about the environment and the system. As in traditional systems, in the case of self-adaptive ML-enabled systems *Knowledge* also needs to maintain information about the environment so that trends can be observed. These trends can be crucial to detect the shifts that may lead to mispredictions.

Additionally, for MLSs, the *Knowledge* component should evolve in order to keep track of the costs/benefits of each ML adaptation tactic on the affected MLCs and on system utility. This corresponds to gathering: (i) knowledge on how each tactic altered an MLC and on the context in which the tactic was executed; and (ii) meta information on training sets, for instance characterizing the most important features for predicting the costs and benefits of the different tactics. This added knowledge should be leveraged to improve

the decision making process and thus improve adaptation. By gathering knowledge on how each tactic altered an MLC and on the context in which the tactic was executed, the *Analyze* and *Plan* stages can take more effective decisions on *when* and *how* to adapt.

Finally, for a tactic that replaces under-performing MLCs, *Knowledge* must contain a repository of the available components and their meta-data. This meta-data, we argue, should provide information to enable reasoning on whether the necessary preconditions to enable a safe and timely adaptation hold.

4.4 Summary

This chapter introduced a repertoire of adaptation tactics, inspired by the ML literature, for adapting ML components, and discussed their advantages and disadvantages. Then, a fraud detection system case study was used to demonstrate the application of each tactic. Finally, we identified a set of key requirements that should be supported by the various elements of the classic MAPE-K control loop and a set of challenging research problems, out of which we highlight the following: (i) How to estimate the costs and benefits of each tactic? (ii) How to reason about the impact of ML mispredictions on system utility? (iii) How to reason about the long-term impacts of adaptation tactics on system utility? The next chapter addresses these challenges by introducing a framework for engineering self-adaptive ML-based systems.

Chapter 5

A Framework for Self-Adapting Machine Learning-based Systems

This chapter presents a framework for engineering self-adaptive ML-based systems. There are two key requirements associated with reasoning about self-adaptation of ML components. First, it is necessary to understand if and how ML predictions affect overall system utility. Second, it is necessary to estimate the costs and benefits of the available adaptation tactics. Let us now discuss the key challenges associated with each requirement.

I) Impact of machine learning predictions on system utility. A key problem that needs to be addressed to enable automatic reasoning on the dynamics of ML-based systems is determining to what extent incorrect predictions will impact overall system utility. In fact, this is not only application but also context dependent. For example, in cloud configuration recommenders, when the relative difference in job execution latency between the available cloud configurations is low, ML mispredictions have little impact on system utility [31]. Similarly, in a fraud detection system, the impact of mispredictions is different in periods with higher volumes of transactions, in which it is critical to maximize accepted transactions, while accurately detecting fraud [8].

II) Estimating costs and benefits of adaptation tactics. Predicting the cost and benefits of ML adaptation tactics is far from trivial. This prediction is strongly influenced both by the type of models and their settings (hyper-parameters and execution infrastructure), and by the data employed in the adaptation process. For instance, in the case of a tactic that triggers the retrain of an ML model, the benefits of tactic execution depend on the data available for the retrain – data more representative of the current environment contributes to higher benefits. Differently, if the adaptation tactic consists of querying a human (*human-in-the-loop* tactic), the benefits are now dependent on human expertise. The execution latency and economic cost are also likely to be different and affected by factors that are inherently tactic dependent, e.g., the retraining time is affected by the amount of available training data, whereas the latency of a *human-in-the-loop* tactic may depend on the complexity of the problem the human is required to solve.

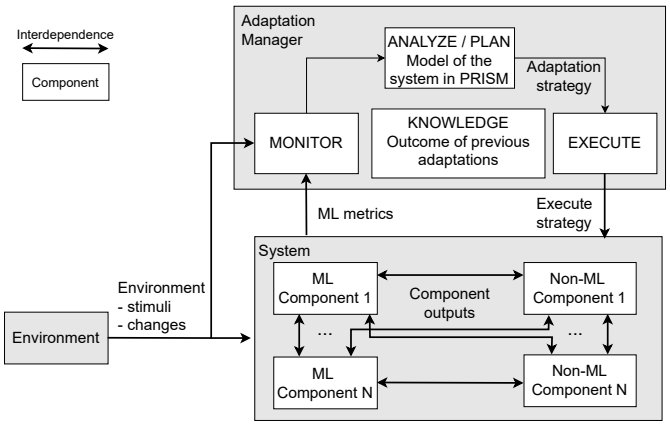


Figure 5.1: Framework modules and inter-dependencies.

This chapter describes a generic framework that can be used to derive formal models of self-adaptive ML-based systems. The resulting models can then be utilized to enable automatic reasoning via probabilistic model checking tools such as PRISM [99]. In fact, the use of model checking tools in the self-adaptive systems (SASs) domain is not new [28, 124, 127]. Conceptually, the proposed framework can be regarded as a specialization of the frameworks already proposed in this field, but instead targeting a specific class of "managed" systems: ML-enabled systems. Further, the proposed framework allows us to abstract away from system-specific issues and instead instantiate the decision of whether to adapt ML components as a general decision that relies on the key factors of ML-based systems. The proposed framework should abide by the following key requirements:

R1 Provide the means to predict the effects of adapting and not adapting the model on its future predictive performance;

R2 Include a way to characterize in a compact but meaningful way the error of an ML component;

R3 Allow the self-adaptation manager to determine the impact of ML mispredictions on overall system utility.

The following sections focus on how to capture the most relevant dynamics of ML components via abstract and generic models that can be extended and customized to specific use cases and domains.

5.1 Architectural Overview

As in typical frameworks for SAS, our framework requires specifying the behavior of the modules displayed in Figure 5.1: environment, system, and the adaptation manager. Next, we discuss how each of these modules is modelled in the proposed framework. For the ML component, we further describe its internal state, how it evolves, and the methods exposed by its interface to allow for inter-component interactions.

Environment. The environment module accounts for the types of *stimuli* to which the system responds/reacts. Additionally, since our goal is to reason about the impact of environment changes, these must also be modeled. Examples of environment *stimuli* are for instance the transactions that a fraud detection system has to classify, the sentences that a machine translation system has to translate, or the jobs received by a scheduler.

Adaptation manager. As in typical SAS, the adaptation manager contains a repository of adaptation tactics. However, and differently from previous work in this domain, we consider adaptation tactics that directly actuate over the ML component [32], such as retraining the ML model. Each adaptation tactic is specified by: (i) a pre-condition that triggers its execution and which generally depends on the state of the system and the environment; (ii) the effects on the targeted components. We model adaptation tactics as a tuple composed of tactic cost and tactic latency. This division allows us to study the impact of the different dimensions of tactics on overall system utility. For example, consider a retrain tactic: it has an associated latency, since retraining a model takes a non-negligible amount of time; if that tactic is executed in cloud environments it also has a monetary cost, which depends both on its latency, and on the underlying cloud platform selected for the execution. By leveraging model checking tools such as PRISM [99] we can explore alternative adaptation policies with the objective of identifying the one that optimizes system utility.

ML-enabled system. The key novelty of our framework is that it enables reasoning about the adaptation of ML-enabled systems, which we abstractly define as systems that comprise two types of components: ML-based and non-ML based components (Figure 5.1). An ML-based component is used to encapsulate an ML model that can be queried and updated (e.g., retrained). Non-ML based components are used to encapsulate the remaining functional components of the system being managed/adapted. Our framework is agnostic to the modeling of the application-dependent dynamics of non-ML based components, which can be achieved by resorting to conventional techniques already proposed in the SAS literature [28, 124, 127]. However, the interactions between non-ML components and ML components may still need to be modeled for two purposes: i) pre-processing data to act as input to the ML component or ii) using the ML component's outputs to perform some function affecting system utility by adding negative/positive rewards upon completion of a task. For example, in an ML-based scheduling system, once a job finishes executing on the selected cloud platform (considered to be a non-ML based component), it triggers the accrual of a reward (e.g., dependent on the job's execution time) that affects system utility.

As for modeling ML components, our design aims to ensure the following key properties: **(i) generic** – applicable to offline and online learning, supervised, unsupervised and semi-supervised models, different types of ML models (e.g., neural networks, random forests); **(ii) tractable** – usable by a probabilistic model checker like PRISM, having a high level of abstraction to aid systematic analysis via model checking; **(iii) expressive** – capable of capturing key dynamics of ML models that are general across ML models; **(iv) extensible** – designed to be easily extended to incorporate additional adaptation tactics

and customized to capture application specific dynamics. The following section details how we formally model ML components to capture their error in a compact but meaningful way (realizing R2), and the impact of mispredictions on system utility (realizing R3).

5.2 Formally Modeling ML Components

We start by abstractly formalizing the behavior of an ML component by specifying (i) its state; (ii) the set of events that change its state; (iii) the logic governing how the internal state evolves due to each possible event.

Machine learning component state. The state of an ML component is characterized by two elements: a *quality matrix* and the set of new data (knowledge) which encodes information regarding the data accumulated so far by the ML component. As the name suggest, a *quality matrix* is a $n \times d$ matrix where n represents the input types for the system, and where d represents the different metrics of interest for a system (e.g., mean absolute error, accuracy). For example, in the case of a machine translation system, it may be relevant to quantify translation quality in terms of news domain, such as sports or finance. Similarly, for a fraud detection system, the input types can represent the types of transactions (e.g., legitimate, fraudulent).

For the specific case of classification ML models, the *quality matrix* can be seen as a confusion matrix. The confusion matrix is a tabular way to represent the quality of a classifier [175] and allows us to abstract over the internal dynamics of the specific model being used while still capturing the quality of its predictions. More in detail, it is a $n \times n$ matrix where n represents the number of output classes. In cell (i, j) the confusion matrix maintains the probability of an input of class i to be classified by the model as belonging to class j. In fact, due to how it is constructed, the confusion matrix provides access to metrics such as false positives/negatives, which constitute the basis for computing alternative metrics, like f-score or recall, that can be of interest for domains such as fraud detection systems [8].

The framework also supports the specification of multiple confusion matrices, which can be of interest when there are several input types to an ML model: for instance, if a scheduler receives different job types, some easier to classify than others, this could be modeled by associating a different quality matrix to each job type. In such a scenario, if the metrics of interest are by-products of the confusion matrix (e.g., accuracy and recall), a single quality matrix can be employed instead of multiple confusion matrices. The quality matrix would thus have each job type for dimension n and each metric for dimension d.

The second element of the ML component's state represents the knowledge maintained by the ML component. This is characterized by (i) the data that the model has already used for training purposes, and (ii) the new data that is continually gathered during operation and that represents the current state of the environment the system is operating in. However, the representation of this knowledge is application dependent. While in simpler use-cases it may be enough to maintain a counter for the inputs that arrive in the system, in more complex scenarios the data gathered during execution may encode important

information to characterize the environment (e.g., measures of dataset shift [149]). This knowledge can be used by the adaptation tactics when adapting the ML component.

Machine learning component interface. For the other components in the system to interact with the ML component, we propose a general interface that enables this interaction. Typically, any ML component that supports adaptation requires three key methods: QUERY, UPDATE_KNOWLEDGE, and RETRAIN. Clearly, new methods can be added to this interface to tailor the framework to specific application requirements, such as to capture manipulations of the dataset (e.g., sub-sampling certain types of inputs) or to support additional adaptation tactics (e.g., unlearning). We detail each key method below.

As the name suggests, QUERY is used to solicit a prediction from the ML component. Since the quality matrix abstracts away the behavior of the ML component, in the general case executing a QUERY corresponds to reading the value for the desired metric from the quality matrix. When the quality matrix takes the form of a confusion matrix, the internal behavior of the ML component when issuing predictions is abstracted away by reasoning only over the likelihoods specified in the confusion matrix. Thus, the output event produced by executing a QUERY is a probabilistic event that can assume any of the possible output classes of the classifier, with the probability given by the classifier's confusion matrix.

The method UPDATE_KNOWLEDGE should be called when the system has reacted to an event and thus there is new data to be accounted for. The framework can keep track either of the pair ⟨*ML input, ML prediction*⟩ or of the triple ⟨*ML input, ML prediction, real output*⟩. The selection of either option is domain dependent. For example, in the fraud detection domain, knowing the actual value of a transaction (legitimate or fraudulent) is not always possible [142]. Similarly, for machine translation systems, the availability of correct/perfect translations for a sentence is not guaranteed. In such cases, the pair ⟨*ML input, ML prediction*⟩ must be employed.

Finally, RETRAIN corresponds to retraining the ML model resorting to the data stored in the ML component's state. As discussed in Section 4.1, different types of RETRAIN, with varying complexities, can be executed, for instance by modifying the hyper-parameters and/or the architecture of the ML model. Ultimately, the type of RETRAIN executed is application dependent.

The framework can be extended to account for more methods, and in particular to more adaptation tactics. Since tactics typically have an associated cost and latency that directly impact system utility, these are captured by the framework as follows: the latency is used to determine after how many units of time the effects of a tactic are applied to the ML component (and to the system); the cost of executing a tactic (when it has a cost) is discounted from the system's utility.

Machine Learning component state evolution. When an interface method is triggered, the state of the ML component can be altered. Since QUERY consists only of asking the ML model for a prediction, it does not alter the component's state. UPDATE_KNOWLEDGE changes the knowledge of the ML component by adding instances to that set. Finally, RETRAIN changes both elements of the state: the quality matrix and the knowledge are

updated to reflect the execution of the adaptation tactic. In the case of a retrain adaptation tactic, the data used for executing the tactic is updated in the knowledge of the ML component such that it is no longer considered new data. The quality matrix is updated to reflect the new values for the system metrics or, in the specific case of a confusion matrix, the new predictive performance of the ML model after the retrain.

5.3 Predicting the Effects of Adapting (or not)

A key requirement of our framework is the ability to predict the costs and benefits of executing adaptation tactics on the ML components (requirement R1). For this purpose, the framework associates an *adaptation impact predictor (AIP)* with each adaptation tactic. The AIP is in charge of predicting: (i) the adaptation tactic's *cost*, that is charged to the system utility; and (ii) the impact of the adaptation on the future *quality* of the ML component. For each ML component, we also include an adaptation tactic corresponding to performing no changes to the ML component (*NOP*). While the AIP for tactic *NOP* always predicts zero costs (this tactic inherently has no cost), its model quality predictor captures the evolution of the model's performance if no action is taken, e.g., the possible degradation of accuracy of the ML component due to data shifts. Overall, this approach allows the model checker to quantify the impact of different adaptation tactics on system utility and reason about their cost/benefits trade-offs.

We focus on the problem of how to estimate the future evolution of the predictive performance of the ML component and describe, in the following paragraphs, how we tackle the problem of implementing AIPs for the *Retrain* and *Nop* tactics for generic ML components. Indeed, for adaptation tactics such as *Retrain* the problem of estimating its cost has been investigated in the system's community. The literature has shown that data-driven approaches based on observing previous retraining procedures [31], possibly mixed with white-box methods [194], can generate accurate predictive models of the retrain cost.

Predicting the future quality of ML components. Given the reliance on a *quality matrix* $\mathcal{C}$ to characterize the performance of ML components, estimating how the predictive performance of these components evolves requires estimating how $\mathcal{C}$ will evolve in the future, for instance due to shifts affecting the quality of the current model (tactic *nop*) or as a consequence of retraining the model to incorporate newly available data.

The proposed method abstracts over the specific adaptation tactic $a()$ by modeling it as a generic function $a(\mathcal{M}, \mathcal{I}, \mathcal{N}) \longrightarrow \mathcal{M}'$ that produces a new ML model $\mathcal{M}'$, and takes as input: (i) model $\mathcal{M}$ prior to the execution of the tactic; (ii) data $\mathcal{I}$, used to generate model $\mathcal{M}$; (iii) new data, $\mathcal{N}$, that became available since the last adaptation, e.g., by deploying the model in production and gathering new samples and corresponding ground truth labels. We assume that both $\mathcal{I}$ and $\mathcal{N}$ contain ground truth labels. Additionally, we assume that $\mathcal{M}$ and $\mathcal{M}'$ are generic supervised ML models that are queried and return predictions for the input samples. These two assumptions allow us to determine the quality matrices of models $\mathcal{M}$ and $\mathcal{M}'$ at any future time interval, since their predictions can be compared with the ground truth labels.

We seek to build black-box regressors (e.g., random forests, neural networks) that, given model $\mathcal{M}$ obtained at time 0 on dataset $\mathcal{I}$, and new data $\mathcal{N}$ available at time $t > 0$, predict the quality matrices of both models ($\mathcal{M}$ and $\mathcal{M}'$) at time $t + k$, where $k > 0$ is the prediction look-ahead window.

Adaptation impact dataset (AID). To train such black-box regressors, we build an AID by systematically simulating the execution of the adaptation tactic using production data at different points in time. This allows us to gather observations characterizing the execution of the adaption tactic in different environmental contexts, such as: (i) different sets of data used to adapt the model; (ii) variations in the time passed since the last execution of the tactic; (iii) different ML performance before and after the adaptation

The first step of the procedure to build an AID consists of monitoring model $\mathcal{M}_0$ of a ML component in production over T time intervals. During this monitoring period, given the absence of AIPs, we assume that no adaptation is executed.

Next, we deploy $\mathcal{M}_0$ on a testing platform (so as not to affect the production environment) and systematically apply tactic $a()$ at each time interval $i > 0$, i.e., $a(\mathcal{M}_0, \mathcal{I}_0, \mathcal{N}_i)$. This yields a new model $\mathcal{M}_i$, which we evaluate at every future time interval $i < j \leq T$, obtaining the corresponding quality matrices, noted as $\mathcal{C}_i(j)$. Overall, this procedures yields T models, resulting from the adaptation of $\mathcal{M}_0$ at different time intervals, and produces $T \cdot (T-1)$ measurements of the quality matrices at times $j > i$.

For each of the $T \cdot (T-1)$ measurements, we generate an AID entry, $e_{i,j,k}$ that describes the quality of model $\mathcal{M}_j$ at time $j + k$ obtained by executing $a(\mathcal{M}_i, \mathcal{I}_i, \mathcal{N}_j)$ at time j on model $\mathcal{M}_i$, where $\mathcal{I}_i$ denotes the data used at time i to generate model $\mathcal{M}_i$, and $\mathcal{N}_j$ the new data gathered from time i until time j. Each entry $e_{i,j,k}$ has as target variables the metrics of the quality matrix or, for the particular case of classification models, the values for the confusion matrix cells, at time $j + k$ of model $\mathcal{M}_j$ and stores the following features:

- *Basic features:* provide basic information on (i) the amount of data (i.e., number of examples) used to generate model $\mathcal{M}_i$, i.e., $\mathcal{I}_i$, and gathered thereafter, i.e., $\mathcal{N}_j$; (ii) the accuracy of the model shortly after its generation and at the present time; (iii) the time elapsed since the last execution of the adaptation tactic, i.e., $j - i$.
- *Output characteristics features:* describe the distribution of the output of models $\mathcal{M}_i$ and $\mathcal{M}_j$. It also includes the distribution of the uncertainty of the models' predictions. This feature is included only when the ML model provides information regarding the uncertainty of a prediction. This information is usually provided by commonly employed ML models like random forests, and Gaussian processes.
- *Input characteristics features:* aim to capture variations in the distributions of the features of datasets $\mathcal{I}_i$ and $\mathcal{N}_j$. Specifically, for each feature f, we compute the Pearson correlation coefficient between its values in $\mathcal{I}_i$ and $\mathcal{N}_j$.

Overall, the AID can be seen as composed of pairs of features, where each pair describes a specific "characteristic" of the data or model at two different points in time, e.g., amount of data available at time i and j, or distribution of predicted classes at time $j + k$ by models $\mathcal{M}_i$ and $\mathcal{M}_j$. The last step of the process consists of extending the AID by encoding the variation of each feature as follows: (i) for scalar features (e.g., amount of data)

we encode their variation using the ratio and difference; (ii) for features described via probability distributions (e.g., prediction's uncertainty) we quantify their variation using the Jensen-Shannon divergence [120] (inspired by previous work [142]), which yields a scalar measurement of the similarity between two probability distributions. This generic methodology can also be applied to the case of the *nop* tactic. In this case, the dataset describes how the accuracy of a model originally obtained at time i will evolve at time $j+k$, based on the information available at time j. This additional information can simply be added to each entry $e_{i,j,k}$.

Adaptation impact predictors (AIPs). We exploit the AID to train a set of independent AIPs, which can be simple linear models or black-box predictors such as random forests or neural networks. Each AIP is trained to predict the value of a different metric of the quality matrix, or the value of a cell of the confusion matrix. Specifically, given an n-ary classification problem, we have $n^2 - n$ independent values for the corresponding confusion matrix, given that each row must sum to 1. For the case of binary classification, $n = 2$, it is sufficient to predict the values of the two elements on the diagonal, which, being in different rows, are not subject to any mutual constraint. For the general case of $n > 2$, it is necessary to ensure that the predictions of the AIPs targeting different cells of the same row sum to 1. This can be achieved by using a soft-max function [17] to normalize the predictions generated by the AIPs into a probability distribution.

Integrating the AIPs in the formal model. If the AIPs are complex models, possibly black-box (e.g., neural networks) they can not be efficiently (nor easily) implemented as part of the formal model, which is verified by a probabilistic model checker such as PRISM [99]: these tools do not allow the verification process to interact with external processes (which could be used to encapsulate the implementation of the AIPs) during model analysis. However, if the framework is being used to plan for the following time interval, the AIPs can be queried before the formal verification takes place and their predictions can be sent as input variables to the formal model. Section 5.5 provides a detailed discussion on how to use the framework to plan for the long-term.

5.4 Accounting for Label Delay

While it is fairly trivial to compute values for features such as (i) the amount of new samples with which the ML model has not been trained and (ii) the time elapsed since the model was last retrained, computing the ML model's real time predictive performance may not be as straightforward. As discussed in Section 2.3.2, this is the case in contexts for which ground-truth labels may take a non-negligible time to become available.

To address this challenge, we leverage a state-of-the-art ML algorithm [67] to deal with ground-truth label unavailability when adapting ML-enabled systems. This way the proposed framework can be used for a wider range of domains without requiring assumptions on label availability, increasing the framework's applicability and generalizability.

The following paragraphs provide an overview of the state-of-the-art approach employed for performance estimation [67], along with a discussion of its limitations and a proposal for a new version that is better suited to the framework and, in particular, to ML classifiers. Note that regression problems (the ML model's output domain is continuous) can be turned into classification ones by discretizing the target domain, although at the cost of an intrinsic prediction error due to the chosen discretization granularity.

Average thresholded confidence (ATC)

For self-containment, we provide an overview of ATC [67] and start by introducing preliminary terminology. Let f be a k-class calibrated classifier and $f_k(x)$, $\forall k \in \mathcal{Y}$, the predicted probability of an input x belonging to one class k of the set of output classes $\mathcal{Y}$, according to classifier f. ATC requires a score function, $s : [0,1]^k \rightarrow \mathbb{R}$, which takes as input the k-dimensional vector of probabilities output by classifier f, and outputs a real number. The score function s captures the confidence of classifier f in its prediction and is used to estimate the expected mis-classification rate. As such, the score function s is chosen such that if the classifier predicts that the output class for an input x is $i \in \mathcal{Y}$ with high probability relatively to the other classes, then $s(f_i(x))$ should be high: $f_i(x) >> f_j(x), \forall j \neq i \implies s(f_i(x)) > s(f_j(x))$. Conversely, if the classifier predicts class i for input x with relatively low probability, then $s(f_i(x))$ should be low: $f_i(x) \leq f_j(x), \forall j \neq i \implies s(f_i(x)) < s(f_j(x))$.

Originally, ATC [67] considers two score functions: Maximum confidence – $s(f(x)) = \max_{j \in \mathcal{Y}} f_j(x)$; and Negative Entropy – $s(f(x)) = \sum_j f_j(x) \log\left(f_j(x)\right)$. However, in this work we use only the Negative Entropy function.

In a nutshell, ATC estimates an ML model's predictive quality as follows: given a validation set $\mathcal{D}^{val}$ of labeled data (e.g., which includes the most recent ground truth labels for the past predictions of classifier f) ATC identifies a threshold t on $\mathcal{D}^{val}$ such that the number of samples that obtain a score less than t match the number of errors of classifier f on $\mathcal{D}^{val}$. This procedure is illustrated by the pseudo-code in Algorithm 1 and can be expressed compactly as follows:

$$\mathbb{E}_{x \sim \mathcal{D}^{val}}\left[\mathbb{I}\left[s(f(x)) < t\right]\right] = \mathbb{E}_{(x,y) \sim \mathcal{D}^{val}}\left[\mathbb{I}\left[\text{argmax}_{j \in \mathcal{Y}} f_j(x) \neq y\right]\right], \tag{5.1}$$

where $\mathbb{I}$ denotes the indicator function, and y the ground truth class for input x. The left side of the equation defines the ratio of samples in $\mathcal{D}^{val}$ with score below the threshold t and the right side specifies the error rate for $\mathcal{D}^{val}$ (i.e., number of errors of the classifier f on $\mathcal{D}^{val}$), which we also note as $Err(\mathcal{D}^{val})$. Threshold t can be easily computed as it corresponds to the $Err(\mathcal{D}^{val})$-percentile of the distribution of scores for $\mathcal{D}^{val}$. One can then estimate the correctness of individual predictions on a target, unlabeled dataset $\mathcal{D}^{\mathrm{T}}$ based on whether each prediction's score is above/below t. The error rate for $\mathcal{D}^{\mathrm{T}}$ can thus be computed as:

$$Err(\mathcal{D}^{\mathrm{T}}(s)) = \mathbb{E}_{x \sim \mathcal{D}^{\mathrm{T}}}[\mathbb{I}[s(f(x)) < t]]. \tag{5.2}$$

Note that ATC can estimate whether individual predictions are correct or not. In the general multi-class scenario ($k > 2$), this does not allow us to pinpoint which class is

Algorithm 1 Logic employed by ATC to determine the decision threshold t

procedure FIND_THRESHOLD(D^{val})
    $Err(\mathcal{D}^{val}) \leftarrow \frac{\|misclassified\ inputs\|}{\|inputs\|}$ ▷ Compute the error rate on the validation set
    return Percentile$^{Err(\mathcal{D}^{val})}$(s(f(x))) ▷ Return the $Err(\mathcal{D}^{val})$-th percentile of the score distribution for the validation set
end procedure

expected to be the correct one, in case a classifier's prediction is deemed incorrect. However, for binary classification problems estimating the correctness of a prediction implies also determining which is the expected class in case the classifier's prediction is estimated to be incorrect. This allows us to employ ATC to estimate the whole confusion matrix.

Class-Based-ATC (CB-ATC)

In general, label delay can be thought of as a form of temporal misalignment between the data and the labels. More formally, for a delay d and current timestamp t, ground truth labels are available for environmental events (e.g., transactions of a fraud detection system) that completed up to time instant $t - d$. Thus, to estimate the current quality of an ML model, one either resorts to (i) delayed labels, hence using stale data as a proxy for the ML model's quality, or (ii) to methods like ATC that aim to estimate the ML model's predictive quality in the absence of labelled data.

While integrating ATC within our framework we identified two relevant shortcomings, which led us to propose a new method: class-based-average thresholded confidence (CB-ATC). The next paragraphs describe ATC's limitations, which are illustrated in Figures 5.2 and 5.3, and explain how CB-ATC circumvents them.

Limitation 1. ATC's first limitation is related to its (implicit) assumption on the distributions of scores for the correct/incorrect predictions of each class being "similar enough" so that by using a single threshold, it is possible to fit the error rate on validation data accurately for both predicted classes. Yet, if the scores of incorrect/correct predictions are distributed in different regions for different predicted class, as illustrated in Figure 5.2 (middle), ATC is unable to correctly fit the confusion matrix using a single threshold. CB-ATC addresses this limitation by computing a threshold per predicted class (Figure 5.2, right), which is set to match the error rate for the prediction of that specific class. More formally, in CB-ATC each class threshold, denoted as t_i, where $i \in \mathcal{Y}$ and $\mathcal{Y}$ is the set of output classes, is computed as follows:

$$\mathbb{E}_{x \sim \mathcal{D}^{val}} \left[\mathbb{I} \left[pred(f(x)) = i \wedge s(f(x)) < t_i \right] \right] = \mathbb{E}_{(x,y) \sim \mathcal{D}^{val}} \left[\mathbb{I} \left[pred(f(x)) = i \wedge y \neq i \right] \right], \quad (5.3)$$

where $pred(f(x))$ denotes the class predicted by classifier f for input x. The use of a per class threshold provides CB-ATC with additional flexibility with respect to ATC and, as illustrated in Figure 5.2, allows CB-ATC to fit precisely the classifier's confusion matrix.

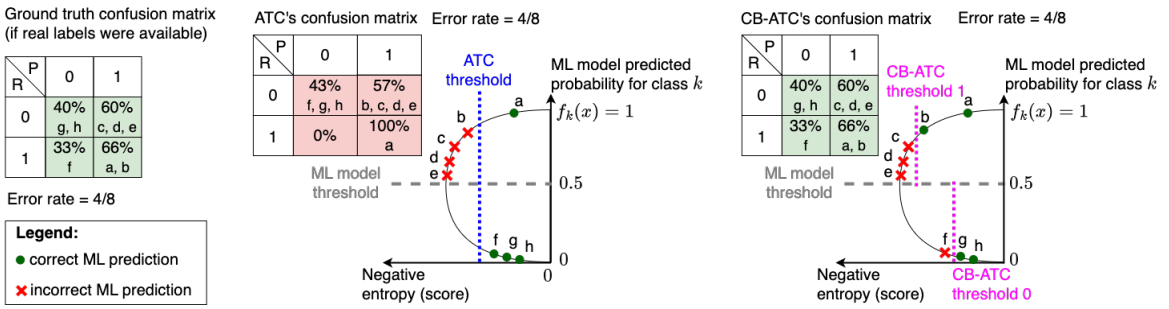


Figure 5.2: ATC (middle plot) is unable to correctly fit the actual confusion matrix (on the left, where 'R' stands for real/actual ground truth; and 'P' for predicted labels) of the validation data considered in this example via a single threshold. This is due to the different characteristics of the distributions of scores for the samples that are predicted as class 0 and 1 (i.e., below and above the assumed 0.5 model threshold). By using one threshold per class, CB-ATC (right plot) accounts for the different distributions of scores in each class and fits the actual confusion matrix.

Limitation 2. The second limitation that is inherent to ATC's design is related to the fact that in many real world applications (including ML-based financial fraud detection systems), the predicted class does not necessarily coincide with the one having higher probability according to the classifier. Focusing on the binary classification case, this means that the threshold used to decide the class to which a prediction belongs (based on the model's output probabilities and referred to as "ML model threshold" in Figures 5.2 and 5.3) is often not 0.5. Figure 5.3 illustrates this scenario, and considers a case in which the ML model's threshold is set, by design, above 0.5.

This example demonstrates that ATC fails to correctly fit the confusion matrix and we argue that this is due to two main causes: (i) the use of a single threshold to fit the global error rate rather than a per class error rate (as discussed by Limitation 1); (ii) the use of a scoring function (like negative entropy), which is symmetric around 0.5, fails to capture the following key desirable property whenever the model's threshold is not 0.5: when the model predicted probability gets closer to the model's threshold (from any given direction, i.e., above or below in Figures 5.2 and 5.3), the scoring function should also decrease (uncertainty grows as we approach the threshold). To tackle this issue, CB-ATC uses a modified version of the Negative Entropy function, which we call Modified Negative Entropy (MNE) and is defined as follows:

$$MNE(f(x), pred(f(x))) = \begin{cases} NE(f(x)), & \text{if } pred(f(x)) = \text{argmax}_{j \in \mathcal{Y}}(f_j(x)) \\ -NE(f(x)) - 2, & \text{otherwise} \end{cases} \quad (5.4)$$

where $NE()$ stands for the Negative Entropy function. Analyzing Figure 5.3, it is possible to see that $MNE()$ ensures, by design, that the two misclassified samples for which class 0 was predicted get a lower score than any other predicted class-0 sample in the validation set (samples e and f are the closest to the ML model's threshold and correspond to more uncertain predictions). We evaluate CB-ATC in Section 6.2.2 with the fraud detection

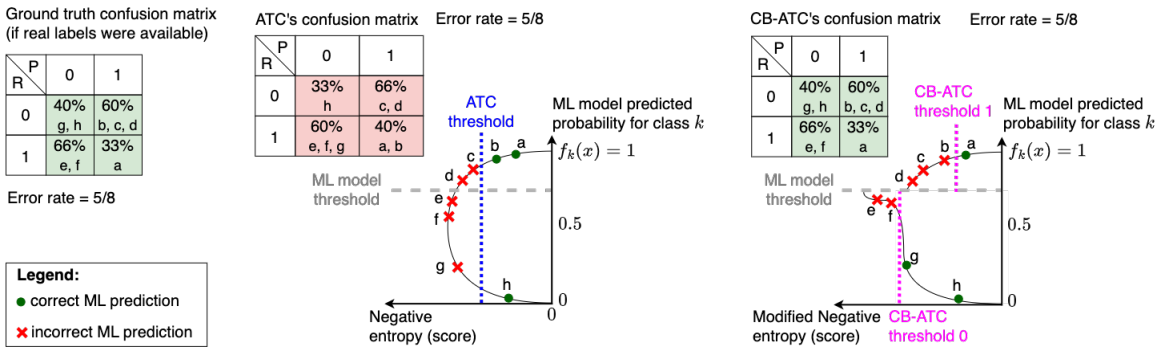


Figure 5.3: ATC (middle) fails to correctly fit the actual confusion matrix (left) in a scenario in which the model threshold is set to a value different from 0.5 (to control the trade-off between recall and precision). In the same scenario, CB-ATC (right) can correctly fit the actual confusion matrix by: (i) fitting the error rate of each class via a different threshold; (ii) employing a Modified Negative Entropy score function, which ensures that $P(f(x) = 1)$ monotonically decreases as it approaches the ML model's threshold (i.e., as a prediction's uncertainty increases).

use-case and demonstrate how it allows the framework to improve overall system utility compared to a version that uses ATC and to a version that relies on delayed labels.

5.5 Long-term Estimation of the Benefits of Model Retrain

Up until this point, we presented a short-sighted version of the adaptation framework that does not take full advantage of the probabilistic model checker to plan for the long term. In this section we introduce RIPPLE, (**R**etrain **I**m**P**act **P**redictor for **L**ong-t**E**rm Planning), a self-adaptation approach to decide for the long term when to retrain ML-enabled systems. Specifically, RIPPLE focuses on the model *Retrain* tactic and is later evaluated based on the fraud detection use-case (Section 6.2).

Challenges of using the framework for long-term planning

The methodology described in Section 5.3 does not support long-term planning and extending it to plan for the long term is not trivial for the following reasons.

I. AIPs cannot be integrated in model checkers such as PRISM [99]. Because the AIPs are complex models, possibly black-box (e.g., random forest or neural network models) they can not be efficiently (nor easily) implemented as part of the formal model. Therefore, as described in Section 5.3 the AIPs are queried prior to extracting the expected optimal adaptation strategy with the probabilistic model checker. The predictions of the AIPs, which encode the expected benefits of executing the *retrain* and *nop* adaptation tactics, are then sent to the model checker as input features for the formal verification

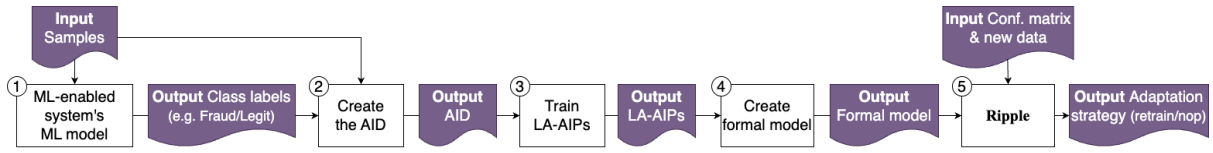


Figure 5.4: Overview of the process to instantiate long-sighted self-adaptive ML-enabled systems via Ripple.

process to start. This approach is acceptable for short-term planning, as one has to pre-query each AIP only once to get the prediction of the effects of each adaptation tactic solely for the subsequent time interval.

However, for long-term planning scenarios of look-ahead $L > 1$ it is not feasible to pre-query the AIPs to get predictions for the L future states of the ML model (and for all the tactics available for execution – e.g. *retrain* and *nop*), and then send these predictions as input to the model checker. Getting predictions for the L future states of the ML model requires simulating the evolution of the system and of the environment outside the model checker. Yet, this is exactly what model checkers do during the verification process. As such, repeating the simulation outside the model checker would render the whole process inefficient and error-prone.

II. Model checkers lack support for pausing and resuming a formal verification process. Due to the inability to straightforwardly include the AIPs in the formal model, an alternative option could be to query the AIPs throughout the verification process, whenever it requires the benefits prediction. However, existing model checkers do not support invoking external black box functions mid-way through the verification process. Supporting this feature is challenging given that external queries of black-box functions would cause the state of the model being checked to vary in unknown and unpredictable ways. This drastically reduces the possibility of applying path-pruning techniques in the verification process, which are crucial to managing state space explosion and enhancing efficiency of the verification process [49].

5.5.1 Ripple: Retrain ImPact Predictor for Long-tErm planning

Ripple extends the framework and enables long-term planning to determine when to *retrain* an ML model. To do so, and as shown in Figure 5.4, Ripple requires engineers to: **(1)** monitor the ML-enabled system that is in production and collect data D on the queries that the ML model receives and on the outputs that it generates; **(2)** build an application specific adaptation impact dataset (AID) based on data D; **(3)** based on the AID, train ***L****ook-****A****head*-AIPs (LA-AIPs); **(4)** create a formal model of the ML-enabled system, integrating the LA-AIPs; **(5)** deploy Ripple in production and plan when to retrain the ML model using a finite look-ahead L. Ultimately, Ripple needs as inputs the ML model's current predictive performance (i.e., quality matrix or confusion matrix), the old data that has been used to train the ML model, and the new data that the ML model might be re-trained with, so that Ripple can generate an expected optimal policy

for deciding when and when not to *retrain* over a fixed look-ahead horizon. The following sections elaborate on RIPPLE's building blocks.

Look-ahead adaptation impact dataset (LA-AID)

The LA-AID is the basis for building LA-AIPs and is constructed using a two-step process. The first step is to build an AID by following the process outlined in Section 5.3. This requires: (1) getting production data that characterizes the behavior of the environment (e.g., previous queries for the ML model and associated ground truth labels) during a fixed period, e.g., 2 months; (2) setting a time interval (e.g., hourly, daily, or weekly) and retraining the ML model at each time interval; (3) evaluating the performance of each retrained ML model on all future time intervals (without further retraining each model).

By dividing the production data selected for building the AID into t time intervals, this process yields R_i, $1 \leq i \leq t$ retrained models. To finalize the AID and compute the features that the AIPs employ when estimating the benefits of a retrain, it is necessary to compare the performance of different retrained models at specific time intervals. For instance, to get the benefits of retraining the ML model at time interval j, we compare retrained model R_j with each previously retrained model R_i, $1 \geq i < j$. Globally, by comparing all pairs of retrained models, we obtain an AID of size $O(t^2)$ (more precisely $\frac{t*(t-1)}{2}$), based on the t retrained models.

Once the AID is created, we can generate the LA-AID. Although as detailed in the next section the LA-AID construction varies slightly depending on the type of LA-AIPs that are implemented, there are still common steps that need to be performed. Specifically, the AID needs to be extended with the benefits of retraining models in future time intervals. For instance, for retrained model R_j, the LA-AID contains the benefits of this retrain at times $j+1, j+2, ..., j+L$, where L is the target look-ahead. Conversely, the AID only has the benefits at time j.

To create the AID and LA-AIDs we assume that the adaptation tactics executed do not influence the future data that the system receives. For instance, the fraud detection system has a single trace of transactions that is not affected by model retrains: the transactions that occur at time t are not affected by the adaptation tactic executed at time $t-1$.

Look-Ahead Adaptation Impact Predictors

We start by describing the challenges of creating LA-AIPs, and the trade-offs involved in selecting input features for LA-AIPs. Then, we present two alternative strategies to implement LA-AIPs, discussing their advantages and disadvantages.

Challenges of building LA-AIPs. Instantiating look-ahead adaptation impact predictors (LA-AIPs) is challenging for two main reasons. First, to leverage the model checker's long-term planning capabilities, the LA-AIPs have to be models of low complexity (e.g., linear models [57], shallow decision trees [16, 23]) such that they can be encoded in the formal model.

The second challenge, resulting from including the LA-AIPs in the formal model, is simulating how their input features (e.g., the state of the environment or the current predictive quality of an ML-model) will evolve within the look-ahead horizon. This is necessary because the model checker simulates the future states of the system during the verification process. Since the LA-AIPs are part of the formal model and depend on input features to make a prediction, the evolution of these features over the planning horizon must also be captured throughout the verification process.

Next, we discuss the trade-offs involved in selecting features for the LA-AIPs and then explain how we address these challenges and instantiate LA-AIPs.

LA-AIPs input features. For the initial version of the framework we investigated the use of a large number of alternative input features to estimate the variation of ML models' quality (cf. Section 5.3). However, when using model checking techniques to plan for the long term, the larger the number of input features used as input to the LA-AIPs, the larger the number of features whose evolution needs to be estimated throughout the planning horizon. This creates a trade-off between having LA-AIPs that rely on more features, thus potentially achieving better accuracy, versus using features that, due to their estimated nature are inherently noisy. Thus, when creating LA-AIPs it is crucial to keep into account not only the importance of input features for the predictive quality of the LA-AIPs, but also how accurately the evolution of these features can be estimated throughout the planning horizon.

Keeping these considerations into account, and given the goal of crafting low complexity, long-term predictors, the first step towards building LA-AIPs consists of a feature selection phase aimed at navigating the trade-off of LA-AIP accuracy versus noisy simulation of feature evolution. This can be performed using standard feature selection techniques [106] on the AIPs, since at this stage the goal is to understand which input features are more informative to the AIPs.

For instance, starting from an initial (large) set of available features, possible ways for selecting/discarding features that are expected to be of less importance to the LA-AIP correspond to: (i) looking at feature correlation between the available features and discarding features that are highly correlated with other features; (ii) training an initial predictor with all features available, looking at feature importance for that predictor, and selecting the most important (top N) ones.

Instantiating LA-AIPs. Regardless of the underlying modeling strategy (e.g., linear models, shallow decision trees) employed to instantiate LA-AIPs, the key missing piece for enabling long-term planning is simulating the evolution of the input features over the look-ahead horizon. Recall that this is required since the model checker needs the feature values at each step of the verification process to query the LA-AIPs and estimate the expected benefits of executing, or not, a *retrain*.

Out of the three feature sets for the AIPs (cf. Section 5.3), simulating the evolution of the environmental features and of the amount of data available for *retrain* is relatively straightforward since past data can be used to build time-series models that estimate the

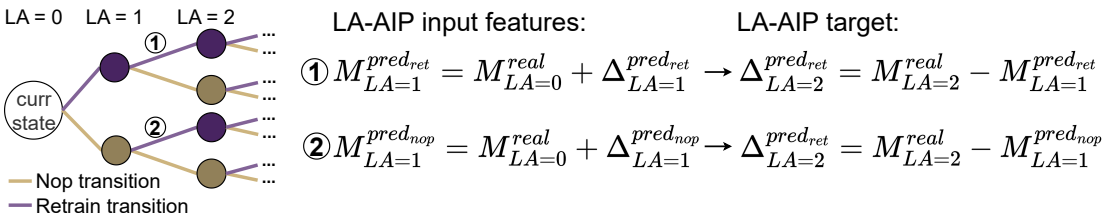


Figure 5.5: **Left:** tree representing the paths and future system states (nodes) explored by the model checker during simulation. **Right:** example for creating two input samples for the Prediction-based AID for LA=2. M represents the state of the ML model at each node in the tree. Δ represents the predictions of the different LA-AIPs. Note that $M^{pred_{nop}}_{LA=1}$ and $M^{pred_{ret}}_{LA=1}$ differ due to the transition that occurs prior to transitions 1 and 2.

future values for these features [33, 124, 162]. Similarly, keeping track of how long ago the *retrain* tactic was executed requires an additional variable in the formal model which does not significantly increase its complexity.

The major challenge lies in simulating how the features that characterize the state of the ML model evolve. This corresponds to estimating the expected benefits that each adaptation tactic will yield at each future step[1], which is what the LA-AIPs are trained to predict. Thus, they can be leveraged to compute the evolution of the state of the ML model during the verification process. While developing RIPPLE, we designed two alternative ways for creating LA-AIPs which we describe bloew, discussing their pros and cons. Both alternatives are grounded on the key principle of instantiating low-complexity versions (e.g., linear models, shallow decision trees) of the AIPs that can be embedded in the formal model of the ML-enabled system.

Prediction-based-LA-AIPs. The most straightforward approach to do long-term planning with the AIPs would be to always use the same AIP in a chained/recursive fashion, namely using its predictions for time t as input features for predicting the benefits at time $t+1$. However, given the chaining of the predictions, and the re-use of predictions as input features when estimating the benefits for the following simulation step, there is a mismatch between train (non-noisy data) and inference (noisy) data and non-negligible error propagation throughout the look-ahead horizon.

Prediction-based-LA-AIPs aim to address the mismatch problem by aligning the dataset used to train the LA-AIPs with the data used to query them in the planning phase. Thus, to build *Prediction-based-LA-AIPs*$_{LA=L}$ for look-ahead L, we create a LA-AID$_{LA=L}$ based on the (inherently imperfect) predictions of *Prediction-based-LA-AIPs*$_{LA=L-1}$.

Figure 5.5 illustrates the process of creating LA-AIDs for the *Prediction-based-LA-AIPs*. The tree shows the exploration paths simulated by the model checker during the verification process, given the two adaptation tactics (*retrain* and *nop*) available at each time-step. Let us focus on the two purple nodes at LA=2, reached via *retrain* transitions 1 and 2. Note

[1]Note that the predictive performance of the ML model depends on the sequence of tactics executed, while the evolution of the environment's state is independent from the adaptation strategy.

that transition 1 follows a prior *retrain* tactic, while transition 2 is preceded by a *nop* tactic. Consider transition 1. The AID used to train the *Prediction-based-LA-AIP*$_{LA=2}$ uses $M^{pred_{ret}}_{LA=1}$ as the ML model state input features and $\Delta^{pred_{ret}}_{LA=2}$ as its target. Note how $M^{pred_{ret}}_{LA=1}$ (the ML model state input features) is computed based on $\Delta^{pred_{ret}}_{LA=1}$, which is predicted by *Prediction-based-LA-AIP*$_{LA=1}$. Also, note how $\Delta^{pred_{ret}}_{LA=2}$ is computed as the difference between the *real* ML model state at LA= 2 ($M^{real}_{LA=2}$) and the *predicted* ML model state at LA= 1 ($M^{pred_{ret}}_{LA=1}$). This way, the prediction errors that are expected to be seen when the model checker is planning the adaptation strategy are accounted for at the time the LA-AIPs are created.

Note that these predictors need to be built sequentially, given that when training the *Prediction-based-LA-AIPs* for look-ahead L, one needs the predictions of the *Prediction-based-LA-AIPs* for look-ahead $L-1$. To start creating LA-AIDs for each look-ahead, we first need to obtain the predictions of the AIPs. Recall that the AIPs predict for look-ahead 1 based on the current state of the environment and of the system and, thus, the AID does not contain any predicted/noisy value (neither for the input features nor for the target).

Root-LA-AIPs. Although the *Prediction-based-LA-AIPs* address the mismatch issue previously described, the use of predictions of look-ahead L to compute both input features and target values for look-ahead $L+1$ leads to an error accumulation problem. *Root-LA-AIPs* address this problem by trading-off feature estimation error for feature staleness. Specifically, *Root-LA-AIPs* exploit the observation that only at the start of the simulation process do we actually know the current state of the ML model without error. Based on this insight, *Root-LA-AIPs* (also look-ahead specific) are trained to estimate the ML quality variation with respect to the known and error-free (albeit stale) ML state at the start of the simulation. This way, they avoid relying on error-prone estimations of the ML model state to generate predictions for the following time step.

Also in this case, to create *Root-LA-AIPs* for each look-ahead up to L we need L LA-AIDs: one LA-AID for each look-ahead. Each LA-AID is created from the base AID by going through all its samples, considering that each one is a possible root for a tree such as the one depicted in Figure 5.5. Then, for each root, the tree is constructed up to the look-ahead of the LA-AID under construction. Once the desired look-ahead is reached, all the nodes (i.e., possible future ML model states) at that depth are added as samples to the LA-AID. For each node n, the input features that characterize the state of the ML model are the ones at the root of the tree, and the target that the *Root-LA-AIP* will be trained to predict is computed as the difference between the real values of the ML model state at node n and the values of ML model state features at the root.

Regardless of the LA-AIP type that is instantiated, we need to predict the benefits both when the ML model is retrained and when it is not. Thus, we train two LA-AIPs for each look-ahead: a *retrain* LA-AIP and a *nop* LA-AIP. To train these LA-AIPs, the LA-AID for the corresponding look-ahead is sub-divided into *retrain* and *nop* transitions, which are used to train the corresponding LA-AIPs. For instance, in Figure 5.5, purple

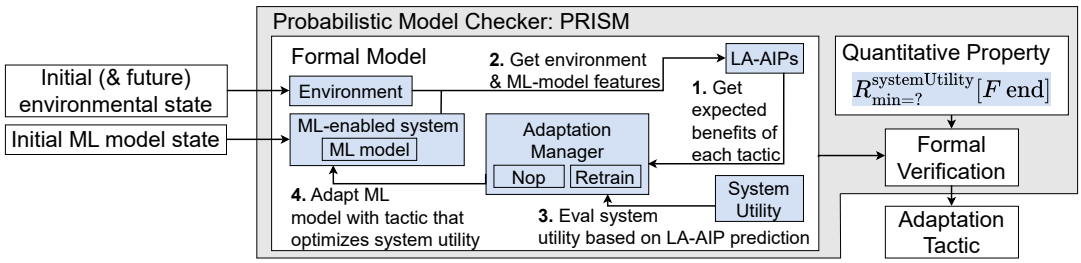


Figure 5.6: Modules of the formal model (in blue) required by RIPPLE.

nodes are used as input to *retrain* LA-AIPs, because they come from *retrain* transitions, while brown nodes are used for *nop* LA-AIPs.

Formal Model of the ML-Enabled System

The formal model of the ML-enabled system (Figure 5.6) is composed of modules for the environment, the adaptation tactics, and the ML-enabled system, which encapsulates the ML model abstraction. Additionally, and differently from the short-term implementation of the framework, RIPPLE's formal models incorporate the *LA-AIPs*. This is a key building block to plan for the long term when to retrain. Finally, the system utility component encodes the system-specific function to optimize.

The *LA-AIP* component encapsulates the logic that is used by the model checker to obtain, during the verification phase, estimates of the impact of the *retrain* and *nop* adaptation tactics on the predictive quality of the *ML-model*. RIPPLE needs as many LA-AIPs as the product of available tactics, metrics of interest and look-aheads. For instance, for the fraud detection use-case with SLAs on true positive rate (TPR) and FPR, RIPPLE would require 4 LA-AIPs for each look-ahead: TPR-retrain, TPR-nop, true negative rate (TNR)-retrain, and TNR-nop (recall that, for a normalized confusion matrix, FPR$=1-$TNR). Listing 5.1 (lines 2-3) shows how the TPR *Root-LA-AIPs* for look-ahead 2 for the fraud detection use-case are implemented in the PRISM model checker via linear-models. Specifically, these LA-AIPs rely on three key features, whose selection is explained in Section 6.2.4: fraud rate, the TPR of the fraud detection model at the time when the model checker was queried, and the TNR also at the start of the planning process. Recall from the previous section that the *Root-LA-AIPs* use the performance of the ML model at the *root* (i.e., LA$=0$) of the tree of possible exploration paths (Figure 5.5, left) to estimate the benefits of retrain at the target depth L. Estimating the benefits for other look-aheads requires instantiating more of these predictors in the formal model, replacing the coefficients associated with each feature and the noise term.

The system utility definition, which encodes the goal of the self-adaptive system, is defined via PRISM's notion of *rewards*. By defining the property that the model checker verifies as a function of the *rewards*, we can enable reasoning over quantitative measures of interest. For the fraud detection use-case, where the goal is to minimize the overall cost incurred by the system, the *rewards* keep track of the different costs that the system incurs: the cost of executing the *retrain* tactic, and the costs of violating either (or both) SLAs

Listing 5.1: Integration of the *Root-LA-AIPs* in PRISM's formal model & system utility definition via rewards function[2].

```
1  // TPR Root-LA-AIPs for look-ahead 2 and for tactics Retrain (ret) and Nop
2  formula LA_AIP_TPR_ret_LA2 = round(
3     0.6401*ret_Fraud_Rate + -0.9281*INIT_TPR + 1.7525*INIT_TNR + -1.1272*100
4  );
5  formula LA_AIP_TPR_nop_LA2 = round(
6     0.5036*nop_Fraud_Rate + -0.8167*INIT_TPR + 1.0348*INIT_TNR + -0.5498*100
7  );
8
9  // System utility definition
10 formula fpr_SLA_penalty = (fpr > FPR_SLA_THRESHOLD) ? SLA_PENALTY : 0;
11 formula tpr_SLA_penalty = (tpr < TPR_SLA_THRESHOLD) ? SLA_PENALTY : 0;
12 rewards "systemUtility"
13     [retrain_complete]  true & (time > 0) : (RETRAIN_COST);
14     [tick]              true & (time > 0) : (fpr_SLA_penalty + tpr_SLA_penalty);
15 endrewards
```

(TPR and FPR). These are shown in lines 8-11 of Listing 5.1: whenever a retrain completes, the system accrues the retrain cost; whenever time advances (there is a new time tick), the system accrues the FPR and TPR SLA violation penalties (lines 6-7). Based on this *rewards* structure, we define the quantitative property $R^{systemUtility}_{min=?}[F\ end]$ (cf. Figure 5.6) that is checked during the formal verification process. The definition of system utility is application dependent and when it is modified, the *rewards* and the quantitative property should be updated accordingly.

5.6 Summary

This chapter presented a framework to engineer self-adaptive ML-based systems, that generates policies that specify *when and how* to adapt an ML-based system. Additionally, the framework supports both short-term as well as long-term planning, and deals with ground-truth label delay by leveraging state-of-the-art ML methods for estimating ML model performance in the absence of ground-truth labels.

To create the framework, we addressed two main challenges: (i) estimating the benefits of the adaptation tactics available to adapt the ML component, and (ii) reasoning about how the execution of a specific adaptation tactic was expected to affect overall system utility. We solved the first challenge by creating adaptation impact predictors (AIPs), which are models trained to predict the expected benefits of executing an adaptation tactic. The second challenge was addressed by leveraging probabilistic model checkers and creating a formal model of the system under adaptation. This allowed to model the impact of an adaptation on other system components and reason about how an adaptation is expected to impact overall system utility.

[2]In PRISM commands are encoded as probabilistic state transitions following the format *[action] guard* $\rightarrow$ $prob_1$*:update*$_1$ + ... + $prob_n$*:update*$_n$. When *guard* is true, *update*$_1$ is applied with probability $prob_1$ (called transition probability). *action* allows to specify a name for the command or to synchronize commands between modules. Thus, commands with the same *action* are only triggered when all the *guard* of all commands is true.

Finally, we introduced Ripple, an approach for using the framework to plan for the long term when to adapt the ML component. Ripple leverages different AIPs that are integrated in the system's formal model to take advantage of the probabilistic model checker's state exploration mechanism and plan for the long term.

The following chapter presents the claims that are used to evaluate the thesis, evaluates the contributions, and validates the claims via two use-cases.

Chapter 6

Evaluation

This chapter evaluates the framework proposed in this thesis based on the following three claims:

1. The proposed framework leads to better system utility when compared against simpler baselines, such as periodic model updates.
2. The framework has low latency, rendering it usable in online planning settings.
3. The framework is generalizable and applicable to a range of scenarios and domains with different characteristics and ML models.

Each claim is further discussed in Section 6.1 and validated via two use cases from two distinct domains. The first use case is a self-adaptive fraud detection system (Section 6.2), which is representative of binary classification systems that leverage tabular data. Binary classification is commonly employed for medical diagnosis systems [62], spam detection systems, and intrusion detection systems [139], highlighting the relevance of testing the framework with binary classification ML-enabled systems. Additionally, the considered use case targets a highly impactful domain, namely AI-based financial fraud detection, whose market was valued at approximately 12.1 billion USD and is projected to grow to over 108 billion USD by 2033 [117].

The second use case (Section 6.3) considers a self-adaptive machine translation (MT) system, highlighting the framework's flexibility, since MT systems represent sequence-to-sequence problems (family of ML approaches typically leveraged to deal with natural language problems). Typical MT applications include (i) breaking language barriers for international business, diplomacy, and personal communication (e.g., news translation [75]); (ii) adapting products, services, and content to different languages and cultures; (iii) teaching non-native speakers to understand content with language learning tools (e.g., Duolingo [4]), and (iv) helping users via applications in travel, customer support, and social media where immediate translation is beneficial [143]. Software engineers have also started exploring the capabilities of MT models for tasks such as automatic program repair [87], code changes [177] and bug fixing [178], or code completion [46]. The wide applicability of MT models highlights the relevance of this use case and of the application of the proposed framework to this domain and class of systems.

Table 6.1: Framework capabilities and representation in the use cases.

	Fraud detection (Section 6.2)	Machine translation (Section 6.3)
ML self-adaptation	✅	✅
Extension to multiple tactics	✅	✅
Long-term planning	✅	❌

Table 6.1 shows the capabilities of the framework and the use cases in which those capabilities were implemented. Specifically, in both use cases the base ML-enabled system was turned into a self-adaptive system that attained higher system utility when compared against simpler baselines (e.g., periodic adaptation). This demonstrates the applicability of the framework and validates claim 1 (Sections 6.2.1 and 6.3). Then, both use cases demonstrate how the framework can be extended to account for multiple adaptation tactics: for the fraud detection system, we conducted a study showcasing how the formal model of the system could be extended to account for an extra tactic and how the PRISM model checker was capable of handling such extension (Section 6.2.3); the machine-translation system is either not adapted (tactic *nop*) or the ML model is fine-tuned. This tactic had not been considered in the fraud detection system, thus demonstrating how the framework can cope with varying adaptation tactics (Section 6.3). Finally, the extension to planning for the long term was only evaluated on the fraud detection use case (Section 6.2.4).

6.1 Claims

1. The framework leads to better system utility when compared against simple heuristic baselines, such as periodic model updates. This claim is evaluated by analyzing the ability of the framework to synthesize adaptation strategies that lead the system to achieve a utility that is closer to the optimal one than when simpler baselines are employed (such as periodically or reactively adapting the ML model). The general idea is that, given that the framework is equipped with predictors of the expected benefits and costs of executing each ML adaptation tactic, it will do a better job at trading off these two dimensions thus bringing system utility closer to the optimal.

2. The framework incurs low latency, rendering it usable in online planning settings. The framework is effective at optimizing system utility due to the contributions in the Analyze and Plan components. Namely, the adaptation impact predictors and the probabilistic model checking approach for synthesizing adaptation strategies. However, adding these components to the decision-making loop increases the time complexity of the overall approach for deciding whether to adapt, particularly when compared against the simpler baselines, which incur zero decision latency. Since the framework is meant for online use, its time complexity should be sufficiently low, such that it enables near real-time

adaptation decisions for domains such as recommender systems, enterprise systems, and Internet of Things (IoT) systems, but not necessarily for real-time safety-critical systems.

3. The framework is applicable to a range of scenarios and domains with different characteristics and ML models. Besides being feasible for online planning, the framework should also generalize to a multitude of domains and systems, regardless of their specific characteristics, such as type of ML model – e.g., random forest versus neural network – or ML task – classification versus sequence-to-sequence problems. Demonstrating this capability is achieved by instantiating the framework to render two different ML-enabled systems self-adaptive: the credit card fraud detection system (Section 6.2) and the machine translation system (Section 6.3).

6.2 Self-Adaptive Fraud Detection System

To demonstrate that the proposed framework can be used to instantiate self-adaptive ML-based systems, we engineer an adaptation manager for a ML-based credit card fraud detection system, based on a real credit card fraud detection system [53]. Typically, fraud detection systems rely on supervised binary classifiers to classify incoming (credit/debit card) transactions as either legitimate or fraudulent and have banks and merchants as their clients. In this domain, quality attributes of interest are for example the overall cost of service level agreement (SLA) violations. Hence, we consider that our system has SLAs on the target: (i) true positive rate (TPR) – percentage of fraudulent transactions actually caught – and (ii) false positive rate (FPR) – percentage of fraudulent transactions not caught – which should be kept within pre-defined thresholds:

$$\textsc{system}(\text{TPR}) \geq \text{TPR\_threshold};$$
$$\textsc{system}(\text{FPR}) \leq \text{FPR\_threshold};$$

SLA violations can occur when the ML component misclassifies a substantial amount of samples, such that either the TPR decreases below the threshold, the FPR becomes higher than acceptable, or both. These misclassifications are typically caused by environmental changes through data shifts: the input samples to the ML component change such that the ML model is no longer capable of correctly classifying those samples. This occurs for example when the amount of fraud in a given period increases or when fraudsters change their strategies (see Section 4.2) [32, 111].

Whenever these SLAs are violated, the system incurs non-negligible costs, which we assume are fixed (this assumption is discussed in Section 7.2.1. We further assume that the fraud-detection system is deployed in production and that new data is gathered continuously. Although we start by assuming immediate ground truth label availability, we later relax this assumption and evaluate CB-ATC the approach presented in Section 5.4 to deal with delayed ground truth labels for transactions that are available only d time units after the transaction has been processed (Section 6.2.2).

Periodically, the self-adaptation manager can decide to either do nothing (tactic *nop*), i.e., not to adapt the model, or to *retrain* the model, leveraging the newly collected data

and labels. We consider fixed *retrain* costs since the problem of estimating these costs has already been addressed in the literature [31, 194].

The framework solves the problem of deciding *when to retrain* such that the global cost given by the sum of SLA violation penalties and adaptation costs is minimized. System utility ($sysU$) is thus defined as:

$$\begin{aligned} &sysU = \text{total cost} = \\ &= \textsc{cost}(\text{TPR SLA violation}) + \textsc{cost}(\text{FPR SLA violation}) + \textsc{cost}(\text{tactic}), \end{aligned} \tag{6.1}$$

where total_cost is the global cost the system is expected to incur in L time units; COST(TPR SLA violation) and COST(FPR SLA violation) are the costs of violating the TPR and FPR SLAs, respectively, established for the system. The system is charged either of these costs when the monitored TPR is below the pre-defined TPR threshold and/or when the FPR is above the FPR threshold. Finally, COST(adaptation tactic) encodes the cost of adapting the ML model. This cost is set to zero for the *nop* adaptation tactic. Finally, although the definition of system utility is application-dependent, it is expected that the cost (e.g., monetary, environmental) and/or latency of adaptation will need to be accounted for, hence making this term of the equation general to other applications.

At a first stage we set the planning horizon L to one time unit, thus employing the myopic version of the framework. In this case, we capture *retrain* latency by having it weigh in on system utility: *retrain* latency is first translated into a percentage of the time period; for this percentage of time, system utility is computed based on the confusion matrix of the ML component that represented the state prior to the *retrain*; for the remainder of the time interval, system utility is computed based on the confusion matrix of the retrained model. Since the distribution of environment generated events may not be uniform, system utility is further weighted by the percentage of events in each period (during adaptation and after). Later, we show how the framework can plan for the long term (i.e., $L > 1$) *when* to adapt the ML model (Section 6.2.4).

6.2.1 Myopic Self-Adaptive Fraud Detection System

We start by describing the formal model of the system, illustrating key components resorting to PRISM syntax. These PRISM blocks can be re-used when applying the framework to additional use cases, requiring only slight modifications that are use case dependent (e.g., metrics of interest, repertoire of tactics, system utility definition). Then, we explain how probabilistic model checking is employed to extract the expected optimal adaptation tactic and how the repertoire of tactics can be extended. Finally, we discuss the AIPs required for this use case.

Formal model of the fraud detection system. The formal model of the system requires modules for each of the different moving parts that have an impact on the system. Thus, we model: (i) the environment under which the system is operating; (ii) the adaptation tactics – *nop* and *retrain*; and (iii) the ML component – to analyze how mispredictions

Listing 6.1: Adaptation manager and System rewards[1]

```
1  module adaptation_manager
2      selectTactic : bool init false;
3      currTactic : [none .. retrain] init none;
4
5      [newEvent] !selectTactic -> (selectTactic'=true)&(currTactic'=none);
6
7      // non-deterministic choice between adaptation tactics
8      [nop]     (selectTactic=true) -> 1:(currTactic'=nop)&(selectTactic'=false);
9      [retrain] (selectTactic=true)&(newData > 0) ->
10         1:(currTactic'=retrain)&(selectTactic'=false);
11
12     [tick] (currTactic != none) -> 1:(currTactic' = none);
13 endmodule
14
15 formula tactic_cost = (currTactic = retrain) ? retrainCost : 0;
16 formula fpr_violation_cost = (fpr > FPR_THRESHOLD) ? FPR_COST : 0;
17 formula tpr_violation_cost = (tpr < TPR_THRESHOLD) ? TPR_COST : 0;
18
19 rewards "systemUtility"
20     [tick] true & (time>0) : (tacticLatency * percentTxs * (
21         ((INIT_FPR > FPR_THRESHOLD) ? FPR_COST : 0)
22         + ((INIT_TPR < TPR_THRESHOLD) ? TPR_COST : 0)
23     ) + (1 - tacticLatency) * (1 - percentTxs) * (tacticCost + fpr_violation_cost +
       tpr_violation_cost));
24 endrewards
```

affect system utility, to simulate the execution of the tactics, and to understand the impact of mispredictions and adaptation tactics on system utility.

Since we assume that the two tactics (*nop* and *retrain*) cannot be executed simultaneously, we further consider an adaptation manager module that prevents this from happening and non deterministically selects which tactic to execute. As shown in Listing 6.1, whenever there is a new event generated by the environment (line 5) – for the fraud detection system an event consists of a batch of transactions – the adaptation manager enters the `selectTactic` state and can select to execute one tactic among the available ones[1]. For example, while tactic *nop* can always be executed (line 8), tactic *retrain* can only be executed when there is `newData` with which to train the ML model (line 9). Finally, since our approach assumes that time is divided into fixed-sized intervals, we further model a clock to keep track of the passing of each time interval. The clock module is implemented as in Moreno et al. [124].

Synthesizing optimal adaptation policies. As can be seen in Listing 6.1, each `tick` of the clock triggers the accrual of a reward[2] (lines 19-21). As defined in Equation (6.1),

[1]In PRISM commands are encoded as probabilistic state transitions following the format `[action] guard` $\rightarrow$ $\texttt{prob}_1$`:update`$_1$ `+ ... +` $\texttt{prob}_n$`:update`$_n$, where `guard` is a predicate over all variables in the model (including variables from other modules). When `guard` is true, $\texttt{update}_1$ is applied with probability $\texttt{prob}_1$ (called transition probability). Transition probabilities of a command must sum to 1. `action` allows to specify a name for the command or to synchronize commands between modules. Thus, commands with the same `action` are only triggered when all the `guard` of all commands is true.

[2]The basic building blocks of PRISM's syntax are modules, rewards structures, and formulas. Each module is composed of a set of variables and commands, which affect the variables belonging to the

system utility for the fraud detection system is given by the total costs incurred by the system due to the tactics executed and possible SLA violations. To encode this definition in the formal model, we use PRISM's reward structure (lines 18-22).

We account for *retrain* latency by considering that it takes up a percentage of the time interval to execute, which is given by `tacticLatency`. During this period, the system is receiving and classifying transactions. The transactions that are received while the tactic is on-going are classified resorting to the non-adapted ML model, while the remaining transactions are classified resorting to the adapted model. The percentage of transactions classified during the tactic execution period is encoded in `percentTxs`. Thus, if the non-adapted model violates any SLA (TPR or FPR), the cost the system incurs in is proportional to $\texttt{tacticLatency} \times \texttt{percentTxs} \times \texttt{sum of costs of SLA violations}$. For the remainder of the time instant, i.e., $1 - \texttt{tacticLatency}$, the remaining $1 - \texttt{percentTxs}$ transactions are classified with the adapted model, hence leading the system to possibly incur different SLA violation costs due to variations in the system's TPR and FPR. This is encoded in lines 20-21 of Listing 6.1.

To generate optimal adaptation strategies, we need to define a formal property for the PRISM model checker to verify. Since for this use case system utility is defined as the total costs incurred by the system (Equation (6.1)) and since the goal is to minimize these costs, the property that leads to the expected optimal adaptation strategy corresponds to minimizing system utility, which is defined in PCTL (reward-based property specification logic, cf. Section 2.1) as $\mathbf{R}^{\text{systemUtility}}_{\text{min=?}}[\mathbf{F}\text{ 'END'}]$, which means "*minimum system utility when time 'end' is reached*". Here $\mathbf{R}$ and $\mathbf{F}$ are the operators described in Section 2.1, min=? is the QUERY, and systemUtility specifies the reward structure to use as target. 'END' defines the simulation horizon, i.e., how many future time intervals we want the formal model to simulate, which in this case is set to 1.

Extending the tactic's repertoire. To reason about self-adaptation considering a broader set of adaptation tactics, the formal model needs to be changed only through the addition of the corresponding tactics' modules such that the adaptation manager can consider them as available when making its nondeterministic choice. This can be achieved by adding these tactics to Listing 6.1, in addition to *nop* and *retrain* (lines 8-9). This is demonstrated and evaluated in Section 6.2.3.

AIPs for the fraud detection system. As discussed in Section 5.3, the framework instantiates an AIP for each adaptation tactic. The AIPs are trained using the features presented in Table 6.2. In this case, since there are two adaptation tactics (*retrain* and *nop*), the framework instantiates two AIPs for each: one for predicting the increase/decrease in the true positive rate (TPR) and a second one to predict the true negative rate (TNR). Thanks to the properties of the confusion matrix, by predicting the future TPR and TNR,

module. The state of the MDP is given by the composition of all variables of all modules. The actions and transitions that the MDP can execute and take at a particular state are given by the commands that are enabled at a specific moment in time, by the different variables. Finally, the rewards that the MDP collects are specified with the rewards structure.

Feature group	Basic Features	Output Features
Feature name	**BF1.1:** $\lvert\mathcal{N}_j\rvert$, # transactions never used for training **BF1.2:** $\lvert\mathcal{I}_i\rvert$, # transactions used for training **BF1.3:** total data = $\lvert\mathcal{I}_i\rvert + \lvert\mathcal{N}_j\rvert$ **BF1.4:** ratio new-old data = $\lvert\mathcal{N}_j\rvert/\lvert\mathcal{I}_i\rvert$ **BF2.1:** current TPR **BF2.2:** current TNR **BF3.1:** time elapsed since the last retrain **BF4.1:** current fraud rate	scores-JSD

Table 6.2: Features used by the AIPs to predict the benefits of retraining and not adapting.

we can fully characterize the ML component's confusion matrix in the following time interval. These predictions are then provided as inputs to the formal model and leveraged by the probabilistic model checker to synthesize an optimal adaptation strategy.

To evaluate the improvements to system utility attained via the use of the framework and its tractability when applied to the myopic self-adaptive fraud detection system we consider the following research questions:

RQ1 Can the benefits of a model retrain be predicted with acceptable accuracy?

RQ2 Does the framework allow to improve system utility when compared against baselines such as periodic retrains, or reactive policies that retrain upon SLA violations?

RQ3 How do alternative execution contexts affect the gains achievable by the framework?

RQ4 Is the time complexity of the framework acceptable for online adaptation?

RQ5 What is the impact of label delay when estimating model performance?

These research questions evaluate and demonstrate the claims discussed in Section 6.1.

Experimental settings. We leverage Kaggle's IEEE-CIS Fraud Detection dataset [164] and the winning solution of the challenge [53] as basis for our implementation. The winning solution relies on an XGBBoost random forest model [39] that uses 216 features, including both features originally present in the IEEE-CIS Fraud Detection dataset as well as additionally engineered features. We utilize this winning solution to implement the data cleaning, and feature selection tasks. The data splits for training, validation, and test, the self-adaptation mechanisms, and the generation of the AID are then implemented on top of that base solution. Further, for the purpose of our use case we leverage only the train dataset of the Kaggle competition for which labels are available (the test dataset does not have labels of the transactions). Label availability is necessary to evaluate the predictive quality of the ML model and the benefits of retraining.

Also, we always ensure that the fraud transactions are fed to the ML model respecting their original timeline, as we do not wish to give any advantage to the model by providing it with information about the future. As such, we use the first 1/3 of the original Kaggle

Model type		Retrain S	Retrain M	Retrain L	NOP S	NOP M	NOP L
TPR	**MAE**	0.1141	0.1158	0.1162	0.1343	0.1254	0.1276
	PCC	0.6811	0.6727	0.6731	0.6165	0.5793	0.5737
TNR	**MAE**	0.0055	0.0060	0.0059	0.0066	0.0068	0.0068
	PCC	0.7436	0.7086	0.7121	0.6142	0.6008	0.5943

Table 6.3: AIP performance on different sets of features (S, M, L), evaluated resorting to the mean absolute error (MAE) and to the pearson correlation coefficient (PCC). *NOP* represents the AIPs that estimate the TPR and TNR when the model is not retrained.

train dataset to train (70%) and validate (30%) the initial fraud detection model. The remaining 2/3 are divided as follows: 70% are used for training and validation of the AIPs (80% and 20%, respectively), and the remaining 30% for testing the framework.

Throughout the evaluation, the cost of an SLA violation is fixed to 10 and the retrain cost is varied. This approach is justified as the costs/benefits of adaptation are determined by the relative values of these costs, rather than by their absolute values. Thus, by fixing the SLA violation cost and varying the retrain cost we can conduct a sensitivity analysis to evaluate the effectiveness of the proposed framework in a broad range of scenarios (including different retrain latencies).

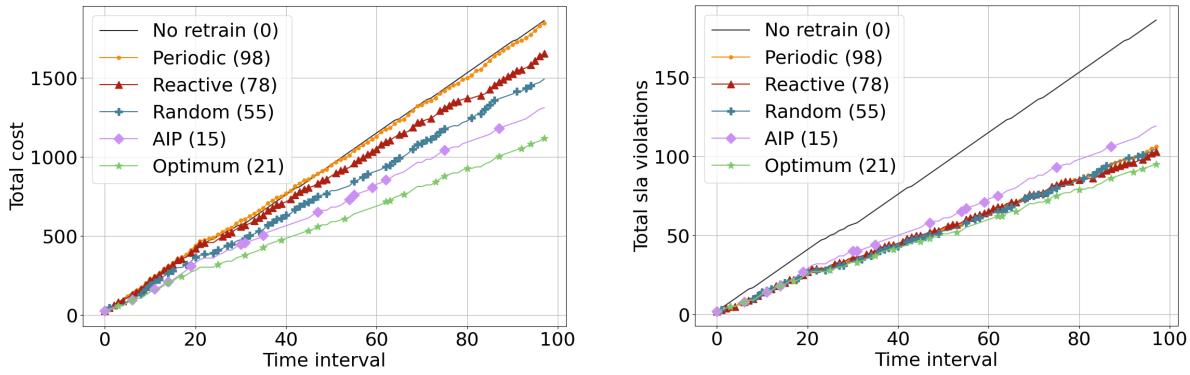
In this study, the AIPs are random-forest predictors of the sklearn package [140] with default parameter values except for the number of trees which is set to 12, similarly to the fraud detection ML model. The time interval is set to 10 hours and the horizon to one time unit. Our implementation is available at https://github.com/cmu-able/ACSOS22-ML-Adaptation-Framework.

Baselines. We consider the following baselines:

- **No-retrain:** the fraud detection ML model is never retrained;
- **Periodic:** the model is retrained at every time interval;
- **Reactive:** the fraud detection model is retrained whenever there is an SLA violation;
- **Random:** at each time step, there is a 50-50 choice that the model will be retrained;
- **Optimum:** this is the optimal (short-term) solution which is computed by looking at the actual future results of both retraining and not retraining the model.

Accuracy of the AIPs (RQ1)

This section answers **RQ1** by evaluating the performance of the AIPs resorting to the mean absolute error (MAE) and to the Pearson correlation coefficient (PCC), and considering different sets of features employed by each predictor. Specifically, we consider three different feature sets: S – minimal set with only the basic features (cf. Section 5.3); M – medium set, which includes the basic features and output characteristics; L – encompasses the features of the previous sets and the input characteristics. Table 6.3 displays these results. Interestingly, we see that an increase in the size and complexity of the feature set does not



(a) Cumulative cost.

(b) Cumulative SLA Violations.

Figure 6.1: Utility improvements achievable via the proposed framework. The execution context for this experiment is: FPR threshold = 1, TPR threshold = 70, retrain cost = 8, retrain latency = 0. The number of retrains executed by each baseline is shown in the legend of each plot, between brackets after the baseline's name. The retrains are represented by the markers in each line.

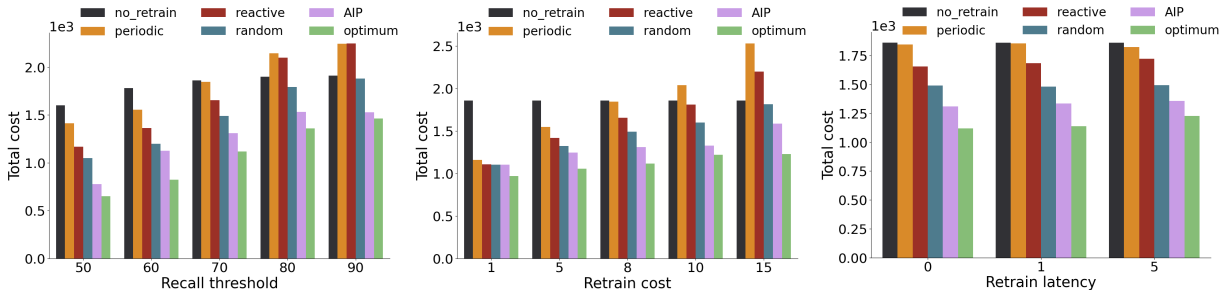
yield better AIPs. The results also show that the models responsible for predicting the future TPR and TNR when the model is retrained achieve a higher accuracy (lower MAE and higher correlation) than their *nop* counterparts (which predict the future TPR and TNR when the model is not retrained). Overall, on the one hand, the accuracy of the AIPs proposed in this work demonstrates that they can be employed to estimate the expected benefits of adaptation. On the other hand, the accuracy metrics reported in Table 6.3 confirm that predicting the future performance of ML models is far from trivial and that the proposed predictive methodology still has significant margins of improvement (e.g., by identifying different features, black-box predictors or combining white-box methods [55]).

Utility Improvement due to Retrain (RQ2)

Figure 6.1 compares the proposed framework (represented by line *AIP*) against the baselines. To evaluate whether the use of the framework allows to improve system utility over baselines that do not explicitly estimate the benefits of retrain, we define the SLA thresholds as TPR $\geq 70\%$ [8] and FPR $\leq 1\%$ [200], fix the retrain latency to 0 and the retrain cost to 8. These SLA threshold values were set based on values typically employed by related work in this domain [8, 200]. As Figure 6.1a shows, by leveraging the proposed framework it is possible to have the fraud detection system minimize its total costs and be closer to the optimal possible cost. This answers **RQ2** and shows that the framework does improve system utility over simpler, model-free baselines due to its ability to estimate the benefits of executing the retrain tactic. As for the number of SLA violations, as shown in Figure 6.1b, the AIP violates slightly more SLAs than all other baselines except No-retrain (the baseline that violates the most SLAs). However, as seen previously, this does not translate into higher incurred costs, which is the quality attribute under optimization.

TPR threshold	Retrain cost	Retrain latency
[50, 60, 70, 80, 90]	[1, 5, 8, 10, 15]	[0, 1, 5]

Table 6.4: Values tested for different execution contexts.



(a) Variation of TPR SLA (b) Variation of retrain cost (c) Variation of retrain latency

Figure 6.2: Impact of execution context on the total cost incurred.

Impact of Execution Context (RQ3)

To answer **RQ3** and evaluate how different execution contexts impact the need for retrain, we ran experiments for different SLA thresholds, retrain costs, and retrain latencies. Specifically, we tested the values shown in Table 6.4 for each dimension, fixing the remaining two dimensions to the values of the base scenario (TPR threshold = 70, retrain cost = 8, retrain latency = 0). Figure 6.2 displays these results.

Varying the TPR SLA. Regarding the TPR threshold (Figure 6.2a) the results show that the cost incurred by the approaches increases as the TPR threshold increases. This is justified by the fact that an increase in the TPR threshold yields a more difficult problem – the system tolerates less incorrect classifications of fraud transactions. This leads to more SLA violations, thus increasing the cost. The Optimum and AIP approaches also suffer a cost increase since retraining does not necessarily prevent them from violating the SLAs.

Varying the retrain cost. Focusing now on the retrain cost (Figure 6.2b) we see that if the cost is very low, the decision of whether to retrain is fairly trivial and so all approaches that retrain the ML component are close to the Optimum. However, as the retrain cost increases, we start to notice how carefully selecting when to retrain, accounting for the costs and benefits of the tactic, does pay off, as the AIP gets closer to the Optimum than the other approaches. As expected, the No-retrain approach is not affected by this dimension.

Varying the retrain latency. Finally, regarding the retrain latency dimension, the values tested correspond to percentages of the time interval that are occupied with the retrain execution. That is: retrain latency = 0 $\implies$ retrain is assumed instantaneous; retrain latency = 1 $\implies$ during the first 10% hours of the time interval the model is under retrain and transactions are classified using the existing, non-retrained model. The same rationale applies to retrain latency = 5. The results (Figure 6.2c) show that this dimension has

Total (mean)	Total (stdev)	PRISM (mean)	PRISM (stdev)
3.459	0.070	3.113	0.061

Table 6.5: Time overhead (secs) of the adaptation strategy generation process. The columns named 'PRISM' show only the time overhead due to verifying the formal model. The remaining columns display the total time overhead due to the AIPs and to the probabilistic model checking.

little impact on the cost of any approach, although the total cost of the Optimum solution increases slightly as the retrain latency grows. In fact, even if this baseline can always correctly determine whether it is worth it to retrain the model at any time t, if the retrain latency increases, a fraction of the transactions for the t-th interval are classified using an old model, thus increasing the amount of misclassifications and SLA violations.

Time Complexity (RQ4)

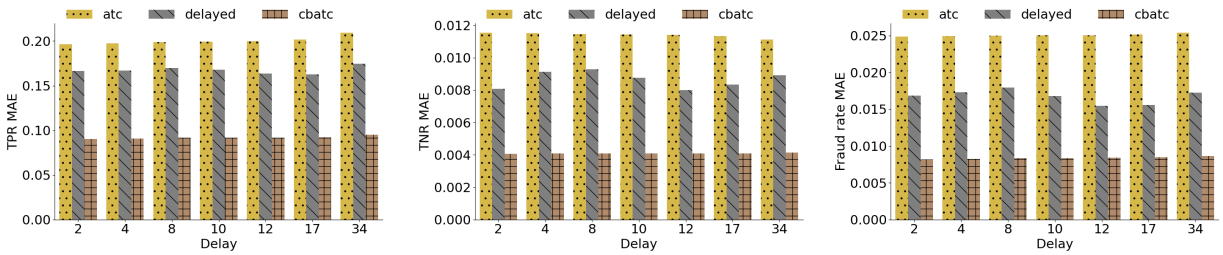
Since the purpose of the framework is to enable run-time adaptation of ML components to improve system utility, we evaluate the time complexity of the process of generating the adaptation strategy. This process corresponds to querying the AIPs and having PRISM verify the property of interest for the formal model of the system. Table 6.5 shows the average and standard deviation of the time overhead due to the whole process and of the formal model verification alone. These values correspond to the execution context defined in Section 6.2.1 and were obtained by running the experiments on a machine with an AMD EPYC 7282 CPU@2.8GHz, with 16 cores and 128GB RAM. As can be seen, the process of generating the adaptation strategy takes around 3.5 seconds, which is perfectly affordable considering that retraining ML components has a much higher time overhead. This answers **RQ4** and shows that the proposed framework can be employed to adapt ML-enabled systems online.

6.2.2 Impact of Label Delay

We now evaluate (i) the impact of label delay when estimating the performance of the ML model and (ii) how the accrued estimation error affects AIP performance. This allows us to understand the real impact of the assumption on ground truth label availability. To do so, we consider the following variants of our framework:

- **AIP:** assumes immediate label availability, delay= 0 (base version of the framework);
- **AIP_del:** estimates the current ML model performance based on the delayed metrics;
- **AIP_atc:** leverages ATC to estimate ML model performance;
- **AIP_cbatc:** leverages CB-ATC to estimate ML model performance;

We structure this study in two parts. We start by analyzing the effectiveness of ATC and CB-ATC (see Section 5.4) in predicting the current confusion matrix of the fraud-detection



(a) True positive rate (TPR). (b) True negative rate (TNR). (c) Fraud rate.

Figure 6.3: Mean Absolute Error (MAE) of the delayed, ATC and CB-ATC baselines.

classifier, as well as the fraud-rate in the environment. Recall that these constitute some of the input features to the AIPs. Next, we evaluate to what extent using ATC and CB-ATC to predict the input features for the AIPs impacts the effectiveness of the self-adaptive framework when optimizing system utility. Throughout this study, we consider delay intervals ranging from 2 to 34 time intervals, which correspond to, approximately, 1 and 14 days, respectively (recall that a time interval corresponds to 10 hours).

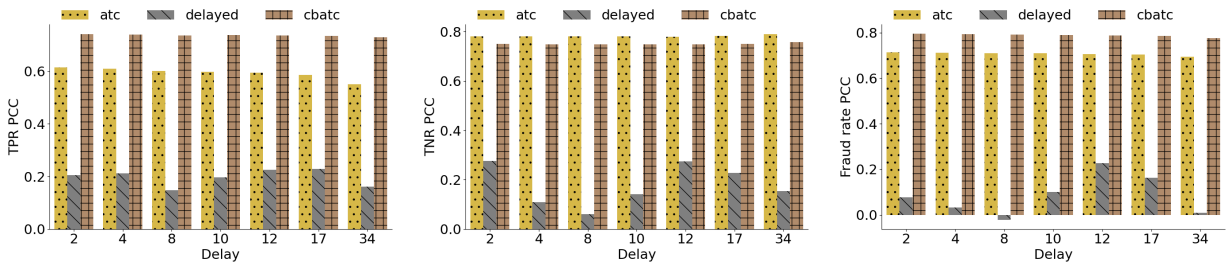
Estimation of the confusion matrix and fraud rate. Figures 6.3 and 6.4 evaluate the performance of both ATC and CB-ATC when estimating the current (i.e., before adaptation is enacted) TPR, TNR and fraud rate of the fraud detection ML model. Specifically, we report the mean absolute error (MAE) (Figure 6.3) and the Pearson correlation coefficient (PCC) (Figure 6.4) for each approach, and for each value of label delay tested.

Regarding the MAE, we observe that CB-ATC substantially reduces the estimation error for all metrics (TPR – Figure 6.3a; TNR – Figure 6.3b; fraud rate – Figure 6.3c) when compared against the delayed labels baseline. ATC however is outperformed by the delayed labels baseline. This is due to the fact that ATC was developed to estimate the accuracy of the classifier and not the performance of individual classes, which is the current case. Regardless, and particularly for the TNR and the fraud rate (Figures 6.3b and 6.3c) all approaches have a low estimation error.

In terms of the PCC (Figure 6.4) both ATC and CB-ATC present much higher correlations than the delayed approach for all metrics evaluated. In this setting, high correlation means that the estimations obtained by ATC and CB-ATC provide meaningful insights into the actual real values of the metrics. Overall, since CB-ATC is the approach that presents the best trade-off between MAE (low) and PCC (high), we expect it to yield better performance than the ATC and delayed baselines when leveraged by the framework.

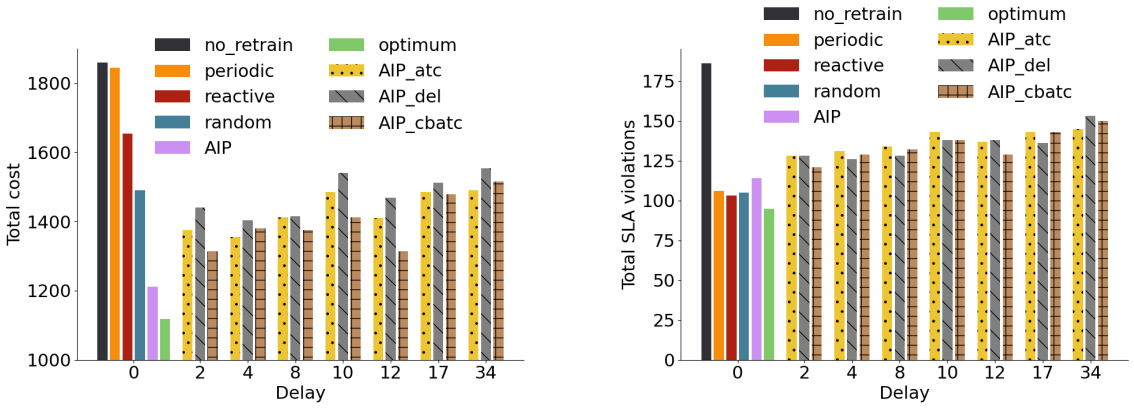
These results corroborate our hypothesis on ATC's limitations (Section 5.4) and show that CB-ATC can effectively solve them, yielding lower estimation error.

Impact on system utility. Next we evaluate the impact that using these approaches has on system utility (i.e., total cost) when dealing with different values of delay. Figure 6.5 reports the total cost incurred by each baseline for the same environmental settings of Figure 6.1. These results confirm the superiority of CB-ATC, which globally appears



(a) True positive rate (TPR). (b) True negative rate (TNR). (c) Fraud rate.

Figure 6.4: Pearson Correlation Coefficient (PCC) of the delayed, ATC and CB-ATC baselines.



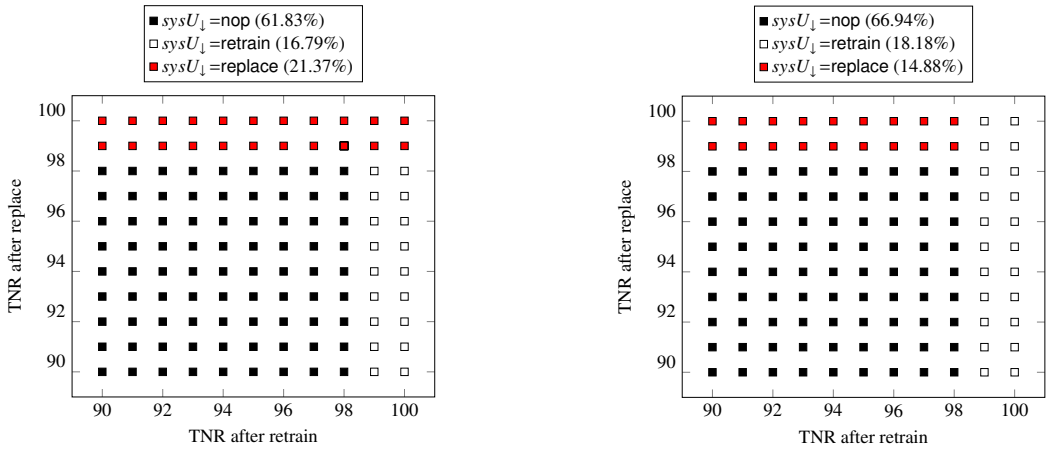
(a) Total cost for each delay value. (b) Total SLA violations for each delay value.

Figure 6.5: Total cost and SLA violations incurred by the different approaches when accounting for label delay. The values reported correspond to the last instant of the simulation, corresponding to time= 98 in Figure 6.1.

to be the most competitive solution across the considered delay values up to delay 12 (corresponding to 5 days). More precisely, the average total cost in the range of delay values [2, 12] for the approaches CB-ATC (AIP_cbatc), ATC (AIP_atc) and delayed metrics (AIP_del) is 1359, 1407 and 1453, respectively. This is expected given the conclusions drawn from the analysis of Figures 6.3 and 6.4.

Further, these results show that, for relatively small delay values (i.e., delay 2, corresponding to approximately 1 day), the use of CB-ATC allows the framework to achieve a system utility that is close to the one obtained in a setting where labels are immediately available (i.e., 8% worse than the AIP baseline). As the delay grows beyond 12 (i.e., 5 days), the performances of CB-ATC and ATC tend to degrade relatively to the delayed baseline that only has access to delayed information, leading them to achieve a performance that is on par with the random baseline. Conversely, for delays in the [2, 12] interval, CB-ATC achieves an average gain of approximately 10% with respect to the random baseline.

We suspect that the root cause of the problem is that the quality of the AIPs degrades significantly as the delay grows, independently of whether their input features are pre-



(a) Replace cost = 0 and retrain cost = 8 (b) Replace cost = 8 and retrain cost = 5

Figure 6.6: Optimal adaptation tactic selection as a function of the true negative rate (TNR) offered by the component replacement and retrain adaptation tactics. Recall that the TNR offers a direct proxy to the FPR SLA since $FPR = 1 - TNR$. The model checker is capable of selecting between multiple adaptation tactics by analyzing the expected cost-benefits offered by each and selecting the one that optimizes system utility.

dicted via ATC or CB-ATC or if delayed values are used. Regardless, and as previously observed, CB-ATC can predict these features more accurately than an oracle that simply outputs delayed information. We argue that this limitation might be imputed to the modeling approach that we currently use to construct the AIPs, which can be addressed by investigating alternative modeling techniques and different feature engineering methods.

6.2.3 Extending the Repertoire of Adaptation Tactics

Although for this use case we created AIPs only for the *nop* and *retrain* tactics, the planning component (i.e., the probabilistic model checker) of the framework can still be used to support "what-if" analysis aimed at identifying in which scenarios the use of additional adaptation tactics optimizes system utility.

To demonstrate the extensibility of the framework, we consider the availability of a third tactic, which we refer to as *component replacement*: replacing the ML model used for fraud detection with a rule-based model defined by human experts. We assume that the rule-based model offers a fixed TPR of 75% and that the current performance of the ML model is TPR= 75% and TNR= 98%. The SLA thresholds and costs are the ones previously defined. We then use the planning component of our framework to conduct a what-if analysis that aims to identify the optimal adaptation tactic when varying: (i) the TNR for the rule-based model, (ii) the TNR for the ML model after it is retrained, and (iii) the costs for the *retrain* and *component replacement* tactics.

Figure 6.6 shows the results of this analysis, indicating with different colors the optimal tactic for each setting of: rule-based model TNR, retrained ML model TNR, and tactic costs. The figure reports on the x-axis the TNR of the ML model after retrain and on the

y-axis the TNR of the rule-based model. Note that Figure 6.6a and Figure 6.6b consider different execution costs for each tactic. This study demonstrates that the model checker is capable of selecting between multiple adaptation tactics by reasoning on the impact of each alternative tactic on system utility and identifying the one that yields maximum gains.

6.2.4 Long-Sighted Self-Adaptive Fraud Detection System

This section evaluates Ripple (Section 5.5.1), addressing the following research questions:

RQ1 What is the predictive quality of the LA-AIPs?

RQ2 What are the system utility gains achieved by Ripple?

RQ3 Under which conditions is long-term adaptation useful?

RQ4 Is Ripple's latency acceptable for online planning?

RQ5 How expensive is it to train LA-AIPs?

These research questions provide further evidence regarding the tractability of the framework and its ability to lead to system utility gains compared to simpler adaptation heuristics in the more challenging scenario of long-term adaptation.

Experimental settings. For this study we use the same experimental settings as before. Specifically, we use Kaggle's IEEE-CIS Fraud Detection dataset [164]: the first third of Kaggle's train dataset is used to train the base fraud detection model (70% train, 30% validation). The remaining 2/3 are used to create the AIDs: 70% for training and validation (20% for validation) and the remaining data (30%) to test the framework, simulating an actual self-adaptive fraud detection system in production. By default, Ripple retrains at the first time-interval.

For evaluation purposes, the SLA violation cost is set to 10 and the FPR SLA threshold to 1% [200]. The retrain cost and latency, TPR SLA threshold, and look-ahead parameters are varied throughout the study. To estimate the total number of transactions, number of fraud transactions, and fraud rate in future time steps, we employ ARIMA time-series models from the Statsmodels library [160].

The experiments were performed in a virtual machine with 32 virtual CPU cores running on an Intel(R) Xeon(R) Gold 5320 CPUs at 2.2GHz and 48GB of DDR4 RAM. The virtual machine uses Ubuntu 22 LTS, Java 11, and version 4.8.1 of PRISM.

Baselines. To evaluate Ripple we consider the following baselines:

- **No-retrain:** never retrain the model – this baseline employs the initially trained fraud detection ML model;
- **Periodic:** retrain the model at every time step;
- **Reactive:** retrain the model upon any SLA violation;
- **Random:** retrain the model at each time step with 50-50 probability;

Table 6.6: Feature importance of the top 5 most important features for the random forest AIPs.

Feature Set	Retrain-TPR Feature	Imp.	Nop-TPR Feature	Imp.	Retrain-TNR Feature	Imp.	Nop-TNR Feature	Imp.
Small	curr-TPR	66.47	curr-TNR	56.02	curr-TPR	47.16	curr-TNR	48.46
	currFraudRate	28.74	currFraudRate	39.88	currFraudRate	38.16	currFraudRate	40.22
	ratioNewOldData	2.69	curr-TPR	3.15	ratioNewOldData	9.61	curr-TPR	5.35
	curr-TNR	1.71	ratioNewOldData	0.73	curr-TNR	3.78	ratioNewOldData	4.17
	prev-TNR	0.2	prev-TPR	0.12	prev-TPR	0.66	prev-TNR	0.91
Medium	curr-TPR	65.46	curr-TNR	55.82	curr-TPR	46.25	curr-TNR	47.68
	currFraudRate	26.14	currFraudRate	36.89	currFraudRate	33.05	currFraudRate	37.38
	ratioNewOldData	2.33	curr-TPR	2.34	ratioNewOldData	9.05	curr-TPR	4.17
	currUncertaintyFraud	2.06	currUncertaintyFraud	2.27	currUncertaintyFraud	4.02	ratioNewOldData	3.03
	curr-TNR	1.48	ratioNewOldData	0.64	curr-TNR	2.75	currUncertaintyFraud	2.58
Large	curr-TPR	64.76	curr-TNR	55.49	curr-TPR	45.39	curr-TNR	46.91
	currFraudRate	23.17	currFraudRate	35.55	currFraudRate	29.92	currFraudRate	35.1
	currUncertaintyFraud	1.66	currUncertaintyFraud	2.02	ratioNewOldData	7.69	curr-TPR	3.32
	ratioNewOldData	1.52	curr-TPR	1.97	currUncertaintyFraud	3.21	currUncertaintyFraud	1.91
	curr-TNR	1.07	scores-JSD	0.41	curr-TNR	2.02	ratioNewOldData	1.72

- **Optimum:** optimum oracle, which has perfect knowledge about the future, and, unlike the previous baselines, is sensitive to the look-ahead parameter. Hence we define both myopic (look-ahead = 1) and long-sighted (look-ahead > 1) optimum oracles;
- **Myopic Ripple:** retrain the model according to Ripple's decision with look-ahead = 1 (i.e., predict what is expected to happen in the immediate following time interval when each adaptation tactic is executed).

Predictive Quality of the LA-AIPs (RQ1)

To answer **RQ1** and evaluate the predictive quality of the LA-AIPs, we (i) start by analyzing the feature importance of the top 5 features for the AIPs based on the three feature sets; then (ii) we compare various modeling approaches that can be leveraged to instantiate LA-AIPs; and finally (iii) compare the methodologies presented in Section 5.5.1 for creating LA-AIPs.

LA-AIP feature set. Table 6.6 shows the top 5 most important features for the AIPs as a function of the feature set (small, medium, large). We see that the top 5 features are fairly consistent regardless of the feature set. Based on this analysis, we select the following features for the LA-AIPs: current TPR, current TNR and current fraud rate. These features correspond to approximately 90% of feature importance for the majority of the AIPs that were tested.

LA-AIP underlying model selection. To select the modeling approach to instantiate the LA-AIPs we started by comparing several low complexity approaches with the previously selected feature set (i.e., current TPR, current TNR and current fraud rate). Specifically, we tested linear models, shallow decision trees (maximum depth of 3), and three types of support vector machine (SVM) models with different kernels (radial basis function – rbf, polynomial – poly, and linear – lin). We evaluated the models via the Mean

Table 6.7: Mean Absolute Error (MAE) and Pearson Correlation Coefficient (PCC) of the AIPs in the test-set with feature set [`curr-TPR, curr-TNR, curr-fraud-rate`].

Type	MAE				PCC			
	TPR		TNR		TPR		TNR	
	RET	NOP	RET	NOP	RET	NOP	RET	NOP
RF	0.1170	0.1349	0.0065	0.0075	64.75	52.74	67.28	51.33
Lin	**0.1057**	0.1221	0.0058	0.0066	**73.31**	68.20	75.20	**63.79**
DT	0.1080	0.1225	0.0062	0.0073	71.12	63.86	69.61	51.50
SVM rbf	0.1068	0.1277	0.0068	0.0079	70.14	59.67	64.15	44.80
SVM poly	0.1302	0.1305	0.0068	0.0074	57.53	56.48	58.91	50.16
SVM lin	0.1070	**0.1210**	0.0055	**0.0064**	72.75	**68.61**	**75.28**	63.58
Cubist	0.1097	0.1241	**0.0054**	0.0068	69.62	61.30	74.42	57.31

Table 6.8: Mean Absolute Error (MAE) and Pearson Correlation Coefficient (PCC) attained by the three types of LA-AIPs. Bold font highlights the overall best MAE and PCC attained for each of the four predictors (TPR-retrain, TNR-retrain, TPR-nop, TNR-nop).

Error Metric	Tactic	Predictor	TPR LA 1	TPR LA 2	TPR LA 3	TPR LA 4	TPR LA 5	TNR LA 1	TNR LA 2	TNR LA 3	TNR LA 4	TNR LA 5
MAE	RET	AIPs (large)	**0.114**	—	—	—	—	0.006	—	—	—	—
		Chained	0.115	0.113	**0.115**	0.113	0.112	0.006	0.006	**0.006**	0.006	**0.005**
		Pred-based	0.115	0.113	**0.115**	0.113	0.112	0.006	0.006	**0.006**	0.006	**0.005**
		Root	0.115	0.113	0.116	0.113	0.112	0.006	0.006	0.007	0.006	0.006
	NOP	AIPs (large)	**0.120**	—	—	—	—	0.007	—	—	—	—
		Chained	0.141	0.146	0.147	0.147	0.145	0.007	0.007	0.007	0.0068	0.0068
		Pred-based	0.141	0.127	0.121	0.115	0.112	0.007	**0.006**	**0.006**	0.0065	**0.0056**
		Root	0.141	**0.119**	**0.113**	**0.111**	**0.109**	0.007	**0.006**	0.007	**0.0060**	0.0060
PCC	RET	AIPs (large)	65.80	—	—	—	—	69.33	—	—	—	—
		Chained	**68.11**	29.52	18.73	15.01	11.74	**65.73**	24.21	42.05	44.68	43.19
		Pred-based	**68.11**	29.00	21.50	8.84	12.21	**65.73**	26.10	34.28	10.02	38.96
		Root	**68.11**	**73.59**	**72.57**	**71.78**	**72.32**	**65.73**	**63.26**	**68.47**	**74.51**	**66.14**
	NOP	AIPs (large)	61.34	—	—	—	—	54.63	—	—	—	—
		Chained	**67.37**	21.98	1.94	-9.76	-10.32	**66.04**	5.12	26.98	36.86	34.63
		Pred-based	**67.37**	15.76	3.83	12.61	6.57	**66.04**	14.74	33.94	-25.09	37.15
		Root	**67.37**	**72.48**	**72.14**	**71.45**	**70.99**	**66.04**	**61.54**	**66.60**	**71.31**	**60.67**

Absolute Error (MAE) and Pearson Correlation Coefficient (PCC) and only compared predictors for look-ahead 1 (i.e., myopic predictors).

Table 6.7 shows these results, along with the performance obtained by (more complex) random forest models (100 decision trees of depth 12), which corresponds to the underlying modeling strategy employed for the (short-term) AIPs. All modeling approaches offer similar performance both in terms of MAE and PCC, with the linear models and the SVM with linear kernel offering the best PCCs. Given this empirical analysis, and considering that linear models are simpler and easier to implement in formal models, and that they provide, at least for the considered use case, comparable performance guarantees to more complex alternatives, in the following sections linear models are adopted as the underlying modeling approach for the LA-AIPs.

Look-ahead adaptation impact predictors (LA-AIPs). Table 6.8 compares each LA-AIP type (prediction-based, and root – see Section 5.5.1) using features: current TPR, current TNR and current fraud rate. We include in the comparison also the AIP with the

large feature set and a simple baseline based on the AIPs, called chained-AIPs. We report the Mean Absolute Error (MAE) and Pearson correlation coefficient (PCC).

The chained-AIPs are a straightforward instantiation of the LA-AIPs. They are based on the key rationale that since the AIPs predict deltas (Δ) (i.e., variations in ML predictive quality) based on the current quality of the ML model, one can chain them and re-use their predictions as input features to estimate the ML model's Δ for the following simulation step. While chained-AIPs are simple to implement, the chaining of the predictions when estimating the benefits for the following simulation step leads to non-negligible error propagation throughout the look-ahead horizon.

With respect to MAE, all LA-AIP types (chained, prediction-based, and root) offer similar performance. Further, predicting the TPR appears to be a more challenging predictive problem (higher MAE) than predicting the TNR. This is because the TNR has only small variations (always around 98% and 99%) whereas the TPR has much larger variations (e.g., ranging from 50% to 70%).

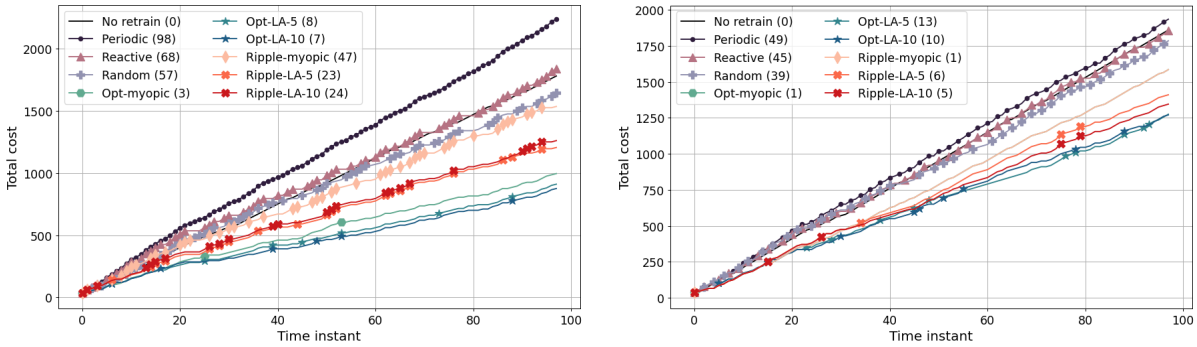
Regarding the PCC, we see that: (i) all approaches perform equally well for look-ahead 1, which is expected given that they differ only for higher look-ahead horizons (recall that for look-ahead 1, all predictors are trained based on the same dataset and they all have access to up-to-date feature values); (ii) the *root-LA-AIPs* provide, by far, the best correlation across all look-aheads analyzed (1 to 5), offering correlations typically around 70% for the TPR and 60% for the TNR; (iii) the *chained-LA-AIPs* have the worst performance across the considered approaches, which shows that the chained invocation of the AIPs has an unacceptably low accuracy; (iv) *Prediction-based-LA-AIPs* offer only slightly better PCC than the *chained-LA-AIPs*, remaining by far inferior to the *root-LA-AIPs*. We argue that this is due to the fact that, despite avoiding the mismatch between using noisy/non-noisy input features when querying/training the LA-AIPs, the prediction-based approach still suffers from error accumulation as the look-ahead increases.

RQ1. Our results demonstrate that: (i) *root-LA-AIPs outperform the other strategies for instantiating long-term predictors*; (ii) the predictive performance of the *root-LA-AIPs* is consistent across the considered look-aheads; (iii) when coupled with the *root-LA-AIPs*, linear models can predict the evolution of the TPR and TNR for *the long term* with a quality comparable to that achieved by the myopic, black-box AIPs (cf. Section 6.2.1).

Utility Improvement due to Ripple (RQ2)

To evaluate how Ripple improves over myopic approaches due to its ability to plan for the long term, we show in Figure 6.7 the total cost incurred by all baselines. For this analysis, we set the retrain cost to 15. Overall, the results answer **RQ2** by showing that in our case study *Ripple's long-term adaptation improves over myopic adaptation by up to* 21.4%.

Figure 6.7a considers a scenario in which there is no retrain latency and the TPR threshold is set to 60%. The figure shows the (cumulative) total cost over time achieved by each baseline. The markers highlight instants when the ML model was retrained and the values between brackets in the legend next to each baseline indicate the number of retrains executed. We verify that Ripple improves over its myopic counter part, which in



(a) TPR = 60 and Retrain-latency = 0

(b) TPR = 70 and Retrain-latency = 2

Figure 6.7: Total cost incurred by each baseline when retrain-cost = 15. The values between brackets indicate the number of retrains executed by each approach. Each marker highlights a time instant when a *retrain* adaptation was performed.

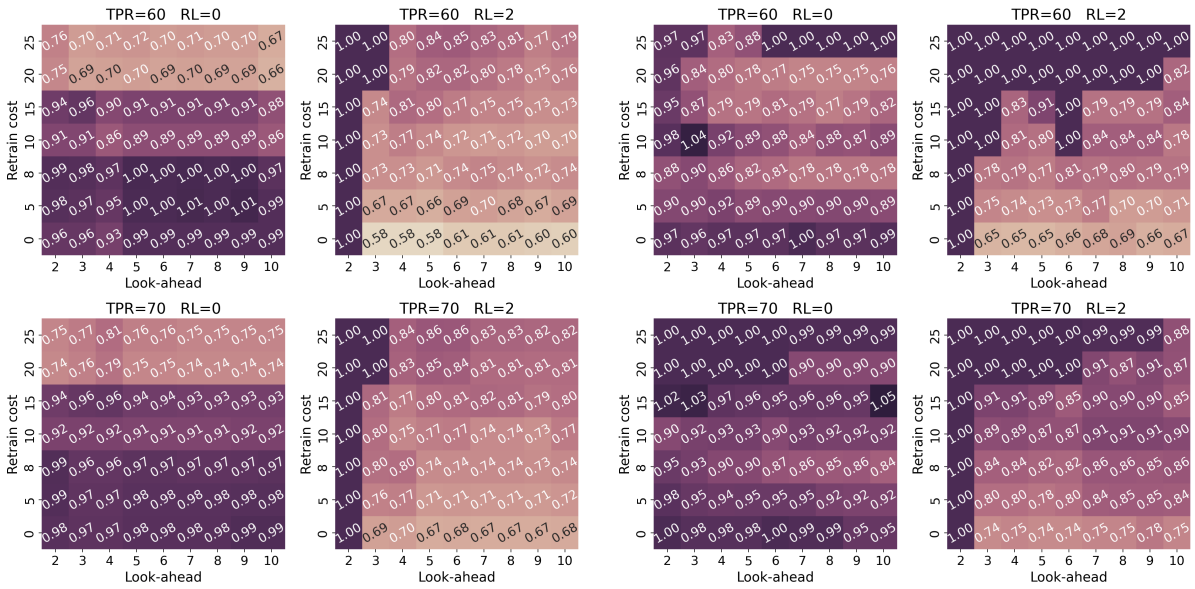
turn improves over the naive baselines. Specifically, Ripple with look-ahead 5 achieves a total cost of 1205, while the myopic version has a cost of 1535 (a reduction of 21.4%). We also see that Ripple with look-ahead 10 incurs a cost of 1260, which is higher than the cost obtained by the version with look-ahead 5. This is not surprising since predicting for longer look-ahead horizons is harder than for short look-ahead horizons: the further into the future one aims to predict, the higher prediction error [31, 101, 124]. This can ultimately lead the planner to make incorrect decisions when adopting long planning horizons.

In Figure 6.7b we consider a scenario where executing a retrain takes 2 time intervals. We see that both myopic baselines (optimum and Ripple) perform only the initial retrain (at time 0). This is due to the fact that myopic Ripple cannot properly evaluate the benefits of executing a retrain since it evaluates only a single future time interval. However, as the retrain takes 2 time intervals to complete, myopic Ripple perceives a retrain as having exactly the same impact of not retraining, except at a cost. Hence it decides to never retrain. Conversely, the long-sighted versions of Ripple get closer to the corresponding optimal oracles, achieving a cost reduction of 15.14% with respect to myopic Ripple. This demonstrates how long-term planning allows systems to become latency-aware and improve overall system utility.

Impact of System/Environmental State (RQ3)

To understand the landscape of possible improvements that long-term adaptation can yield, we investigated how several tunable system variables impact both the need for and gains achievable by adaptation. Thus, we varied the following parameters: SLA on the system's true positive rate (TPR), retrain cost (RC), retrain latency (RL), and the look-ahead. The optimum oracle is the only baseline considered because it is the only one sensitive to the look-ahead parameter.

The results can be observed in Figure 6.8, where we show the total cost (i.e., system utility) attained by the optimum (Fig. 6.8a) and by Ripple (Fig. 6.8b) normalized to the



(a) Optimum. (b) Ripple.

Figure 6.8: Total cost obtained by Ripple and Optimum in multiple execution contexts, varying the retrain cost, retrain latency, look-ahead, and TPR SLA threshold. The costs are normalized to the cost of the myopic approach (look-ahead=1), hence cells with values lower than 1 indicate scenarios for which long-term adaptation is beneficial.

total cost obtained by the (corresponding) myopic baseline (i.e., the total cost achieved by long-term optimum is normalized to the cost achieved by myopic optimum and the cost achieved by long-term Ripple is normalized to the cost achieved by myopic Ripple). Recall that the goal of the self-adaptive system is to minimize the total cost incurred, thus values lower than 1 represent gains with respect to the myopic counter part of each approach (Optimum and Ripple).

Overall, a first high-level analysis of the Optimum (Fig. 6.8a), shows that for any (fixed) retrain cost, there is at least one look-ahead for which long-term planning improves system utility (i.e., lowers total cost).

In practice, Ripple's results also demonstrate benefits stemming from long-term adaptation: when retrain latency is accounted for (RL= 2 and RL= 4) Ripple improves over the myopic version for retrain costs up to 20. Further, when no retrain latency is considered (RL= 0), our results show that Ripple, via its long-term planning capabilities, improves overall system performance even in scenarios where the retrain cost is 25 (recall that SLA penalties can amount to at most 20 in a single time interval). This shows that by looking ahead, Ripple can identify situations in which delaying retraining can lead to greater benefits in the long term. This could be due to the environment changing in non-trivial ways or because the current data available for the retrain might not result in a better-performing ML model under the current environment.

Comparing Ripple with Optimum, we see a clear performance gap, particularly when considering higher retrain costs: although Ripple can improve system utility with retrain

costs up to 25 (e.g., TPR= 70 and RL= 2), this occurs only for very specific look-ahead values. Instead, Optimum shows clear benefits with a range of look-aheads. This shows that although the performance of the *root-LA-AIPs* is always consistently around 60% and 70% correlation, there are still margins for improving long-term prediction.

Finally, regarding the TPR SLA threshold, we see that as it increases, the gap between the long-term and myopic approaches decreases (all values get closer to 1). The reason for this is that the TPR affects the difficulty of the underlying problem of fraud detection: higher TPR means that the system has to make fewer mistakes and correctly flag more fraudulent transactions. With higher TPR SLAs even if the system retrains, it might still not comply with the SLA. This intuitively justifies why all values get closer to 1.

RQ3. When retrain has latency, *Ripple's long-term planning provides benefits of up to* 35% (TPR= 60, RL= 2, RC= 0, LA=[3, 4, 5]) *and of up to* 25% (TPR= 60, RL= 0, RC= 20, LA=[7, 8, 9]) when retrain latency is not considered. Typically, higher retrain costs hinder Ripple's ability to reap the benefits that long-term planning yields. Additionally, higher TPR thresholds also lead to lower benefits, since it is more challenging to comply with the SLA. Finally, we verify empirically that mid-range look-ahead horizons (e.g., 4, 5) typically provide improved system utility. This is aligned with the previous observation (cf. RQ2) that excessively long look-ahead horizons can lead to sub-optimal planning decisions.

Feasibility for Online Planning (RQ4)

Since Ripple aims to enable the online reasoning of when to retrain ML-enabled systems, we analyze the time complexity of the formal verification process (i.e., how long it takes to determine the next adaptation tactic to execute) as a function of the look-ahead. Table 6.9 reports these results and shows the average latency, and the 95-th and 99-th percentiles.

Overall, the results shows that incrementing the look-ahead naturally leads to an increase of the latency to extract the adaptation strategy. However, even for look-ahead 10 it takes on average less than 3 seconds to extract the adaptation strategy, and 3.51 seconds on the 99-th percentile. Regarding **RQ4** we verify that *Ripple's latency is fast enough for a large majority of non-safety-critical ML-enabled systems* (e.g., fraud detection, machine translation, and recommendation systems).

Cost of Training LA-AIPs

When creating LA-AIPs, the dominant cost is associated with building the AID, given that this entails performing multiple retrains of the ML model under adaptation. As such, we conduct a sensitivity analysis aimed at studying how reducing the number of *retrains* performed to build the AID impacts the predictive quality of the LA-AIPs.

Specifically, we sub-sample the AID at random to reduce the number of retrains used to gather information on ML retrain benefits. The resulting AID is then used to train the *root-LA-AIPs*, which are evaluated on the full test-set. Each sub-sample percentage is repeated 10 times for reliability. The results are depicted in Figure 6.9, which shows that the performance of the LA-AIPs remains largely unaltered when using up to 30%

Table 6.9: Latency of the adaptation strategy generation process (in seconds) and model size as a function of the look-ahead (recall that a myopic system has look-ahead 1).

LA	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Avg ± stdev [s]	2.64 ± 0.36	2.70 ± 0.35	2.71 ± 0.38	2.77 ± 0.39	2.80 ± 0.41	2.80 ± 0.41	2.87 ± 0.38	2.81 ± 0.44	2.87 ± 0.42	2.90 ± 0.40
95-perc [s]	3.13	3.15	3.25	3.29	3.31	3.32	3.35	3.37	3.41	3.42
99-perc [s]	3.20	3.22	3.32	3.36	3.37	3.39	3.43	3.44	3.49	3.51
Num states (min)	13	62	194	303	444	582	720	841	985	1081
Num states (max)		77	336	524	710	917	1138	1352	1587	1837
Num transitions (min)	14	70	229	347	503	644	791	921	1074	1250
Num transitions (max)		88	402	622	835	1074	1329	1574	1836	2125

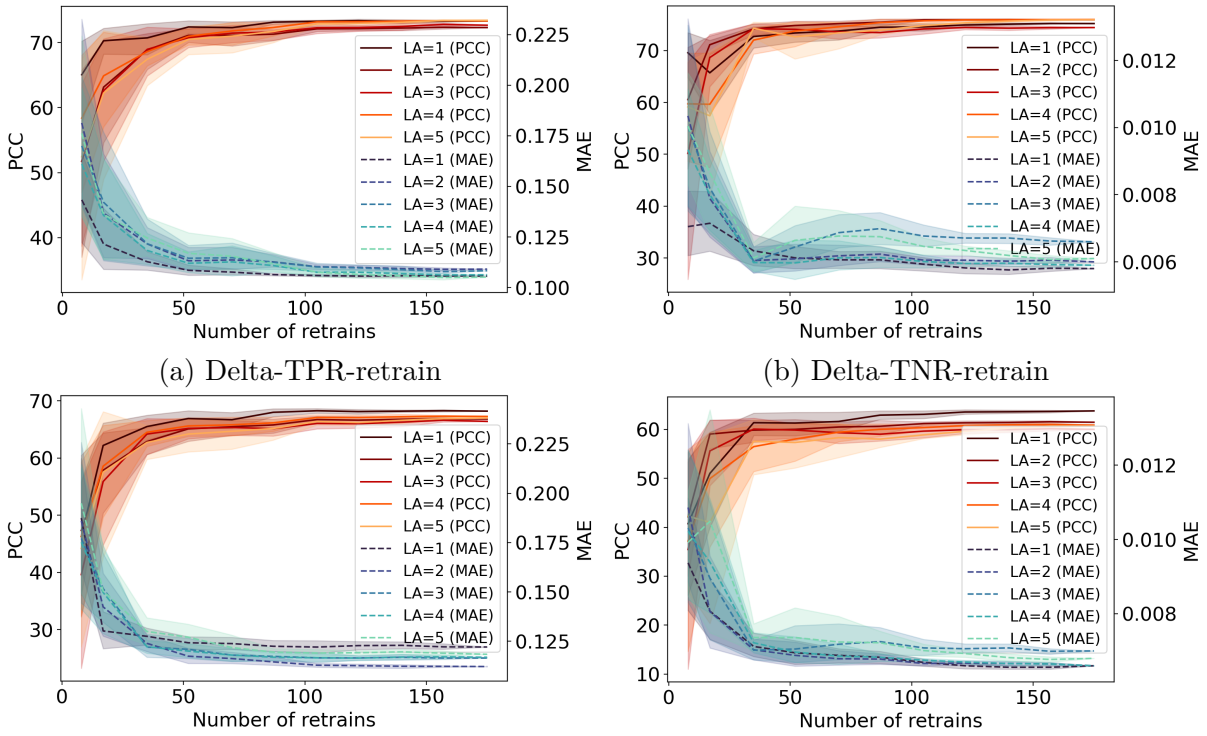
smaller AIDs. In other words, the cost benefits enabled by Ripple (see Figure 6.1a) can be achieved by performing just one third of the ML model retrains used to build the AIDs considered previously in this study.

6.3 Self-Adaptive Machine Translation Systems

To further demonstrate the applicability and generality of the proposed framework, we use it to engineer a self-adaptive machine translation (MT) system. Specifically, consider an MT system tasked with translating news from a popular news-source, *Znn.com* [44, 159], from a *source* language into a *target* language.

As time passes and new events occur, the distribution of data that the model needs to serve at inference time might differ from the distribution of data used to train it. Because of this difference between inference and training time data the MT model might have a decline in translation quality while it tries to cope with new terms, morphological constructions, or different words meanings, among other linguistic phenomena [158]. Further, the popularity of different news domains (e.g., sports, politics) also varies over time: for instance, if an important election is coming up, the majority of news will likely be about politics; yet, if the Olympics are about to start, there might be a surge of sports news. When the news domain changes, the MT model may require adaptation such that higher translation quality for the emerging domain is achieved. This is represented in Figure 6.10, which shows that there is a football game scheduled for Sunday and a political debate scheduled for Thursday. Thus, an increase in sports news would be expected on Monday, while an increase in political news would be expected on Friday. In this scenario, it would be desirable to have an MT proficient in sports news on Monday, and then adapt it to political news to provide the best translations by Thursday,

The framework evaluates the need for adaptation based on the expected benefits of fine-tuning the MT model and the cost of doing so, such that the utility of the news website is maximized. Specifically, we focus on two main quality attributes: (i) variation of the translation quality upon a model fine-tuning $\Delta\mathcal{Q}$ – provided via a combination of state-of-the-art MT evaluation metrics such as COMET22 [151] and chrF [144]; and (ii)



(a) Delta-TPR-retrain

(b) Delta-TNR-retrain

(c) Delta-TPR-nop

(d) Delta-TNR-nop

Figure 6.9: Predictive quality of the *root-LA-AIPs* in terms of MAE and PCC when decreasing the number of retrains employed to create the AID. Due to space constraints, we show the plots only for *retrain*, however *nop root-LA-AIPs* showed the same trends.

costs incurred by the news website. These costs are defined in terms of the expected and actual $\Delta\mathcal{Q}$ as follows: (a) if the MT model is not fine tuned (i.e., *nop*) and the actual $\Delta\mathcal{Q}$ had the MT model been fine-tuned exceeds a pre-defined threshold T, the system incurs a missed opportunity cost $\mathcal{O}$; (b) if the MT model is fine-tuned, the adaptive action will incur a cost $\mathcal{F}$, capturing, e.g., cloud provisioning or energy costs; (c) if the MT model is fine-tuned, but its performance improvement is sub-par, i.e., below T, the system will incur a penalty $\mathcal{R}$, which embodies a "regret" cost (e.g., from the sustainability perspective) for having performed a useless fine-tuning. Based on these costs, on the set $\mathbf{M}$ of MT metrics of interest, n news domains each with its own translation quality $\mathcal{Q}_{i,m}, 1 \leq i \leq n, \forall m \in \mathbf{M}$ and importance/weight w_i, the system utility $SysU$ of the MT system is:

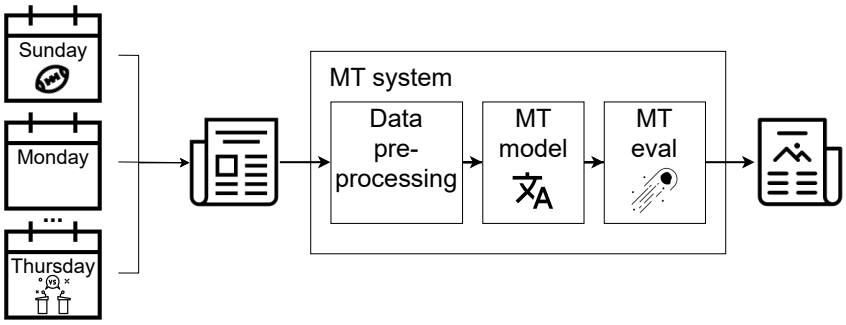


Figure 6.10: The distribution of news domains translated by the Machine Translation system varies during the week. This variability may signal the need for adaptation.

$$\textsc{Optimize}(SysU) = \textit{choose tactic than minimizes total cost}$$
$$\textsc{Optimal Tactic} = \underset{\text{tactic}\in\{\text{finetune,nop}\}}{\arg\min} \textsc{cost}(\text{tactic})$$
$$\textsc{cost}(\text{nop}) = \sum_{m=1}^{\mathbf{M}} \mathcal{O} \times \textsc{bool}((\frac{1}{n}\sum_{i=1}^{n} w_i \times \Delta\mathcal{Q}_{im}) > T)$$
$$\textsc{cost}(\text{finetune}) = \mathcal{F} + \sum_{m=1}^{\mathbf{M}} \mathcal{R} \times \textsc{bool}((\frac{1}{n}\sum_{i=1}^{n} w_i \times \Delta\mathcal{Q}_{im}) < T)$$

The goal of the self-adaptive MT system is thus to select the tactic that minimizes the sum of the overall costs incurred.

Flexico

Given the differences between this use case and the fraud detection one, and since MT models are widely used in a number of applications, we describe how the framework can be specialized to instantiate self-adaptive machine translation systems. We call this specialization Flexico and describe how each key component of the framework is modified in Flexico to account for the intrinsic characteristics of MT systems.

Since Flexico considers the *nop* and *fine-tune* adaptation tactics, and since the MT domain requires a different set of predictive features compared to the fraud detection use case, we introduce Fine-tuning Impact Predictors (FIPs), which specialize the idea of AIPs by introducing novel methodological aspects in order to be effectively employed in the context of MT systems. These include:

1. Identifying a new set of MT specific features capable of capturing dataset shifts in the input and output of MT models, while achieving an acceptable trade off between their effectiveness (i.e., predictive power) and computational costs (as this has to be incurred not only for training but also for querying the FIPs).

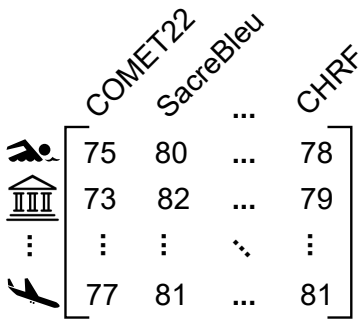


Figure 6.11: Machine Translation Quality Matrix (MTQM) – each row represents a domain that is relevant to the MT system, and each column represents a specific MT evaluation metric (e.g., COMET22, SacreBLEU, chrF).

2. For MT systems like *Znn*, that consider several domains (e.g., politics news vs sports news) with different weights, the impact of a *fine-tune* may have benefits for some domains but a detrimental impact for others. Thus, to focus on studying the predictive quality of the FIPs we decouple the problems of (i) estimating future evolutions of the input distributions to the MT model and (ii) predicting the variation in the translation quality of an MT system due to fine-tuning the MT model. As such, we consider an alternative methodology for estimating the variations of the predictive quality of the MT model under adaptation: evaluating the MT model based on a set of reference test-sets that are representative of the expected input distributions across different MT domains.

3. By leveraging the notion of domain-specific test sets, we propose two alternative FIP designs, namely FIPs trained to predict the variation of an MT model's translation quality after fine-tuning either (a) for a single domain, or (b) for any target domain. These FIP variants, as we will see, yield different trade-offs regarding generalization to unknown domains versus predictive quality on known domains.

Formally modeling MT models. In FLEXICO, we formally model the MT model via a *Machine Translation Quality Matrix* (MTQM). The MTQM is a specialization of the quality matrix described in Section 5.2 in which each column represents a domain that is relevant to the MT system, and each row represents a specific MT evaluation metric (e.g., COMET22 [151], chrF [144], SacreBLEU [145]). Figure 6.11 shows an example of this matrix, considering domains like sports, finance, and tourism.

Estimating the expected benefits of fine-tuning. Out of the three feature groups presented in Section 5.3, the basic features' one is the most re-usable, given its completely domain-agnostic nature. Yet, the remaining feature groups (input and output features) are strongly domain (and possibly application) dependent.

We propose and evaluate the use of a specialization of these two sets of features for MT models: *MT-quality* features *content-aware*. The former capture the translation perfor-

mance of the MT model, both on the current environment (recently translated sentences) as well as on fixed MT test sets.

The latter (*content-aware*) includes features inspired by work on statistical machine translation [93] that aim to characterize the data shifts of the textual content of the sentences available to the MT models, from the perspective of different language properties such as semantic similarity and lexical overlap. These features are designed to detect shifts in the distributions of two different data groups: a group representing "old data", i.e., sentences that the MT model has been fine-tuned with, and another group representing "new data", i.e., fresher sentences with reference translations that have become available since the last model update (and could be used to perform a new model fine-tuning).

More in detail, we investigate the use of the following feature sets for creating predictors aimed at estimating the expected benefits of fine-tuning MT models:

Basic features. This feature set includes: (i) count of new/old/fine-tune[3] sentences available to the MT model; (ii) ratio of new and old data; (iii) count of new/old/fine-tune source and target language words (including and excluding stop words and non-alphabetic characters); (iv) ratio between new and old word counts, for both source and target languages; (v) time elapsed since the last fine-tune.

Content-aware features. This set of features is designed with the goal of capturing environment shifts by analyzing textual information at the semantic, and lexical levels. To do so, and inspired by statistical MT, we focused on the popular concepts of n-grams [121], embeddings [27] and sentence overlaps.

- n-grams capture information about the context, word order, phrase structure, and linguistic patterns in sentences or texts. By computing the distance (via the JensenShannon [108] distance) between the n-gram distributions in the new and old datasets, we capture variations at the lexical level (e.g., if sentences started having more/less typos in the new data).
- To capture variations at the semantic level, we leveraged embeddings, which are representations of sentences into lower dimensional spaces that preserve the semantic relation between the transformed words/sentences [27]. Comparing the distributions of embeddings (via both Cosine, and Euclidean [51]) in the new and old data, should allow us to capture new domains that arise in the new data, thus signaling shifts in the environment.
- Finally, sentence overlap allows us to capture information at the lexical level, by measuring common words between the new and old data. This allows us to capture semantics information and detect new domains, given that if the overlap between new and old data is low, that likely means that the environment is changing.

MT-quality features. COMET22 [151], COMET22Kiwi [152], SacreBLEU [145], chrF [144]. These four metrics capture important textual information at the semantic similarity (COMET22 and COMET22Kiwi), lexical levels (chrF and SacreBLEU), allowing to characterize the translation quality of the MT model for both the current environment and for the fixed test sets. It should be noted that, except for COMET22Kiwi, all other

[3]We consider the *fine-tune set* as being composed of the new and old data.

features in this class assume the availability of references (i.e., human translations) for the data available for fine-tuning. Thus, including exclusively COMET22Kiwi among the features used by FIPs (or avoiding completely the use of MT-quality-aware features) allows FLEXICO to estimate whether it would be worth fine-tuning an MT model even before incurring the cost of obtaining references for the new sentences. In this case, FLEXICO enables an additional dimension of cost reduction: beyond saving the cost of useless fine-tunings, FLEXICO also reduces the costs of obtaining references for sentences that are not predicted to be useful for improving the MT model's quality.

Reference test-sets. A test set is a collection of sentences for a language pair composed of a sentence in the source language (source sentence) along with its corresponding reference translation in the target language. Each domain usually has its own test set with sentences sampled from a large pool of data of that domain. The test sets are disjoint from any data used to train or optimize models. In consider a test-set to be a reference test-set, when we keep the it the same across different steps of fine-tuning. The test set creation process is application-dependent (e.g., specialized fields such as medical, legal, technical, or conversational domains). "Fixed" test sets have been used for a long time in academic benchmarks, such as the WMT [61, 92] and IWSLT [3] shared tasks.

Recall that FLEXICO does not attempt to predict what is the expected performance of the MT system at a specific time in the future, which entails having to keep into account what type of inputs the system will be subject to in the future. Instead, FLEXICO relies on fixed test sets, which provide a fairer and more accurate comparison between the translation quality of different MT models (as all of them are evaluated on the same test set). Additionally, by using multiple fixed test sets, representative of various domains, we are able to understand what model performs best across different domains.

Fine-Tuning Impact Predictors (FIPs). FIPs are trained using an FID (Fine-tuning Impact Dataset) that is constructed similarly to an AID. We start by fine-tuning N times the target MT model and, for each fine-tuning, we measure the quality of the resulting MT model on the selected fixed test-sets using multiple quality metrics. This yields the MTQM for each fine-tuned model $M_i, i < N$. When fine-tuning model i, we fine-tune it considering all the data available up to i. This includes data with which model $i-1$ was trained as well as new data gathered since. Next, for each pair of fine-tuned models M_i, M_j, where $i < j$, we generate a new sample for the FID by computing: (i) the previously described features, assuming that the *old data* is the data used to fine-tune M_i and that the *new data* is the data used to fine-tune M_j and not used to fine-tune M_i (i.e., the additional data used in fine-tune j w.r.t. fine-tune i); (ii) the target quality variation as the difference between the MTQMs for M_j and M_i. We propose two alternative FIPs variants, which explore different trade-offs between generalization, and prediction quality:

Domain-specific FIPs. These FIPs, as the name suggests, are built to estimate the expected difference in MT performance for a specific domain (e.g., sports or finance). For instance, with this type of FIP, if our goal was to understand how fine-tuning the MT model would impact its performance in terms of COMET22 in the domain of sports, we

would have to query the FIP that had exclusively been trained with examples describing COMET22 variations evaluated based on the sport domain test set. For instance, for *Znn*, this approach requires having $\mathbf{M} \times n$ FIPs. Having more FIPs will be more expensive (more models to train), but as the FIPs are tailored to each domain, they are expected to attain higher predictive quality.

Generic FIPs. These are more ambitious predictors that, unlike the specific FIPs, are domain-agnostic and can be queried to predict the quality fluctuations of an MT model with respect to an *arbitrary* domain, including domains not seen in the training phase. Similarly to the domain-specific FIPs, we require a generic FIP for each metric $m \in \mathbf{M}$. Generic FIPs exploit the fact that both the content-aware and the MT-quality features encode information about the domain test set, which can be exploited by a FIP to learn how to adjust its predictions to the characteristics of the target test set. In fact, the content-aware features describe data shifts between the new-data set and the domain test set, and the MT-quality features quantify the performance of the MT model for the domain test set. We leverage these observations and train generic FIPs by using a training set obtained by simply merging the training sets of the various domain-specific FIPs. This way, during training, we expose the generic FIP to information about how the quality of a MT model varies across different domain test-sets, with the objective of enabling the resulting FIP to generalize its predictions to arbitrary domain test-sets

We evaluate FLEXICO with respect to the following research questions:

RQ1 How accurately can FIPs predict the benefits of fine-tuning?

RQ2 What features lead to higher FIP predictive quality and how expensive is it to train and query them?

RQ3 Can the FIPs generalize across domains?

RQ4 What utility improvements does FLEXICO yield?

Use cases

We consider two use cases accounting for different languages pairs and data types/contents. For both use cases, the MT model is evaluated on fixed test sets.

English-Chinese (En-Zh). This use case, inspired by *Znn*, uses the OpusMT model [9, 172] available through HuggingFace and LDC's 'Hong Kong News Parallel Text' [112], which contains news data from 1997 to 2000. To create the fixed test sets, we used chat-GPT [132] to assign domains, which are not present in the original dataset, to each news article as follows. To bound the output of chat-GPT, we created a list of news domains that was passed to chat-GPT: `politics`, `business & economy`, `technology`, `science`, `health & wellness`, `environment`, `entertainment`, `sports`, `culture & education`, `social issues`, `travel & tourism`, `fashion & lifestyle`, `opinion & editorial`, `law & legal issues`, `finance`, `real estate & infrastructure`. This list was created by browsing the news categories of multiple online news sites and by asking chat-GPT to provide a list of news domains. With this list, we prompted chat-GPT to give us the top

Table 6.10: Sizes (in number of news articles and sentences) of the En-Zh fixed test-sets.

News Topic	Entertainment	Environment	Finance	Governance	Health & Wellness	Sports	Travel & Tourism
#Articles	50	254	172	113	112	36	158
#Sentences	764	3295	2199	2131	1923	573	2514

3 domains (from the list) that it associated with each news article, along with a number between 0 and 100 for each domain selected, to allow us to order the domains from most to least important. We thus sent the following query to elicit news topics from chatGPT:

```
CHATGPT_ANSWER_FORMAT = (
    "1. Topic: value - Score: value"
    "2. Topic: value - Score: value"
    "3. Topic: value - Score: value"
)

prime_msg = {
    "role": "system",
    "content": (
        f"You are a helpful assistant that classifies
        news articles into topics based on this topics
        list: [TOPICS_LIST]. Your output consists of
        the top 3 topics and a number from 0 to 100
        associated with each topic to allow me to order
        the topics from most import to least important.
        Format your answer as {CHATGPT_ANSWER_FORMAT}."
    ),
}

query_msg = {
    "role":    "user",
    "content": f"Classify the following
                article: `{article}´",
}
```

After getting domains for all articles, we isolated the articles of year 2000, grouped them by the domain with the highest score and selected the domains with the most articles and that could be more different among each other to create 7 fixed test sets (Table 6.10). Note that `Governance` was not an original category and we created it by merging the following three categories: `politics`, `Law and legal issues`, and `Business and economy`.

The remaining years of the Hong Kong News dataset [112] (i.e., years 1997, 1998, and 1999) are used to (i) create the FID which is used to train the FIPs (years 1997 and 1998), and (ii) test the FIPs (year 1999). When computing the content-aware and the MT-quality features for the FID, we used a sample of 10% of the data available to compute these features. The FID was created by fine-tuning the OpusMT model 147 times, which yields 10585 FID samples.

English-French (En-Fr). For this use case, we employ the Opus datasets [173] and the Tatoeba OpusMT model [168, 171]. To create our own dataset out of the Opus dataset, we started by comparing the training data of the Tatoeba OpusMT model with each Opus dataset, to guarantee that we only selected datasets whose data had not previously been used to train the base MT model. From this analysis, we obtained 7 datasets, from domains

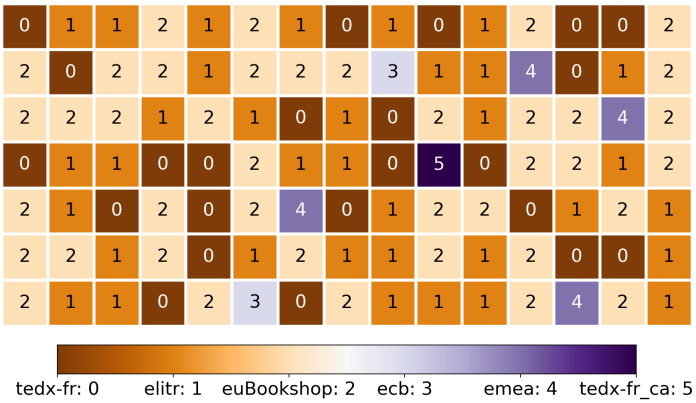


Figure 6.12: En-Fr FIP fine-tune dataset. Each cell corresponds to a chunk of 10000 sentences belonging to each of the Opus datasets used for fine-tuning the base MT model.

Table 6.11: Opus datasets used for fine-tuning and testing [173].

Dataset	**Task**	**#Sents**	**Domain**	**Source**
ELRC [59]	Test	1625	Legal	ELRC_1075_EUIPO - IP case law French-English
PHP [141]	Test	1754	Tech	Parallel corpus originally extracted from http://se.php.net/download-docs.php
Quran (Tanzil) [146]	Test	440	Religion	Collection of Quran translations
ELITR [187, 188]	Test & fine-tune	1774 | 359000	Legal	Derived from documents published by the European Court of Auditors
EU Bookshop [174]	Test & fine-tune	1022 | 405000		Corpus of documents from the EU bookshop
TedX-fr [153, 167] TedX-fr_ca [153, 167]	Test & fine-tune	1927 | 233000 1495 | 10000		Crawl of nearly 4000 TED and TED-X transcripts from July 2020

such as Religion, Legal, and Tech. Table 6.11 lists the Opus datasets that served as basis for the creation of fixed test sets for this use case. Some datasets (e.g. PhP and Quran) had very few samples so they were used only for testing purposes.

After gathering these datasets, and assigning some of them only for test purposes due to their smaller size (less than 2000 sentences), we split the remaining ones into chunks of 10000 sentences. The "left-over" of these fine-tuning sets was used to define the fixed domain test sets for MT quality evaluation. Finally, we ordered the fine-tuning data chunks randomly to form a larger dataset with which we could simulate fine-tunings and create the FID. Figure 6.12 displays the random order of the random chunks in the dataset.

Again for this use case, when computing the content-aware and the MT-quality features for the FID, we used a sample of 10% of the data available to compute these features. We use 70% of the FID for training the FIPs and 30% for evaluation. The FID was created by fine-tuning the Tatoeba OpusMT model 105 times, which yields 5356 FID samples.

Table 6.12: Performance of the generic FIPs and average performance (across domains) of the specific FIPs, for the best feature-sets for each domain.

Use Case	Model	MAE		PCC	
		Generic	Specific	Generic	Specific
En-Zh	RF	0.0049	**0.0032**	76.77	**86.44**
	XGB	0.0053	**0.0048**	76.51	**84.83**
	Lin	0.0066	**0.0054**	68.3	**89.07**
En-Fr	RF	**0.0013**	0.0021	30.09	**65.63**
	XGB	**0.0012**	0.0016	33.72	**62.33**
	Lin	0.0018	**0.0016**	25.11	**70.49**

FIP Prediction Quality (RQ1)

To evaluate the prediction quality of the FIPs we conducted a study varying three key aspects of the FIP building process: (i) three different model types – XGBoost trees [38] (XGB), and Random Forest (RF) and Linear (Lin) regressors from the SKLearn Library [140]; (ii) three sets of features – basic, content-aware, all; (iii) two versions of the target – domain-specific or generic. The quality of the resulting predictions is evaluated with mean absolute error (MAE) and pearson correlation coefficient (PCC).

Let us start by comparing, in Table 6.12, the prediction quality of the generic and domain-specific FIPs for COMET22, trained with the best performing sub-set of features among the ones presented earlier (this set can be discovered, e.g., using a validation set). We report the MAE and PCC across all domains, for both use cases and considering the three model types mentioned above for the FIPs. The use case for which both FIP variants achieve the highest PCC is En-Zh, where the S-FIPs and G-FIPs achieve an average PCC of 86.44 and 76.77 for the best performing modeling approach (Lin and RF, respectively). The En-Fr is more challenging for the FIPs, since it encompasses data originated from a much wider set of domains and have been with trained with approximately half of the data available for the En-Zh use case. In fact, with the En-Fr use case, the PCC of the S-FIPs and G-FIPs drops to 70.49 (Lin) and 30.09 (XGB), respectively.

Overall, in both use cases, the G-FIPs achieve lower performance than the S-FIPs, both in terms of MAE and PCC. The superiority of S-FIPs is not surprising, as being queried solely on the domain on which they were trained inherently favors them. The performance gap between the two FIPs variants is not very large in the En-Zh use case (the PCC of the best G-FIPs is 76.77%), but it is quite large in the En-Fr use case (where the he PCC of the best G-FIPs drops to 30.09%). This can be explained by recalling that the En-Fr use case spans a broader range of domains, creating more challenging conditions for domain-agnostic predictors like the G-FIPs.

FIP Feature Importance and Cost (RQ2)

We now focus on the S-FIPs that achieved higher predictive performance than the generic FIPs for both use cases. Specifically, we analyze how different combinations of feature sets impact the FIPs predictive performance.

For this study, we consider: (i) the 3 MT feature sets presented earlier (i.e., Basic, Content aware, MT-quality aware); (ii) all 3 feature sets together (i.e., All); (iii) combinations of the 3 feature sets (Basic + Content aware, Basic + MT-quality, Content aware + MT-quality); (iv) combinations of the COMET22Kiwi metric with the Basic, Content aware, and both to evaluate the expected quality should no reference sentences be available; (v) a version of the content aware features that does not account for the n-grams (content aware-no-ngrams). As we will show later, this is motivated by the high computational cost of computing n-gram features compared to the remaining content aware features.

On the En-Zh use case the content-aware-no-ngrams feature set (i.e., sentence overlaps and embedding features) is the best performing across the domains considered. On the En-Fr use case (cf. appendix) we see much stronger variations, with different feature sets having substantially different performances across domains. On average, the best performing feature set is Basic + MTQual, followed by content-aware + MTQual.

Additionally, looking at the feature set combinations that use MT quality metrics[4], we see that for En-Zh these features are typically not needed: only 2 out of the 7 domains achieve the highest PCC with a feature set using MT-quality aware features (Governance and Travel and Tourism). Differently, for the En-Fr use case, only 1 out of 7 domains (Elitr) does not require MT-quality aware features to achieve the highest predictive performance.

This suggests that for scenarios in which the domains are less heterogeneous, MT-quality aware features are less important to the S-FIPs and that S-FIPs can be used, in these scenarios, to determine whether to ask human annotators for references. Ultimately, the set of best features for the FIPs is dataset/domain dependent and can be discovered via classical feature selection techniques

Feature computation cost. Since the features employed by the FIPs impact how much it costs to query them (each time a FIP is queried, its input features need to be computed), we investigated how much it costs to compute each feature-set. The results, which we include in the appendix, represent upper-bounds on the actual feature computation times as the feature computation process was not fully optimized and other users could concurrently access the machines, possibly introducing noise in the measurements.

Specifically, for the En-Zh S-FIPs which achieve the highest predictive performance with the content aware-no-ngrams feature set, this represents cost savings of around 80× w.r.t. the cost of fine-tuning the MT model. We see that the trends for the En-Fr use case are similar, albeit smaller, allowing for cost savings of around 40× when considering the Basic + MTQual feature set and comparing to the fine-tune cost.

Note that due to resource limitations we considered relatively small MT models. Since the cost of feature computation is independent from the MT model, unlike the fine-tuning cost that is strongly dependent on the MT model's size, for largerMT models we expect to see even larger cost savings.

[4]Recall that these features, except for COMET22Kiwi, require reference translations, which are obtained by paying human annotators.

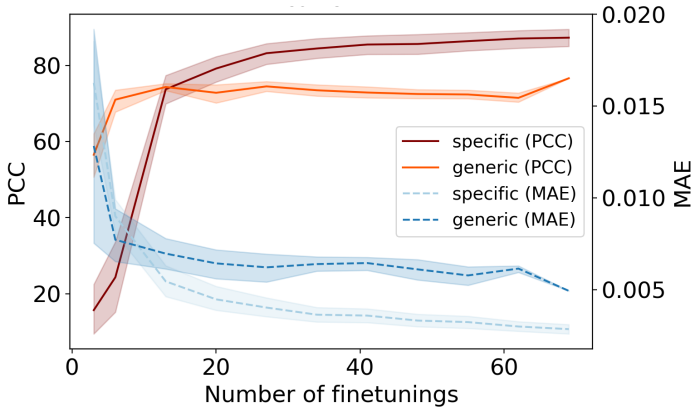


Figure 6.13: MAE and PCC for FIPs with RF on En-Zh and COMET22, varying the number of fine-tunings to create the FID.

Computational cost of building the FID. Finally, we analyze the cost of creating the FID. The FID was obtained by fine-tuning the MT models 147 and 105 times (69 and 57 fine-tunings for training the FIPs, En-Zh and En-Fr, resp.). Despite the methodology for FID generation produces $O(n^2)$ samples out of n fine-tunings, fine-tunings can be costly and the cost of feature computation also adds up. To understand whether we can reduce the cost associated with creating the FIDs, we explore how much data is actually needed to train accurate FIPs, by conducting a sensitivity study analyzing the FIP's prediction quality (measured via PCC and MAE) as we vary the size of the G-FIPs's training set.

We fixed the G-FIP feature set to *Basic and System Performance features* (i.e., the feature combination that led to the best performing G-FIPs) and tested with the random-forest model. We repeated each sample size 10 times for reliability. Figure 6.13 displays the results, showing that decreasing the number of fine-tunings performed to 20% ($\approx$ 13) of the original number of fine-tunings in the train-set (69) still guarantees high performance.

FIP Domain Generalization (RQ3)

To evaluate the domain generalization capability of the G-FIPs, we conduct a leave-one-out cross-validation study by leaving each domain test set out of the G-FIPs' train-set and then evaluating the predictive performance of the FIP on the test set that was left out. Table 6.14 shows the results obtained for the specific case of random forest FIPs for the En-Zh use case. We focus this study on the En-Zh use case since we have already established (cf. RQ1) that the G-FIPs achieve poor performance in the En-Fr use case, due to its challenging characteristics (and that S-FIPs are recommended in such scenarios). We still report the full results in the appendix.

Overall, we observe very high PCC values (above 85% and up to 95%) for all domains except Entertainment, where the G-FIPs achieve $\approx$67% PCC. The best G-FIPs, similarly to the S-FIPs (Table 6.13), also do not use feature sets with MT quality metrics. These results confirm that in the En-Zh use case the G-FIPs are very accurate when challenged with out of distribution queries regarding domains not seen in training.

Table 6.13: Performance — pearson correlation coefficient (PCC) and mean absolute error (MAE) — of the S-FIPs when predicting COMET22 with RF models on En-Zh.

Metric	Test set / Feature Set	Entertainment	Environment	Finance	Governance	Health & Wellness	Sports	Travel & Tourism
PCC	All	61.14	92.08	88.82	95.62	90.63	76.31	84.75
	All + Kiwi	61.73	91.74	88.61	95.55	90.73	75.30	84.51
	Basic	61.52	91.88	91.20	95.15	90.61	74.98	85.08
	Basic + ContAware	60.64	91.50	**91.29**	95.06	90.80	74.41	84.21
	Basic + Kiwi	62.65	92.21	86.75	95.89	89.91	76.11	85.60
	Basic + MTQual	61.47	92.21	87.11	**95.97**	89.49	76.63	**85.66**
	ContAware	48.79	80.02	89.26	93.72	89.37	69.02	80.31
	ContAware-no-ngrams	**64.65**	**95.66**	89.92	95.00	**91.94**	**79.90**	85.15
	ContAware + Kiwi	51.67	78.82	87.87	93.93	87.84	70.12	80.34
	ContAware + MTQual	50.50	79.84	87.59	93.75	89.16	69.07	80.85
	MTQual	-2.18	13.26	-33.36	31.43	-5.80	-7.05	-19.26
MAE	All	0.0098	0.0110	0.0078	0.0025	0.0021	0.0046	**0.0033**
	All + Kiwi	0.0102	0.0115	0.0082	0.0026	0.0022	0.0052	**0.0033**
	Basic	0.0107	0.0096	0.0065	0.0028	0.0022	0.0042	0.0034
	Basic + ContAware	0.0104	0.0096	0.0064	0.0026	0.0019	0.0052	0.0034
	Basic + Kiwi	0.0105	0.0114	0.0082	0.0025	0.0025	0.0040	**0.0033**
	Basic + MTQual	0.0100	0.0109	0.0076	0.0023	0.0023	0.0036	0.0035
	ContAware	0.0066	0.0054	0.0106	0.0026	0.0018	0.0051	0.0039
	ContAware-no-ngrams	**0.0032**	**0.0040**	**0.0018**	**0.0015**	**0.0011**	**0.0020**	0.0040
	ContAware + Kiwi	0.0064	0.0061	0.0120	0.0027	0.0022	0.0048	0.0037
	ContAware + MTQual	0.0061	0.0052	0.0118	0.0028	0.002	0.0044	0.0038
	MTQual	**0.0026**	**0.0025**	0.0044	0.0022	0.0024	0.0048	0.0064

MT System Utility Improvement (RQ4)

To evaluate the improvement in system utility attained by FLEXICO, we used it to create a self-adaptive version of the *Znn* use case described earlier. Specifically, the set **M** of MT metrics of interest is given by COMET22 [151] and chrF [144]. We consider fine-tune costs of [1, 5, 10] and thresholds for the delta improvement (i.e., minimum improvement that we want to achieve with a fine-tune) from 0.1 to 1.0 with intervals of 0.1. The missed opportunity cost is set to $2 \times$ finetune cost $\times$ FIP_pred and the incorrect fine-tune penalty to $2 \times$ finetune cost $\times$ ($\Delta\_threshold -$ FIP_pred).

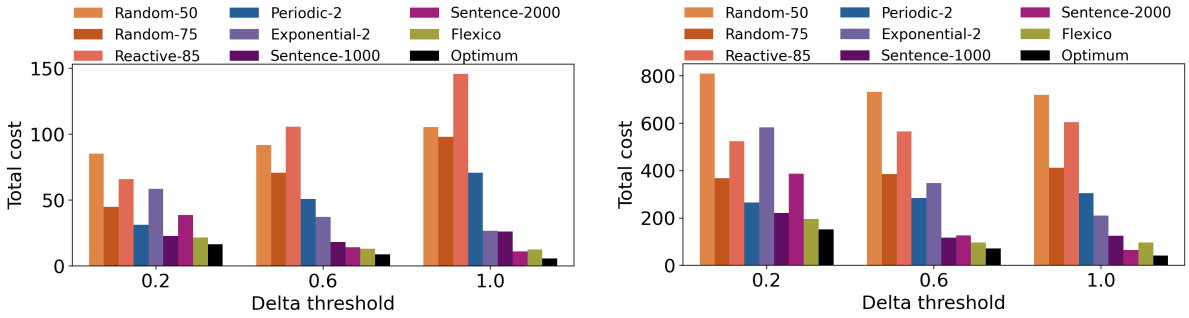
Baselines. FLEXICO uses the generic En-Zh FIPs with all the features, and we compare it against the following baselines:

- **Periodic-*n*:** fine-tune the model at every *n-th* time step. We set n=2;
- **Exponential-*n*:** fine-tune the model with an exponentially increasing period of base n. We set n=2;
- **Random:** fine-tune at each time step with 50-50 or 75-25 probability;
- **Sentence:** fine-tune whenever at least 1000 or 2000 new sentences are available;
- **Reactive-85:** fine-tune if any target MT metric is below 85;
- **Optimum:** an ideal oracle that has perfect knowledge up to 5 steps into the future, and thus knows exactly the actual benefits of fine-tuning the MT model;

Figure 6.14 compares the utility (i.e., total cost) attained by each baseline for each fine-tune cost tested and for a sub-set of the delta thresholds. Results for all delta thresholds can be consulted in the appendix. The results demonstrate that FLEXICO improves system utility over the naive baselines, getting closer to the optimum oracle. The results show that some of the baselines can achieve performances close to FLEXICO for some specific

Table 6.14: Leave-one-out cross validation performance — pearson correlation coefficient (PCC) and mean absolute error (MAE) — of the G-FIPs with RF models on En-Zh.

Metric	Test set / Feature Set	Entertainment	Environment	Finance	Governance	Health & Wellness	Sports	Travel & Tourism
PCC	All	57.25	89.07	87.95	92.87	90.62	85.88	77.55
	All + Kiwi	60.58	85.48	86.11	92.38	91.26	85.53	82.73
	Basic	61.40	88.50	86.11	92.55	90.07	86.56	84.28
	Basic+ ContAware	60.47	89.66	88.98	91.96	88.74	84.72	75.46
	Basic + Kiwi	60.08	84.95	84.47	92.7	91.29	86.13	83.45
	Basic + MTQual	59.66	85.65	80.97	94.00	90.70	85.86	83.88
	ContAware	66.98	93.86	94.45	76.14	87.16	86.70	**86.01**
	ContAware-no-ngrams	64.89	**95.74**	94.04	**94.44**	**92.16**	86.21	85.76
	ContAware + Kiwi	**68.79**	89.32	90.52	90.69	91.36	**87.61**	85.1
	ContAware + MTQual	66.59	93.23	**94.59**	64.34	91.38	85.92	84.73
	MTQual	13.62	18.25	16.48	12.5	21.6	6.73	10.89
MAE	All	0.0065	0.0038	0.0033	0.0076	0.0056	0.0051	0.0040
	All + Kiwi	0.0052	0.0059	0.0025	0.0046	0.0083	0.007	0.0049
	Basic	0.0062	0.0044	0.0040	0.0070	0.0075	0.0056	**0.0031**
	Basic+ ContAware	0.0063	0.0042	0.0040	0.0088	0.0087	0.0051	0.0039
	Basic + Kiwi	0.0054	0.006	0.0026	0.0045	0.0082	0.0072	0.005
	Basic + MTQual	0.0070	0.0029	0.0026	0.0045	0.0061	0.0074	0.0060
	ContAware	0.0049	0.0026	0.0024	0.0032	0.0020	**0.0015**	0.0043
	ContAware-no-ngrams	**0.0020**	**0.0016**	**0.0009**	0.0037	0.0041	0.0016	0.0063
	ContAware + Kiwi	0.0034	0.0041	0.0016	**0.0013**	0.004	0.0018	0.0045
	ContAware + MTQual	0.0036	0.0027	0.0023	0.0039	**0.0019**	0.0016	0.0038
	MTQual	0.0039	0.0024	0.0026	0.0024	0.0028	0.0038	0.0062



(a) Fine-tune cost 1.

(b) Fine-tune cost 10.

Figure 6.14: Total cost incurred by each baseline as a function of: the fine-tune cost, the target delta improvement for MT metrics.

settings (e.g., sentence-2000 for the largest delta threshold of 1.0). However, none of the baselines achieves robust performances across all delta thresholds (e.g., sentence-2000 for a delta threshold of 0.2 and fine-tune cost of 1, yields 80% higher total cost than FLEXICO).

6.4 Summary

This chapter evaluated this thesis via two use cases (credit card fraud detection and machine translation systems), providing evidence for the claims presented and discussed in Section 6.1. Sections 6.2.1, 6.2.4, and 6.3 demonstrated that overall system utility can be successfully optimized by leveraging the proposed framework to engineer self-adaptive ML-based systems (claim 1), both for the short-term (Sections 6.2.1 and 6.3) and for the long-term (Section 6.2.4). Additionally, the latency of the process for generating adaptation strategies proved to be suitable for online planning in non-critical scenarios (claim 2),

regardless of whether we are planning for the short-term or for the long-term. Finally, by successfully enabling self-adaptation for both use cases, we demonstrate that the framework is generalizable to these two use cases and can be applicable to a range of execution scenarios with different characteristics and types of ML models, thus validating claim 3.

Chapter 7

Discussion and Future Work

This chapter starts by discussing how users and practitioners should proceed when adopting the framework and instantiating it for their ML-enabled systems in Section 7.1. Then, in Section 7.2, we go over the assumptions and limitations of the framework proposed in this thesis. Section 7.3 describes promising directions for future work that further validate and expand the proposed framework. Finally, Section 7.4 concludes this thesis.

7.1 Framework Adoption

As demonstrated in the previous sections, the framework proposed in this thesis allows practitioners to engineer self-adaptive ML-based systems as a way to optimize the utility of the ML-based system. However, for practitioners to actually benefit from the use of the framework, they need to be able to apply it to their use cases. This requires understanding the key steps that should be followed and components that ought to be tailored when applying the framework to new use cases. These are described in this section.

For practitioners wanting to engineer self-adaptive ML-enabled systems, the following elements must be tailored to their specific application needs: (i) system utility definition – this is intrinsically system dependent, as some systems may be more concerned with latency and throughput, whereas for others user-experience may be the most important quality attribute; (ii) formal model of the system – although some components of the formal model are generic across all implementations, such as the abstraction over the ML models via the *quality matrix*, the representation of the system's environment of operation, and the adaptation manager with the repertoire of tactics for adaptation (e.g., *retrain* and *fine-tune*), each system may have specific components that are critical for the system's overall utility and that should be accounted for when reasoning about ML adaptation.

Regarding the AIP features, we argue that the *basic* feature set (see Section 5.3) is generic across ML systems and that the *content-aware* feature set is generic across MT systems and can be re-used when implementing Flexico for tasks in the natural language processing domain. The remaining features, such as the *MT-quality-aware* features and the fraud rate, which are specific to the MT and fraud detection domains, respectively, can be

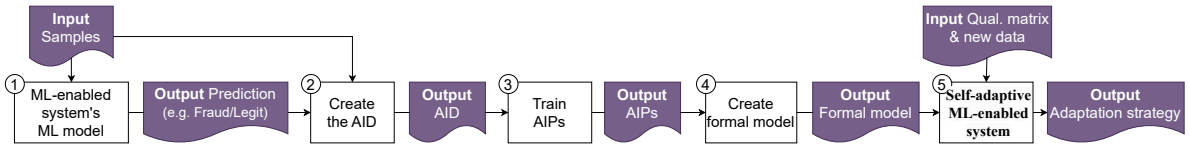


Figure 7.1: Framework adoption pipeline.

replaced with performance features specific to the task/domain at hand. This step requires domain knowledge, and/or experimentation for creating new domain-specific features.

Overall, and as shown in Figure 7.1, to use the framework to engineer self-adaptive ML-based systems for new applications, it is necessary to (i) create an AID by following the proposed methodology (Section 5.3), (ii) instantiate AIPs for the selected adaptation tactics and metrics of interest, leveraging the AID that was built by observing the impact of the considered adaptation tactics on the ML model's predictive quality, (iii) create a formal model of the system, capturing the evolution of the state of the ML component that will be adapted as described in Section 5.2. Note that the proposed strategy to formally model ML components does not constrain the applicability of the framework to specific types of ML models (e.g., neural networks, random forests).

7.2 Assumptions and Limitations

The goal of the work developed in this thesis was to create a general framework to decide *when and how* to adapt ML-based systems. Despite the successful creation of two self-adaptive ML-based systems, limitations still exist. This section discusses these limitations, in particular the assumption of ground-truth label availability (Section 7.2.1). Then, we present threats to the generalizability of the results (Section 7.2.2), followed by a discussion on the assumption of environmental state independence from the tactics executed (Section 7.2.3), and on how we account for uncertainty (Section 7.2.4). Finally, we analyze the extension of the framework to account for multiple adaptation tactics (Section 7.2.5).

7.2.1 Ground-truth Label Availability Assumption

We assume, in both use-cases, the availability of ground truth labels for an immediate evaluation of the ML model's predictive performance. In reality, ground truth labels may not be immediately available for the system's performance to be evaluated. This is often the case in the fraud detection domain, since transaction labels may take months (if ever) to become available [111, 169]; Similarly, in the machine translation domain, standard metrics for evaluating the quality of machine translations (such as Comet22 [151], SacreBLEU [145], and chrF [144]) are subject to the availability of golden reference sentences (i.e., human translated sentences) which are not always available and are expensive to obtain.

However, the problem of real-time performance estimation with unlabeled samples is orthogonal to the problem addressed in this work. Nonetheless, we attempted to mitigate this threat by showing how to leverage state-of-the-art ML algorithms to address the issue

of ground-truth label availability when adapting ML-enabled systems, by proposing CB-ATC (Section 5.4), an extension of ATC [67] (a state-of-the-art algorithm), which we empirically evaluate with the fraud-detection system use-case (Section 6.2.2).

7.2.2 Threats to Validity

External validity. The findings regarding the predictability of the impacts of *retrain* and *fine-tune* tactics are dataset and domain- dependent and so they cannot be generalized to other domains or datasets beyond those evaluated in this thesis. This also applies to the time complexity of the approach, which depends on the complexity of the formal model.

We mitigate the general applicability threat by instantiating the proposed framework for two use-cases with different characteristics (classification-based fraud detection ML model and sequence-to-sequence MT model). Further, the MT use-case is tested with two datasets representing two different language pairs (one of a widely used European language pair, and one of high importance due to the number of speakers) and resorting to two different MT models. We empirically verified that the improvements in overall system utility were consistent across use cases, which supports the claim that the framework can be applied to multiple domains with different characteristics and ML models.

Nonetheless, evaluating the framework with additional domains (e.g., intrusion and spam detection) and with systems that rely on other types of ML models (e.g., NNs, support vector machines, linear models) will further strengthen the generalizability claim.

Analogous considerations apply to the evaluation of CB-ATC, which has only been conducted in the context of the fraud detection use case. Clearly, it would be desirable to evaluate this method on a broader set of datasets to verify whether the benefits observed in our study generalize to different domains.

Internal validity. The conclusions regarding the improvements to system utility are intrinsically dependent on the definition of system utility and on the evaluated execution contexts (regarding tactic cost and latency, system SLAs, MT delta threshold, MT metrics, underlying AIP model and features). For this reason, in both use-cases, we evaluate: (i) the performance of the framework under different contexts, and (ii) the predictive performance of the AIPs with different feature sets and modeling approaches.

Regarding tactic latency, we test the framework under different assumptions for adaptation latency. For the fraud-detection use-case, we start by assuming that *retrain* latency is lower than one time interval. This assumption holds for the considered use-case since the ML components employed by the fraud detection system are relatively simple. Nonetheless, when testing Ripple, the long-sighted version of the framework, we relax the *retrain* latency assumption and instead assume that a tactic's latency is lower than or equal to the look-ahead horizon considered. Without this assumption, the LA-AIPs revert to the myopic setting and can no longer gauge the benefits of adapting because these would only be available further into the future than what is actually analyzed. For the MT use-case *fine-tune* latency was not accounted for, since the focus of this use case was on understanding whether the impact of fine-tuning MT models could be estimated, as a first step towards creating self-adaptive MT systems.

However, as demonstrated by the fraud-detection use-case and by existing work in the self-adaptive systems literature [125, 126], accounting for tactic latency may affect the selected adaptation strategy, since tactics may be pro-actively enacted such that their benefits can be collected when they are actually needed. Studying the impact of *fine-tune* latency is an interesting direction for future work as accounting for latency may bring even further benefits, especially in the context of real-time MT applications.

Finally, we consider time to be discretized into application-dependent fixed time intervals. We believe this assumption does not hamper the applicability of the framework, since (i) it is natural to wait non-negligible time intervals to both monitor the evolution of the environment and the impact of the adaptation tactics; and (ii) this wait is necessary to collect new data to enable future executions of the *retrain* and *fine-tune* tactics.

7.2.3 Environment State Independence Assumption

We assume that the adaptation tactics that are executed do not influence the future state of the environment nor the behavior of external (environmental) agents (e.g., fraudsters). This simplifying assumption allows us to focus on estimating the expected benefits of adapting ML models via *retrain* or *fine-tune* tactics, which is the focus of the work developed in this thesis. We believe this to be a reasonable assumption since it is common for external agents to be unaware of system updates, particularly when such updates do not require the agent's authorization to be performed. In such cases, there is no reason to believe that the agent's behavior would be altered. However, it is also possible to consider scenarios where (i) system updates require an agent's authorization (e.g., when a software update requires a user's permission to execute) or (ii) an adversarial agent is monitoring the system and is aware of when it is adapted. In such scenarios it is reasonable to assume that the agent's behavior might change in response to the adaptations. This assumption can be relaxed in future work, for instance by taking a game-theoretic approach to modeling the interactions of the system and the environment [28, 91, 107].

7.2.4 Accounting for Uncertainty

Recent work [25, 80, 127] has shown that capturing uncertainty and including it when reasoning about adaptation contributes to improved decision making. In our work, we capture uncertainty (in the environment and in AIP predictions) via the Extended Pearson-Tukey (EP-T) three-point approximation, used in the self-adaptive systems' literature to deal with uncertain predictions [124]: we get the predictions for the 5th, 50th and 95th percentiles and create three paths in the formal model (for each percentile) with different probabilities. However, since our goals were to predict the benefits of the *retrain*, *fine-tune*, and *nop* adaptation tactics, and to understand the impact of an adaptation on overall system utility, we did not simultaneously account for uncertainty in both the environment and in the AIPs. We also do not test how uncertainty propagation across the ML-enabled system may affect the adaptation decision nor the impact of adaptation on overall system utility. This is a promising research direction which we elaborate in the Section 7.3.1.

7.2.5 Extension to Multiple Tactics

We have constructed and evaluated AIPs to predict the impact of executing tactics *nop*, *retrain*, and *fine-tune*, with each use-case considering only two of these tactics. However, the framework was designed to support other tactics, such as the ones described in Section 4.1. Thus, to demonstrate how the framework can be extended and cope with these additional tactics even in the absence of tactic specific AIPs, we performed a "what-if" analysis with the planner component (i.e., the formal model) and with the *component replacement* tactic (cf. Section 6.2.3). This analysis demonstrated how the planner can be extended to account for multiple tactics and can effectively analyze the cost/benefits trade-offs of the different tactics to generate optimal adaptation strategies. To further validate the framework's ability to account for multiple tactics, it would be necessary to create AIPs for the additional tactics. This is an interesting and promising direction for future work, which we explore in more detail in the next section.

7.3 Open Research Questions & Future Work

In this section we discuss promising directions for future work, that further validate and extend the proposed approach.

7.3.1 Reasoning About Uncertainty Propagation

One promising research direction is studying how uncertainty and errors propagate in a system and how they affect an ML model and ML adaptation. Indeed, uncertainty and errors may originate not from the ML component, but instead from components that are upstream relatively to the ML component, thus potentially inducing the ML model to mispredict. Alternatively, it is also possible to have uncertainty and errors in the planning process impacting downstream components, and affecting system utility in unexpected ways. An interesting research direction would be to explore the role of uncertainty in adaptation of ML-based systems. Specifically, since it has been shown that accounting for uncertainty when adapting non-ML systems leads to improved system utility, it would be interesting to understand (i) whether the proposed framework can be extended with state-of-the-art ML uncertainty quantification methods [1, 69, 119] and (ii) if these methods contribute to the generation of improved adaptation plans, further optimizing system utility. Then, in a second stage, one could explore incorporating uncertainty reduction approaches [131, 182] for ML models in the framework to further improve decision making.

7.3.2 Coupling Self-Adaption of ML and Non-ML Components

As mentioned previously, uncertainty and errors may not be due to the ML component but instead to other (possibly non-ML) components that the system relies on. Thus, in such situations we do not expect ML adaptation to contribute to improving overall system utility, since the ML component is not the one at fault. In such situations, it would be ideal to have at our disposal fault localization mechanisms that allow to flag the misbehaving

component such that we can trigger its adaptation. Since non-ML components might be the ones negatively impacting system utility, it would be interesting to explore how easily the framework proposed in this thesis could be coupled with existing frameworks for self-adaptation of non-ML-based systems and with fault localization mechanisms to trigger adaptation of the correct component.

7.3.3 Collaborative AID construction.

The methodology for AID generation produces $O(n^2)$ samples out of n repetitions of a specific adaptation tactic (e.g., *retrain* or *fine-tune*). However, tactic executions can be costly (not only in monetary terms, but also from computational and sustainability perspectives) and the cost of feature computation also adds up. For instance, in typical production pipelines, it is common practice to periodically update ML models and deploy them only if the updated version improves over the quality of the current model version. Although this allows the AID to be gathered at no additional costs during this initial deployment phase, before instantiating the AIPs and enabling self-adaptation, there are still adaptations that could have been avoided (e.g., model retraining may generate a model that is worse than the one currently in production and that ends up being discarded).

An interesting alternative approach to creating the AID is the idea of *Adaptation Cards*: extending the concept of Model Cards [123] to contain information regarding a model adaptation (e.g., retrain, fine-tune). *Adaptation Cards* are meant to be publicly available in repositories such as GitHub [73] or HuggingFace [83] and to contain information regarding the impact of adapting a model with diverse sets of data. For instance, an *Adaptation Card* characterizing a model fine-tune would make available (i) a pointer to the target MT model, (ii) (a subset of) the features introduced in Section 5.3 providing information on the statistical characteristics of the data set used for fine-tuning the model, (iii) quality of the MT model before and after fine-tuning evaluated using a standardized set of MT evaluation metrics and test sets. This approach should not require users to make the data used for fine-tuning publicly available, but only the resulting features, thus limiting potential privacy concerns.

Ideally, this would allow practitioners to re-use the data collected and compute power spent by other practitioners to improve their ML-based systems or to create an AID for their self-adaptive ML-enabled system. Promoting such data re-use is a step towards greener AI as the environmental cost of these expensive model updates/tests can be diluted across multiple usages of the data.

7.4 Conclusion

With the prevalence of machine learning (ML) systems, and the surge of highly impactful ML-based systems such as large language models [134] (e.g., ChatGPT), detecting when these systems are misbehaving and adapting them to ensure their system utility is optimized becomes a first-order goal. The work developed in this thesis takes a first step at tackling this problem by proposing a repertoire of tactics for adapting ML components

of ML-enabled systems and developing a general framework for engineering self-adaptive ML-enabled systems.

The literature on self-adaptive system has mostly targeted non-ML systems, or leveraged ML as part (but not as target) of the adaptation process, for instance leveraging ML to improve a planner's capabilities and explore the space of adaptation tactics more efficiently. In contrast, we focus on the problem of determining *when and how* to adapt an ML-based system to optimize system utility. This raises non-trivial challenges, such as (i) estimating the expected costs and benefits that an adaptation tactic will have and (ii) reasoning about how the expected benefits will impact overall system utility to understand whether adaptation is worth it.

To address the first challenge, we start by eliciting a repertoire of tactics for ML adaptation from state-of-the-art work on ML. Chapter 4 presents the repertoire, along with a discussion on the advantages and disadvantages of each tactic and examples of when to apply them. Then, we focus on two of these tactics, particularly ML model retrain and fine-tune and create predictors to estimate the expected benefits of executing them.

The second challenge is tackled by leveraging probabilistic model checkers to plan the adaptation strategy to execute. This requires creating formal models of ML-enabled systems and finding an adequate level of abstraction to represent ML components, ensuring not only that their characteristic behaviors are modeled, but also that the formal abstraction is expressive, general, accurate, and that the model verification is tractable for usage in online adaptation of systems.

Thus, in Chapter 5 we present a framework to reason, in a principled way, about the cost/benefit trade-offs associated with adapting ML components of ML-based systems. The framework relies on adaptation impact predictors (AIPs) to estimate the expected benefits of executing different adaptation tactics and on probabilistic model checking to navigate the trade-offs of executing one tactic over another and generate optimal adaptation strategies. The model checker is particularly valuable when considering longer planning horizons and when tactics have a non-negligible execution latency.

The framework was successfully applied to engineer two self-adaptive systems (Chapter 6): a credit card fraud detection system and a machine translation system. In both cases, the framework allowed to achieve higher system utility when compared against simpler baselines that periodically or randomly adapt the ML model. Further, we empirically verified that the execution time of the adaptation strategy generation is suitable to plan how to adapt a system in an online fashion. Finally, we demonstrate how to apply the framework to plan for the long term and how that leads to further improvements to system utility when compared against short-sighted approaches.

Appendix A

This appendix provides additional details on FLEXICO's evaluation and on the experimental results obtained. Specifically:

Section A.1. Provides details about the feature computation cost for querying the FIPs and the cost savings that FLEXICO leads to;

Section A.2. Provides all the results for the specific FIPs of both use-cases (En-Zh and En-Fr) on the 4 MT metrics (COMET22, chrF, SacreBLEU, COMET22Kiwi), 3 models (random forest, linear, xgboost trees) and FIP feature sets tested;

Section A.3. Contains the results for the generic FIPs of both use-cases (En-Zh and En-Fr) on the 4 MT metrics (COMET22, chrF, SacreBLEU, COMET22Kiwi), 3 models (random forest, linear, xgboost trees) and FIP feature sets tested;

Section A.4. Has the results for the leave-one-out cross-validation evaluation of the generic FIPs of both use-cases (En-Zh and En-Fr) on the 4 MT metrics (COMET22, chrF, SacreBLEU, COMET22Kiwi), 3 models (random forest, linear, xgboost trees) and FIP feature sets tested;

Section A.5. Compares the system utility improvements obtained by FLEXICO with those obtained by simpler baselines.

A.1 Feature Computation Cost

Table A.1 displays the cost (in USD) of computing each feature-set to query the FIPs. We see that using FLEXICO to prevent useless fine-tunings can save up to 80× the cost of a fine-tuning for the En-Zh use-case and up to 40× for the En-Fr use-case.

A.2 Specific AIPs

The experimental results obtained for the specific FIPs for the En-Zh use case are displayed in Tables A.2, A.4 (for MAE) and in Tables A.3, A.5 (for PCC).

Tables A.6, A.8 and Tables A.7, A.9 contain the experimental results obtained for En-Fr use-case, for MAE and PCC, respectively.

Table A.1: Cost (in USD) of feature set computation for a single FID sample.

Feature set	**Cost-En-Zh**	**Cost-En-Fr**
All	7.49e-03	5.06e-02
Basic	6.46e-05	1.82e-04
ContAware	4.82e-03	3.69e-02
MTQual	2.61e-03	1.35e-02
Basic + MTQual	2.67e-03	1.37e-02
Sent-overlaps	1.28e-07	2.36e-07
Embeddings	1.09e-03	3.98e-03
n-grams	3.73e-03	3.30e-02
Finetune duration	8.38e-02	4.80e-01

Both sets of tables show the MAE and PCC obtained for each fixed test-set, for each MT quality metric (COMET22, chrF, SacreBLEU, COMET22Kiwi) and model tested (random forest, xgboost, linear).

A.3 Generic AIPs

Tables A.10, A.11, A.12, and A.13 contain the experimental results obtained for the generic FIPs for the En-Zh and En-Fr use-cases, respectively. The tables show the MAE and PCC obtained for each fixed test-set, for each MT quality metric (COMET22, chrF, SacreBLEU, COMET22Kiwi) and model tested (random forest, xgboost, linear).

A.4 Generic AIPs: Leave-one-out evaluation

The experimental results obtained for the leave-one-out cross validation study of the generic FIPs for the En-Zh use-case are displayed in Tables A.14, A.16 (for MAE) and in Tables A.15, A.17 (for PCC).

Tables A.18, A.20 and Tables A.19, A.21 contain the experimental results obtained for the leave-one-out cross validation study of the generic FIPs for the En-Fr use-cases.

Both sets of tables show the MAE and PCC obtained for each fixed test-set, for each MT quality metric (COMET22, chrF, SacreBLEU, COMET22Kiwi) and model tested (random forest, xgboost, linear).

A.5 System Utility Improvements

Figure A.1 complements the results displayed in Figure 6.14 of the main manuscript by showing results for all the delta thresholds and fine-tune costs considered. We see that Flexico is consistently closer to the optimal oracle than the simpler baselines.

Table A.2: mean absolute error (MAE) of the specific FIPs for the en-zh use-case and for the COMET22 and chrF MT metrics.

MT Metric	Pred.	Feature Set	Entertainment	Env.	Finance	Gov.	Health & Wellness	Sports	Travel & Tourism
COMET22	rf	All	0.0098	0.011	0.0078	0.0025	0.0021	0.0046	**0.0033**
		All + Kiwi	0.0102	0.0115	0.0082	0.0026	0.0022	0.0052	**0.0033**
		Basic	0.0107	0.0096	0.0065	0.0028	0.0022	0.0042	0.0034
		Basic + ContAware	0.0104	0.0096	0.0064	0.0026	0.0019	0.0052	0.0034
		Basic + Kiwi	0.0105	0.0114	0.0082	0.0025	0.0025	0.004	**0.0033**
		Basic + MTQual	0.01	0.0109	0.0076	0.0023	0.0023	0.0036	0.0035
		ContAware	0.0066	0.0054	0.0106	0.0026	0.0018	0.0051	0.0039
		ContAware-no-ngrams	0.0032	0.004	**0.0018**	**0.0015**	**0.0011**	**0.002**	0.004
		ContAware + Kiwi	0.0064	0.0061	0.012	0.0027	0.0022	0.0048	0.0037
		ContAware + MTQual	0.0061	0.0052	0.0118	0.0028	0.002	0.0044	0.0038
		MTQual	**0.0026**	**0.0026**	0.004	0.0025	0.0023	0.0033	0.0068
	xgb	All	0.0106	0.0105	0.0076	0.0027	0.0028	0.0046	0.0039
		All + Kiwi	0.0111	0.0115	0.0085	0.0036	0.0032	0.006	0.0038
		Basic	0.011	0.0096	0.0063	0.0031	0.003	0.0032	0.0044
		Basic + ContAware	0.011	0.01	0.0065	0.0032	0.0018	0.0062	0.0041
		Basic + Kiwi	0.0111	0.011	0.0091	0.0033	0.0028	0.0043	0.0042
		Basic + MTQual	0.0104	0.0107	0.0072	0.0032	0.003	0.0034	0.0046
		ContAware	0.0071	0.0077	0.0098	0.0032	0.002	0.0065	0.0041
		ContAware-no-ngrams	0.004	0.0046	**0.0027**	**0.0013**	**0.0016**	**0.0023**	**0.0036**
		ContAware + Kiwi	0.007	0.0081	0.0119	0.0034	0.0028	0.005	0.004
		ContAware + MTQual	0.0063	0.0067	0.0119	0.0029	0.002	0.0045	0.0039
		MTQual	**0.0026**	**0.0025**	0.0044	0.0022	0.0024	0.0048	0.0064
	lin	All	0.0246	0.0396	0.0242	0.0198	0.0127	0.0061	**0.0032**
		All + Kiwi	0.0097	0.0079	0.0084	0.0072	0.0087	0.0183	0.0046
		Basic	0.012	0.0096	0.0094	0.0036	0.0074	0.018	0.0066
		Basic + ContAware	0.0105	0.0134	0.0085	0.0067	0.012	0.0257	0.0078
		Basic + Kiwi	0.0209	0.0064	0.0093	0.0056	0.0059	0.0131	0.0045
		Basic + MTQual	0.0307	0.0497	0.0307	0.0217	0.0138	0.0084	0.0078
		ContAware	0.0075	0.0073	0.0106	0.0022	0.0016	0.0074	0.0114
		ContAware-no-ngrams	0.0039	0.011	0.005	0.0024	0.0018	**0.0031**	0.0049
		ContAware + Kiwi	**0.0025**	**0.0042**	0.0076	0.0021	0.0018	0.0065	0.0128
		ContAware + MTQual	0.004	0.009	0.0101	**0.0015**	**0.0013**	0.0051	0.0047
		MTQual	0.0026	0.0046	**0.004**	0.0049	0.0059	0.0188	0.0255
CHRF	rf	All	1.2988	1.8845	0.8852	0.201	0.2528	0.5287	1.3755
		All + Kiwi	1.3325	1.9266	0.8345	0.1977	0.2623	0.5125	1.3419
		Basic	1.2669	1.7982	0.7468	0.188	0.2918	**0.39**	**1.1686**
		Basic + ContAware	1.3139	1.7564	0.7176	0.2059	0.2279	0.5191	1.3622
		Basic + Kiwi	1.3154	2.0483	0.8955	**0.1723**	0.2997	0.4629	1.2647
		Basic + MTQual	1.2851	1.9641	0.8432	0.1773	0.2939	0.4712	1.3064
		ContAware	0.8964	2.5906	1.4404	0.224	0.186	0.5581	1.449
		ContAware-no-ngrams	**0.325**	**0.6658**	**0.3926**	0.2172	**0.1696**	0.6135	1.5296
		ContAware + Kiwi	0.9221	2.6954	1.5336	0.2168	0.2262	0.5575	1.4343
		ContAware + MTQual	0.8961	2.5787	1.5214	0.224	0.2012	0.5644	1.4347
		MTQual	0.3384	0.8124	0.8026	0.9022	0.5233	0.9051	1.8848
	xgb	All	1.2929	1.9174	1.0962	0.2395	0.4202	**0.4186**	1.2297
		All + Kiwi	1.3264	2.0017	0.8072	0.2501	0.3902	0.425	1.261
		Basic	1.167	1.7862	0.9736	0.2125	0.3685	0.459	**1.0149**
		Basic + ContAware	1.3251	1.9632	0.7553	0.2453	0.4014	0.4727	1.3052
		Basic + Kiwi	1.3376	1.9825	1.0281	0.2086	0.4091	0.4797	1.1483
		Basic + MTQual	1.3195	1.8725	0.8798	**0.203**	0.3978	0.4476	1.1756
		ContAware	0.9549	2.1203	1.4026	0.2806	0.314	0.454	1.3388
		ContAware-no-ngrams	**0.3777**	0.8781	**0.3986**	0.2517	**0.2109**	0.639	1.3925
		ContAware + Kiwi	0.9275	2.408	1.5952	0.2465	0.4448	0.5183	1.3323
		ContAware + MTQual	0.9516	2.0614	1.5635	0.2394	0.3848	0.5306	1.3651
		MTQual	0.4334	**0.8152**	0.6905	0.736	0.5333	0.8348	1.4902
	lin	All	2.4429	7.0028	4.6614	4.0561	3.7619	1.4253	1.2898
		All + Kiwi	1.5736	1.7068	1.9461	0.6751	0.5063	0.5218	0.8251
		Basic	2.0297	1.8097	1.5082	1.4454	0.4298	0.8861	1.3179
		Basic + ContAware	1.2713	2.2282	2.043	**0.4451**	0.6439	**0.4628**	**0.7274**
		Basic + Kiwi	2.4374	1.5041	1.0727	1.6222	0.4386	1.2873	1.4629
		Basic + MTQual	3.1717	9.6014	5.0863	3.8458	3.5403	1.0985	1.3389
		ContAware	0.5671	1.1085	0.8213	1.093	0.3859	0.4743	1.4373
		ContAware-no-ngrams	0.3349	2.0347	0.5316	0.8608	0.4463	0.7464	1.2638
		ContAware + Kiwi	**0.2521**	**0.6423**	**0.4619**	1.1642	0.4441	0.5215	1.0884
		ContAware + MTQual	0.3975	1.002	0.4656	0.4841	**0.2788**	0.9011	2.4838
		MTQual	0.9901	1.2518	1.8091	1.7137	1.1945	2.748	6.1039

Table A.3: pearson correlation coefficient (PCC) of the specific FIPs for the en-zh use-case and for the COMET22 and chrF MT metrics.

MT Metric	Pred.	Feature Set	Entertainment	Env.	Finance	Gov.	Health & Wellness	Sports	Travel & Tourism
COMET22	rf	All	61.14	92.08	88.82	95.62	90.63	76.31	84.75
		All + Kiwi	61.73	91.74	88.61	95.55	90.73	75.3	84.51
		Basic	61.52	91.88	91.2	95.15	90.61	74.98	85.08
		Basic + ContAware	60.64	91.5	**91.29**	95.06	90.8	74.41	84.21
		Basic + Kiwi	62.65	92.21	86.75	95.89	89.91	76.11	85.6
		Basic + MTQual	61.47	92.21	87.11	**95.97**	89.49	76.63	**85.66**
		ContAware	48.79	80.02	89.26	93.72	89.37	69.02	80.31
		ContAware-no-ngrams	**64.65**	**95.66**	89.92	95	**91.94**	**79.9**	85.15
		ContAware + Kiwi	51.67	78.82	87.87	93.93	87.84	70.12	80.34
		ContAware + MTQual	50.5	79.84	87.59	93.75	89.16	69.07	80.85
		MTQual	-13.14	-4.65	-48.1	-9.08	-18.69	-16.65	-28.18
	xgb	All	60.62	92.02	87.8	94.44	84.05	66.68	**84.36**
		All + Kiwi	61.25	91.7	87.18	94.29	83.24	75.95	83.12
		Basic	61.87	92.7	88.54	**96.02**	86.06	**76.71**	75.46
		Basic + ContAware	61.13	92.22	**89.22**	94.77	**90.82**	53.26	75.05
		Basic + Kiwi	62.43	91.44	82.35	93.95	86.72	74.12	77.38
		Basic + MTQual	**62.87**	92.36	84.59	94.44	85.47	74.49	74.57
		ContAware	45.19	75.3	87.9	91.89	87.8	64.35	74.6
		ContAware-no-ngrams	60.3	**93.84**	87.83	93.22	87.7	70.78	82.23
		ContAware + Kiwi	51.57	69.58	84.15	93.26	83.56	60.89	79.52
		ContAware + MTQual	52.48	76.42	84.61	92.64	86.62	66.02	79.45
		MTQual	-2.18	13.26	-33.36	31.43	-5.8	-7.05	-19.26
	lin	All	37.82	-0.81	48.22	31.34	65.76	**89.45**	90
		All + Kiwi	66.05	**95.68**	**90.11**	93.11	80.75	74.96	88.21
		Basic	55.26	94.9	85.17	78.91	-14.63	-27	56.17
		Basic + ContAware	66.1	94.16	88.22	**94.01**	56.24	37.37	74.41
		Basic + Kiwi	27.91	92.32	79.2	5.73	-28.57	-19.47	81.98
		Basic + MTQual	18.06	-15.62	21.05	-33.74	-23.96	39.99	39.39
		ContAware	13.65	86.82	77.86	91.28	86.04	64.99	63.6
		ContAware-no-ngrams	40.71	65.16	85.35	56.64	48.57	30.08	51.65
		ContAware + Kiwi	**69.39**	88.75	81.1	91.11	87.16	76.12	62.26
		ContAware + MTQual	58.75	64.83	85.26	91.35	**90.77**	88.35	**94.06**
		MTQual	37.9	10.18	23.15	28.85	34.98	48.64	69.67
CHRF	rf	All	81.97	96.13	91.4	96.78	94.27	91.77	85.51
		All + Kiwi	81.77	96.09	90.28	96.69	94.82	91.96	85.2
		Basic	82.54	95.84	91.6	97.35	95.03	93.39	90.55
		Basic + ContAware	82.15	95.7	90.36	96.34	94.57	92.41	85.41
		Basic + Kiwi	81.95	96.11	91.61	97.65	95.33	91.92	**91**
		Basic + MTQual	81.78	**96.21**	**92.24**	**97.73**	94.83	92.34	90.39
		ContAware	75.37	82.74	84.32	95.53	95.07	91.9	82.38
		ContAware-no-ngrams	**84.17**	95.39	84.88	95.32	93.95	**93.62**	85.14
		ContAware + Kiwi	75.44	82.42	84.56	95.88	**95.68**	91.06	82.82
		ContAware + MTQual	75.65	83.44	85.13	95.88	95.35	91.47	83.37
		MTQual	-12.04	-23.49	-20.34	-6.45	-15.79	-21.48	-6.98
	xgb	All	81.58	96.35	**92.66**	95.62	93.07	91.5	84.7
		All + Kiwi	81.44	96.02	90.27	95.49	93	91.69	85.36
		Basic	82.46	96.35	92.59	**97.03**	93.61	**93.18**	**89.81**
		Basic + ContAware	**83.74**	**96.39**	88.87	94.84	90.42	86.64	86.52
		Basic + Kiwi	80.64	96.15	92.46	96.64	92.82	89.86	88.73
		Basic + MTQual	81.16	96.02	91.77	96.96	91.38	91.32	88.48
		ContAware	74.33	79.55	81.63	94.7	**93.79**	89.86	85.49
		ContAware-no-ngrams	83.36	91.77	83.84	93.81	92.92	88.34	85.35
		ContAware + Kiwi	75.3	81.99	82.69	94.64	89.65	82.89	80.25
		ContAware + MTQual	74.97	82.3	82.02	95.46	91.77	86.26	81.12
		MTQual	3.81	-12.98	6.14	-1.54	-3.26	-18.95	-6.19
	lin	All	69.37	34.59	10.23	-12.44	6.43	41.16	**92.62**
		All + Kiwi	86.06	**95.7**	90.6	55.01	90.87	71.95	87.61
		Basic	70.82	93.96	89.67	-16.7	51.04	0.84	58.5
		Basic + ContAware	**88.47**	94.34	89.92	**95.51**	90.55	**84.46**	88.74
		Basic + Kiwi	48.98	94.17	87.75	-72.3	31.27	-56.55	32.54
		Basic + MTQual	36.72	15.28	-12.32	-52.58	-30.31	-10.17	78.79
		ContAware	61.99	90.79	**92.12**	85.73	91.84	82.53	82.1
		ContAware-no-ngrams	55.04	68.65	76.93	53.46	75.29	54.35	54.12
		ContAware + Kiwi	77.32	91.79	91.49	82.57	91.94	78.48	86.22
		ContAware + MTQual	85.6	83.16	92	90.94	**93.88**	83.74	83.72
		MTQual	27.16	16.42	38.37	33.73	32.16	40.13	40.74

Table A.4: mean absolute error (MAE) of the specific FIPs for the en-zh use-case and for the SacreBLEU and COMET22Kiwi MT metrics.

MT Metric	Pred.	Feature Set	Entertainment	Env.	Finance	Gov.	Health & Wellness	Sports	Travel & Tourism
SacreBleu	rf	All	0.4513	4.1395	1.0724	1.094	0.7551	0.7358	0.7954
		All + Kiwi	0.5889	4.2074	1.1478	1.1737	0.7897	0.7397	0.7454
		Basic	0.5245	3.9928	**0.5011**	0.6638	0.307	0.7121	0.6187
		Basic + ContAware	0.512	4.0452	1.13	1.1436	0.814	0.7518	0.7455
		Basic + Kiwi	0.601	4.2934	0.5124	0.6712	0.3523	0.7683	0.6535
		Basic + MTQual	0.4785	4.2096	0.5325	0.6408	0.3145	0.6743	0.717
		ContAware	0.465	4.2742	1.3669	2.0439	0.9647	0.9096	0.9184
		ContAware-no-ngrams	0.4443	1.559	0.8352	**0.4391**	**0.2933**	**0.505**	**0.6144**
		ContAware + Kiwi	0.4634	4.441	1.3982	2.1174	0.9452	0.852	0.885
		ContAware + MTQual	0.3429	4.2862	1.1328	1.6838	0.8849	0.9553	0.9285
		MTQual	**0.2384**	**0.4718**	1.1186	1.0666	0.4834	1.1127	0.984
	xgb	All	0.4949	4.0058	0.983	1.1256	0.7215	0.6591	0.9012
		All + Kiwi	0.6069	4.0857	1.1069	1.4065	0.9411	0.6439	0.755
		Basic	0.67	3.9185	**0.5318**	0.8456	0.4019	**0.5145**	**0.5164**
		Basic + ContAware	0.6579	3.8573	1.0346	1.2539	0.8677	0.5525	0.676
		Basic + Kiwi	0.6311	4.2777	0.5603	0.966	0.4172	0.6568	0.6056
		Basic + MTQual	0.5301	4.2265	0.5729	0.7328	**0.2551**	0.6292	0.6301
		ContAware	0.5941	3.9492	1.4202	1.3147	1.0642	0.8614	0.9524
		ContAware-no-ngrams	0.4686	1.578	0.8028	**0.6505**	0.3503	0.5648	0.6457
		ContAware + Kiwi	0.4851	4.2552	1.3207	1.5951	1.014	0.7156	0.9415
		ContAware + MTQual	0.3762	4.0718	1.1933	1.0224	0.7849	0.9599	0.9639
		MTQual	**0.256**	**0.5425**	0.9609	0.8061	0.3828	0.835	0.854
	lin	All	3.4864	12.261	2.4845	8.5803	3.3706	8.5884	1.4786
		All + Kiwi	2.5794	3.2655	6.2742	8.2808	1.5673	4.4889	1.7998
		Basic	1.1463	3.8732	4.5451	8.9053	1.8903	4.98	2.3538
		Basic + ContAware	2.3252	3.3868	6.7316	11.221	1.8873	7.1211	2.221
		Basic + Kiwi	1.2038	3.8149	4.1696	6.6807	1.8231	3.5084	1.8235
		Basic + MTQual	3.8911	13.7753	3.6442	7.4655	3.1398	5.6612	2.0023
		ContAware	**0.4709**	3.0819	0.9458	2.4893	0.4134	2.003	2.6343
		ContAware-no-ngrams	0.6318	3.7758	**0.7622**	1.5584	0.5532	**0.6751**	**0.8542**
		ContAware + Kiwi	0.7591	2.245	1.3426	3.1873	0.6999	4.0094	1.1261
		ContAware + MTQual	1.1344	2.31	1.3716	1.6866	**0.2569**	1.0122	1.6106
		MTQual	0.9101	**0.8251**	2.9408	**0.99**	0.6903	1.1218	1.3302
COMET22Kiwi	rf	All	**0.0016**	0.0037	0.0023	0.0009	0.0018	0.0013	**0.0018**
		All + Kiwi	**0.0016**	0.0037	0.0024	0.0009	0.0019	0.0013	**0.0018**
		Basic	0.0029	0.0019	0.0011	**0.0006**	0.0031	0.0034	0.0023
		Basic + ContAware	0.0025	0.0017	**0.001**	0.0009	0.0032	0.0025	0.0029
		Basic + Kiwi	0.0018	0.0038	0.0025	0.0008	0.0019	0.002	0.002
		Basic + MTQual	0.0018	0.0038	0.0025	0.0008	0.0018	0.0019	0.002
		ContAware	0.0026	**0.0016**	0.0016	0.0008	0.0032	0.0022	0.0024
		ContAware-no-ngrams	0.0021	0.0028	0.002	**0.0006**	0.0023	0.0015	0.0021
		ContAware + Kiwi	**0.0016**	0.0036	0.002	0.0009	0.0022	0.0013	0.0022
		ContAware + MTQual	**0.0016**	0.0036	0.0021	0.0009	0.0022	0.0013	0.0021
		MTQual	0.0018	0.0028	0.0018	0.0009	**0.0013**	**0.0012**	0.0039
	xgb	All	0.0017	0.0027	0.0021	0.0008	0.0018	**0.001**	**0.0019**
		All + Kiwi	**0.0016**	0.0027	0.0021	0.0008	0.0018	**0.001**	**0.0019**
		Basic	0.0028	0.0022	**0.001**	**0.0006**	0.0029	0.0015	0.002
		Basic + ContAware	0.0035	0.0016	**0.001**	0.0008	0.0026	0.0015	0.0027
		Basic + Kiwi	**0.0016**	0.0027	0.0022	**0.0006**	0.0018	0.0012	0.002
		Basic + MTQual	**0.0016**	0.0027	0.002	**0.0006**	0.0019	0.0011	**0.0019**
		ContAware	0.0027	**0.0015**	**0.001**	0.0008	0.0034	0.0015	0.0021
		ContAware-no-ngrams	0.0025	0.0022	0.0018	0.0008	0.0022	0.0014	0.0022
		ContAware + Kiwi	0.0017	0.0029	0.0019	0.0008	0.0017	**0.001**	0.0022
		ContAware + MTQual	0.0018	0.0028	0.0018	0.0008	0.0016	**0.001**	0.002
		MTQual	0.0018	0.0024	0.0019	0.0007	**0.0012**	0.0012	0.0033
	lin	All	0.0121	0.0107	0.006	0.001	0.0091	0.008	0.0161
		All + Kiwi	0.0057	0.0114	0.0099	0.0014	0.0083	0.0062	0.0155
		Basic	0.0066	0.0074	0.0052	0.0027	0.0062	0.0026	0.0064
		Basic + ContAware	0.0076	0.0101	0.0062	0.0033	0.008	0.0025	0.0076
		Basic + Kiwi	0.0099	0.0103	0.0093	0.0009	0.0082	0.0056	0.0138
		Basic + MTQual	0.0142	0.0079	0.0062	0.0012	0.0076	0.0056	0.0125
		ContAware	0.0067	**0.0026**	0.0058	0.0013	0.0026	0.002	0.0122
		ContAware-no-ngrams	0.0026	0.0034	0.0034	0.001	0.0028	0.0021	0.0061
		ContAware + Kiwi	**0.0019**	0.0057	0.0053	0.0007	0.0031	**0.001**	0.0051
		ContAware + MTQual	0.0043	0.0051	0.004	**0.0005**	0.0037	0.0011	0.0064
		MTQual	0.0038	0.0034	**0.0024**	0.0006	**0.0016**	0.0012	**0.0029**

Table A.5: pearson correlation coefficient (PCC) of the specific FIPs for the en-zh use-case and for the SacreBLEU and COMET22KiwiMT metrics.

MT Metric	Pred.	Feature Set	Entertainment	Env.	Finance	Gov.	Health & Wellness	Sports	Travel & Tourism
SacreBleu	rf	All	-9.67	79.61	69.54	80.72	73.18	65.25	74.38
		All + Kiwi	-16.31	79.69	69.09	80.5	72.54	68.2	76.9
		Basic	-14.83	79.69	**69.72**	74.2	**76.01**	63.47	74.82
		Basic + ContAware	-15.75	79.83	69.03	**81.14**	72.85	60.46	**80.02**
		Basic + Kiwi	-15.99	79.72	67.5	75.85	74.87	**70.49**	68.37
		Basic + MTQual	-6.68	79.64	67.3	71.51	73.25	63.16	70.35
		ContAware	-9.94	70.05	65.17	75.19	65.99	-44.19	-2.71
		ContAware-no-ngrams	-13.61	**85.29**	31.07	77.97	64.67	60.52	65.75
		ContAware + Kiwi	-12.47	70.59	65.26	74.22	66.08	-44.27	-8.55
		ContAware + MTQual	0.2	70.92	66.45	75.73	66.71	-17.01	41.66
		MTQual	**37.15**	17.25	-25.4	3.5	-7.37	-4.74	-10.62
	xgb	All	-9.11	78.86	**69.9**	68.8	72.89	67.39	29.86
		All + Kiwi	-14.08	79.86	69.76	77	68.42	73.31	40.29
		Basic	-12.59	78.86	64.03	75.55	72.86	70.46	**73.16**
		Basic + ContAware	-14.25	79.84	69.81	**79.2**	72.37	70	71.18
		Basic + Kiwi	-16.46	79.49	62.29	71.92	70.61	**74.01**	65.31
		Basic + MTQual	-4.96	79.57	59.56	73.72	**74.08**	67.86	62.99
		ContAware	-11.48	70.11	63.32	69.56	63.25	-28.95	-16.15
		ContAware-no-ngrams	-16.72	**84.12**	42.89	72.88	58.98	49.97	23.38
		ContAware + Kiwi	-1.41	69.53	64.61	68.74	63.1	-28.16	-18.44
		ContAware + MTQual	6.59	70.41	65.55	67.19	65.16	-50.25	20.82
		MTQual	**29.14**	14.08	-2.85	-4.42	7.94	1.51	-14.86
	lin	All	11.98	15.07	2.68	-58.65	-7.55	-31.57	34.23
		All + Kiwi	-5.63	84.81	51.78	18.96	36.49	17.4	44.16
		Basic	-11.78	84.01	66	65.51	36.51	54.48	38.92
		Basic + ContAware	-6.68	**85.34**	51.97	52.73	45.76	35.87	45.95
		Basic + Kiwi	-9.51	83.88	66.5	47.8	20.78	51.45	28.49
		Basic + MTQual	10.22	21.57	12.64	-48.37	-8.34	-15.89	22.87
		ContAware	-4.21	78.46	58.84	**72.48**	64.99	47.57	43.86
		ContAware-no-ngrams	-6.67	52.12	52.76	51.4	50.32	**64.12**	25.13
		ContAware + Kiwi	-5.6	80.79	56.14	68	58.61	32.39	47.4
		ContAware + MTQual	31.43	71.02	**75.21**	57.51	**77.5**	53.32	**49.39**
		MTQual	**41.44**	26.42	62.65	33.59	39.25	17.9	28.69
COMET22Kiwi	rf	All	40.04	**55.99**	64.05	50.86	27	**63.86**	**62.82**
		All + Kiwi	38.96	55.45	62.52	51.2	25.27	63.43	61.93
		Basic	10.64	-68.54	63.58	39.95	-56.63	-11.35	-18.52
		Basic + ContAware	-0.68	-34.25	-53.86	-0.82	-52.48	-6.16	-12.26
		Basic + Kiwi	35.08	55.27	63.78	**59.81**	**30.45**	61.69	55.81
		Basic + MTQual	37.48	55.42	63.67	56.13	29.68	62.43	57.66
		ContAware	-3.4	-31.83	-57.48	0.78	-48.42	-11.73	2.85
		ContAware-no-ngrams	11.35	-76.94	-48.98	0.07	-57.83	-4.19	-6.01
		ContAware + Kiwi	31.34	55.38	61.29	56.14	-9.17	63.25	61.06
		ContAware + MTQual	34.52	55.73	**64.18**	56.33	-10.05	63.74	60.36
		MTQual	**47.7**	50.97	43.28	54.18	24.06	54.36	60.17
	xgb	All	36.61	55.6	61.17	54.51	35.61	**62.26**	**67.06**
		All + Kiwi	39.02	55.6	60.51	55.82	39.03	61.16	66.78
		Basic	11.37	-62.05	-9.93	nan	-54.74	nan	6.92
		Basic + ContAware	-6.67	3.99	-18.04	-1.73	-50.86	21.2	-2.23
		Basic + Kiwi	42.06	**56.67**	60.04	**57.16**	27.67	59.75	58.94
		Basic + MTQual	**44.7**	**56.67**	59.08	53.31	24.93	57.96	60.92
		ContAware	-6.66	17.58	8.61	9.52	-46.79	-1.23	15.12
		ContAware-no-ngrams	-0.81	-73.16	-36.87	1.82	-55.75	-2.28	-2.15
		ContAware + Kiwi	26.24	54.15	**64.78**	57.06	44.65	60.28	59.07
		ContAware + MTQual	23.26	55.43	63.33	55.78	**45.5**	61.04	62.91
		MTQual	40.36	44.82	51.8	57.09	33.15	54.05	60.48
	lin	All	7.83	17.26	-58.13	60.44	-11.11	32.88	-14.08
		All + Kiwi	21.49	-18.47	-48.66	52.56	-21.86	38.6	-15.09
		Basic	13.34	-17.79	-74.01	44.53	-54.7	21.08	3.54
		Basic + ContAware	15.04	0.63	-76.09	32.87	-54.19	9.19	0.42
		Basic + Kiwi	17.83	-19.16	-45.36	57.69	-17.61	48.24	-18.16
		Basic + MTQual	4.6	-28.76	-62.48	47.9	-21.35	46.02	-16.92
		ContAware	-28.36	-69	-52.59	-6.59	-62.41	-30.09	-37.3
		ContAware-no-ngrams	0.14	-37.9	-70.65	1.02	-52.94	-12.74	-30.38
		ContAware + Kiwi	**48.98**	-44.27	-32.86	46.3	-35.03	61.75	1.32
		ContAware + MTQual	27.1	-30.15	-26.64	**61.16**	-18.6	64.63	2.9
		MTQual	39.08	**61.96**	**70.37**	58.88	**65.34**	**70.32**	**69.72**

Table A.6: mean absolute error (MAE) of the specific FIPs for the en-fr use-case and for the COMET22 and chrF MT metrics.

MT Metric	Pred.	Feature Set	Elitr	Elrc	EuBookshop	Php	Tanzil	Tedx-fr	Tedx-fr_ca
COMET22	rf	All	0.0014	**0.0016**	**0.0008**	**0.001**	0.009	0.0006	0.0008
		All + Kiwi	0.0014	0.0017	0.0011	0.0018	0.0063	0.0007	0.001
		Basic	0.0013	0.002	0.001	0.0012	0.002	0.0009	0.0009
		Basic + ContAware	**0.0012**	0.0018	0.0011	0.0015	0.0032	0.0008	0.0009
		Basic + Kiwi	0.0014	0.0018	0.0011	0.0012	**0.0017**	0.0009	0.0008
		Basic + MTQual	0.0014	**0.0016**	0.001	0.0012	0.0027	**0.0004**	**0.0006**
		ContAware	0.0027	0.0024	0.0016	**0.001**	0.0104	0.0015	0.0011
		ContAware-no-ngrams	0.0028	0.0025	0.0019	**0.001**	0.0105	0.0015	0.0011
		ContAware + Kiwi	0.0022	0.0017	0.0014	0.0014	0.0065	0.0008	0.001
		ContAware + MTQual	0.0024	**0.0016**	**0.0008**	0.0011	0.0091	**0.0004**	0.0008
		MTQual	0.0023	0.0022	0.001	0.0015	0.0033	**0.0004**	0.0009
	xgb	All	0.0015	0.0015	**0.0008**	**0.0009**	0.0056	0.0005	**0.0006**
		All + Kiwi	0.0015	0.0017	0.001	0.0014	0.0049	0.0008	0.0009
		Basic	**0.0012**	0.0021	0.001	0.001	**0.0015**	0.0007	0.0009
		Basic + ContAware	**0.0012**	0.0019	0.0011	0.0013	0.0035	0.0008	0.0009
		Basic + Kiwi	0.0013	0.002	0.0011	0.0011	0.0019	0.0007	0.0009
		Basic + MTQual	**0.0012**	**0.0014**	0.0009	0.0012	0.0033	0.0005	**0.0006**
		ContAware	0.0024	0.002	0.0012	0.001	0.0058	0.001	0.0011
		ContAware-no-ngrams	0.0025	0.0021	0.0012	0.001	0.0058	0.001	0.001
		ContAware + Kiwi	0.0019	0.0019	0.0011	0.001	0.0048	0.0008	0.0008
		ContAware + MTQual	0.0019	**0.0014**	**0.0008**	0.0013	0.0055	**0.0004**	**0.0006**
		MTQual	0.0018	0.0015	0.0009	0.0013	0.0034	**0.0004**	0.0007
	lin	All	0.0021	0.0024	0.0024	0.0031	0.0129	0.0006	0.0013
		All + Kiwi	0.0018	0.0039	0.0031	0.0048	0.0044	0.0009	0.0023
		Basic	**0.0015**	0.003	0.0012	**0.001**	0.0028	0.0006	0.0015
		Basic + ContAware	0.0025	0.0033	0.0035	0.0024	0.0035	0.0006	0.0026
		Basic + Kiwi	0.0027	0.0036	0.0013	0.003	**0.0016**	0.0007	0.0013
		Basic + MTQual	0.0027	**0.0015**	**0.0008**	0.0014	0.0096	0.0006	0.0011
		ContAware	0.0037	0.0037	0.0018	0.0033	564945.2171	0.0014	0.0015
		ContAware-no-ngrams	0.0033	0.003	0.0012	0.0026	0.0056	0.0013	0.0015
		ContAware + Kiwi	0.0032	0.0025	0.0022	0.0027	714006.1306	0.0008	0.0016
		ContAware + MTQual	0.0036	0.0024	0.0024	0.0035	21901.6952	0.0005	0.0015
		MTQual	0.0041	0.0016	0.0009	0.0015	0.0038	**0.0004**	**0.0006**
CHRF	rf	All	0.0874	0.3745	**0.1323**	0.1048	1.0928	**0.0619**	0.1114
		All + Kiwi	**0.0764**	0.2398	0.1371	0.1278	0.7233	0.1051	0.1833
		Basic	0.0954	**0.2093**	0.1372	0.2102	0.3428	0.1063	0.1442
		Basic + ContAware	0.0869	0.2305	0.1386	0.1264	0.372	0.1013	0.172
		Basic + Kiwi	0.0798	0.2236	0.1359	0.1432	**0.3232**	0.0953	0.1654
		Basic + MTQual	0.0838	0.363	0.1497	**0.0944**	0.4296	0.0651	0.1096
		ContAware	0.2745	0.2968	0.1752	0.4114	0.973	0.2495	0.3192
		ContAware-no-ngrams	0.2618	0.3074	0.1709	0.4247	1.0312	0.2425	0.3229
		ContAware + Kiwi	0.0988	0.3482	0.1359	0.1255	0.8426	0.1126	0.1694
		ContAware + MTQual	0.1551	0.4568	0.1335	0.1455	1.2017	0.0699	**0.1038**
		MTQual	0.135	0.4507	0.1379	0.1255	0.4717	0.066	0.1294
	xgb	All	0.0985	0.2597	0.1289	0.1102	0.7089	**0.0647**	0.1242
		All + Kiwi	**0.0852**	0.2257	0.1474	0.1277	0.5657	0.1329	0.1853
		Basic	0.1333	0.2129	0.1352	0.1469	**0.2314**	0.1259	0.1539
		Basic + ContAware	0.0953	**0.2126**	0.1407	0.1358	0.4612	0.1161	0.1823
		Basic + Kiwi	0.1054	0.2145	0.1407	0.1449	0.2532	0.1139	0.166
		Basic + MTQual	0.0868	0.301	**0.1213**	**0.1076**	0.4166	0.0699	0.1132
		ContAware	0.2375	0.2495	0.1734	0.3805	0.7198	0.2026	0.2074
		ContAware-no-ngrams	0.2363	0.2486	0.166	0.3724	0.7364	0.1869	0.202
		ContAware + Kiwi	0.1043	0.2804	0.1388	0.1375	0.5212	0.1266	0.1667
		ContAware + MTQual	0.1288	0.2691	0.1227	0.1828	0.7108	0.065	**0.104**
		MTQual	0.1132	0.2215	0.1272	0.1668	0.497	0.1004	0.1419
	lin	All	0.2284	0.4376	0.3785	0.5415	1.3405	0.1097	0.2703
		All + Kiwi	0.1505	0.3603	0.395	0.3829	0.4415	0.1161	0.2809
		Basic	**0.1008**	0.2437	0.258	0.4032	0.6187	0.1653	0.3276
		Basic + ContAware	0.1182	0.3354	0.5197	0.3916	0.5045	0.1665	0.2792
		Basic + Kiwi	0.1391	0.3672	0.1682	0.2448	**0.3412**	0.0772	0.3321
		Basic + MTQual	0.2641	**0.2373**	**0.1444**	0.3276	0.9824	0.1405	0.3172
		ContAware	0.3349	0.4995	0.1717	0.6531	56795008.1331	0.2245	0.3308
		ContAware-no-ngrams	0.2924	0.4215	0.159	0.6028	0.6337	0.2233	0.324
		ContAware + Kiwi	0.2076	0.343	0.179	0.2673	77225788.7501	0.1095	0.252
		ContAware + MTQual	0.2233	0.5198	0.2085	0.4271	3477672.0195	0.0872	0.1704
		MTQual	0.219	0.4053	0.1561	**0.1583**	0.5144	**0.0434**	**0.1169**

Table A.7: pearson correlation coefficient (PCC) of the specific FIPs for the en-fr use-case and for the COMET22 and chrF MT metrics.

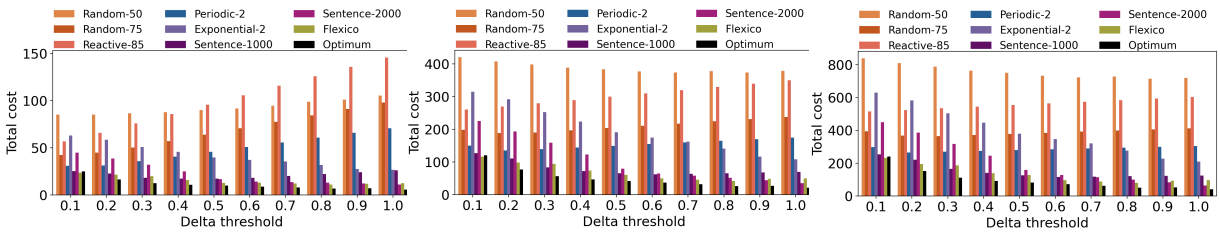
MT Metric	Pred.	Feature Set	Elitr	Elrc	EuBookshop	Php	Tanzil	Tedx-fr	Tedx-fr_ca
COMET22	rf	All	57.52	**57.42**	61.24	61.5	58.44	7.07	71.42
		All + Kiwi	57.05	7.92	-0.32	6.34	42.98	-0.36	31.76
		Basic	66.62	10.43	18.29	1.73	-10.47	3.1	17.41
		Basic + ContAware	**67.11**	-10.19	3.14	-0.97	-13.97	-2.68	2.68
		Basic + Kiwi	53.83	5.17	14.72	4.86	46.11	8.4	45.31
		Basic + MTQual	58.16	53.19	**65.65**	64.41	59.69	46.92	**77.01**
		ContAware	14.9	-22.55	-17.1	-3.38	18.88	21.52	-8.58
		ContAware-no-ngrams	10.77	-14.07	3.03	-5.25	18.77	17	15.86
		ContAware + Kiwi	-8.58	10.33	-12.93	5	66	23.5	32.24
		ContAware + MTQual	5.9	53.88	60.09	**65.02**	**72.98**	**54.23**	70.55
		MTQual	7.95	37.52	57.52	52.53	20.99	29.15	64.64
	xgb	All	32.58	55.99	61.4	58.24	44.3	18.76	**73.94**
		All + Kiwi	39.23	11.7	9.94	-0.79	39.13	6.8	18.26
		Basic	**65.02**	10.03	7.56	0.81	-16.04	2.93	15.85
		Basic + ContAware	57.32	10.02	12.82	-2.73	-17	3.04	-7.67
		Basic + Kiwi	48.73	2.56	5.75	5.49	42.46	17.18	29.27
		Basic + MTQual	61.07	**60.49**	**61.54**	**62.92**	43.25	**54.38**	68.48
		ContAware	13.56	9.46	-9.37	1.66	5.82	3.17	-18.93
		ContAware-no-ngrams	9.19	-15.6	6.81	7.04	5.83	3.17	10.36
		ContAware + Kiwi	-17.9	0.37	-5.91	2.6	53.33	12.94	32.02
		ContAware + MTQual	-19.54	59.69	61.16	53.04	**58.04**	43.96	71.63
		MTQual	20.75	57.3	56.21	50.71	9.57	27.7	59.24
	lin	All	63.9	41.31	17.63	28.58	36.02	59.56	28.69
		All + Kiwi	57.48	11.89	3.82	21.06	19.01	31.11	15.8
		Basic	58.45	-5.44	-0.43	42.72	-23.07	15.62	10.48
		Basic + ContAware	53.56	8.06	-2	19.45	-6.35	22.29	5.79
		Basic + Kiwi	70.64	3.22	16.84	49.25	37.74	23.25	30.42
		Basic + MTQual	**81.21**	**63.05**	63.07	**74.56**	67.86	**60.74**	65.06
		ContAware	19.53	10.63	-3.64	6.63	7.38	8.53	-3.02
		ContAware-no-ngrams	24.96	-0.53	27.75	-16.51	2.51	12.05	-2.21
		ContAware + Kiwi	35.76	11.27	0.95	6.38	7.38	19.07	7.06
		ContAware + MTQual	47.42	41.55	15.7	22.37	7.38	43.07	27.67
		MTQual	59.15	61.9	**66.09**	58.24	**72.93**	59.94	**74.84**
CHRF	rf	All	40.54	12.74	31.94	61.95	54.13	**34.4**	21.8
		All + Kiwi	50.79	29.18	-3.24	17.27	46.02	6.77	-26.25
		Basic	46.04	28.29	5.77	2.16	7.48	-8.13	-23.13
		Basic + ContAware	46.02	**30.93**	1.43	36.68	-5.42	-4.29	-24.59
		Basic + Kiwi	**54.87**	25.51	-6.45	9.96	47.29	8.1	-18.8
		Basic + MTQual	48.78	18.28	**54.69**	**65.22**	**57.81**	25.24	20.52
		ContAware	7.71	-0.77	-2.54	7.56	-0.56	6.74	-2.12
		ContAware-no-ngrams	10.63	-11.36	-3.55	-3.08	-7.66	10.19	4.21
		ContAware + Kiwi	20.58	-21.73	2.23	25.81	43.03	18.58	-6.01
		ContAware + MTQual	4.86	-1.68	33.77	44.41	53.87	18.42	**34.81**
		MTQual	17.42	22.5	26.12	51.73	36.62	25.83	1.56
	xgb	All	27.55	19.58	24.72	**62.64**	35.43	45.63	**33.79**
		All + Kiwi	46	13.21	3.22	29.46	39.97	1.99	-17.22
		Basic	44.07	25.49	2.46	-15.56	-5.68	-4.73	-21.49
		Basic + ContAware	43.69	**33.15**	-0.75	13.82	11.86	-6.26	0.28
		Basic + Kiwi	**48.57**	25.29	-3.52	17.18	42.84	8.54	-22.63
		Basic + MTQual	42.6	0.16	**39.63**	59.17	**47.9**	38.61	26.25
		ContAware	18.72	15.9	-22.08	18.24	4.1	2.79	-3.94
		ContAware-no-ngrams	13.37	15.49	-20.75	-4.85	1.79	7.95	4.83
		ContAware + Kiwi	16.22	-0.8	14.73	33.48	24.96	4.8	-7.37
		ContAware + MTQual	1.96	30.56	35.1	30.85	31.54	**47.3**	33.53
		MTQual	7.96	22.63	17.67	27.56	23.19	21.98	-6.25
	lin	All	41.9	13.8	41.54	**35.12**	30.04	48.13	12.51
		All + Kiwi	25.2	-8.26	25.58	22.15	2.64	24.54	-22.2
		Basic	52.53	19.1	15.16	5.18	-9.04	-4.15	-24.4
		Basic + ContAware	40.23	2.82	17.14	16.9	-15	-2.84	-19.15
		Basic + Kiwi	42.94	6.46	29.58	7.22	22.82	37.74	-28
		Basic + MTQual	58.72	**64.68**	58.96	32.94	60.47	66.01	20.08
		ContAware	-4.29	6.58	13.3	21.51	-16.4	-2.83	-11.5
		ContAware-no-ngrams	22.47	1.92	-2.71	-21.42	-2.01	2.12	-17.45
		ContAware + Kiwi	8.23	-10.43	20.88	17.73	-16.4	19.78	-17.16
		ContAware + MTQual	37.62	12.54	43.04	33.79	-16.4	39.93	17.46
		MTQual	**63.03**	51.72	**59.45**	19.98	**70.42**	**76.08**	**49.76**

Table A.8: mean absolute error (MAE) of the specific FIPs for the en-fr use-case and for the SacreBLEU and COMET22Kiwi MT metrics.

MT Metric	Pred.	Feature Set	Elitr	Elrc	EuBookshop	Php	Tanzil	Tedx-fr	Tedx-fr_ca
SacreBleu	rf	All	0.1821	0.607	0.2525	0.3795	0.7009	0.1037	0.2352
		All + Kiwi	0.1407	0.3684	**0.2174**	0.1861	0.4563	0.1627	0.2669
		Basic	**0.1211**	**0.3272**	0.2198	0.2177	0.3506	0.1573	0.219
		Basic + ContAware	0.1412	0.3546	0.2274	0.1798	0.4137	0.1609	0.2612
		Basic + Kiwi	0.1334	0.3454	0.2184	0.1911	**0.2303**	0.1709	0.2244
		Basic + MTQual	0.1711	0.6225	0.2984	0.3712	0.4046	0.1018	0.2223
		ContAware	0.5934	0.3829	0.3185	0.8886	0.7866	0.2897	0.3253
		ContAware-no-ngrams	0.5624	0.3633	0.3862	0.9039	0.8778	0.2926	0.351
		ContAware + Kiwi	0.1808	0.3985	0.2335	0.1878	0.6793	0.1395	0.208
		ContAware + MTQual	0.2883	0.6237	0.2507	**0.1444**	0.7186	0.1085	**0.2075**
		MTQual	0.26	0.7039	0.2856	0.1533	0.3512	**0.0965**	0.2374
	xgb	All	0.1711	0.4229	**0.2022**	0.4112	0.42	**0.1072**	0.221
		All + Kiwi	0.1372	0.3491	0.2284	0.1824	0.3981	0.2291	0.2703
		Basic	0.1745	**0.3211**	0.23	0.1922	**0.2088**	0.1874	0.234
		Basic + ContAware	0.1378	0.3583	0.2458	0.1965	0.3427	0.1974	0.2726
		Basic + Kiwi	0.1371	0.3377	0.2475	0.1996	0.2591	0.2285	0.2348
		Basic + MTQual	**0.1337**	0.3781	0.2086	0.378	0.2441	0.1311	**0.1984**
		ContAware	0.5548	0.4111	0.3257	0.7955	0.4136	0.265	0.2905
		ContAware-no-ngrams	0.5149	0.3394	0.2967	0.8688	0.4422	0.3052	0.3028
		ContAware + Kiwi	0.1445	0.3607	0.2526	0.2124	0.3896	0.1754	0.2379
		ContAware + MTQual	0.187	0.4961	0.2086	0.1967	0.4043	0.128	0.2032
		MTQual	0.1851	0.3948	0.2136	**0.1702**	0.3709	0.1258	0.2306
	lin	All	0.4789	0.7229	0.2013	0.9234	0.7736	0.1898	0.4702
		All + Kiwi	0.303	0.8847	0.2512	0.5576	0.4963	0.3342	0.3086
		Basic	0.2488	0.5475	0.2721	0.7829	**0.2884**	0.3472	0.3082
		Basic + ContAware	**0.2444**	0.5835	0.2895	0.6555	0.4165	0.4436	0.3147
		Basic + Kiwi	0.3447	0.6557	0.2318	0.6056	0.293	0.1975	0.2992
		Basic + MTQual	0.5744	**0.3539**	**0.1805**	0.736	0.6751	0.2751	0.4794
		ContAware	0.6226	0.7236	0.3965	1.1769	24873987.7997	0.3524	0.4119
		ContAware-no-ngrams	0.6052	0.4368	0.4516	1.1257	0.449	0.3384	0.4259
		ContAware + Kiwi	0.4812	0.511	0.309	0.5688	22193794.0062	0.1916	0.2899
		ContAware + MTQual	0.299	0.4863	0.2697	0.6586	729107.0116	0.1656	0.236
		MTQual	0.3104	0.6357	0.2745	**0.2041**	0.3778	**0.1038**	**0.1825**
COMET22Kiwi	rf	All	0.0012	**0.0026**	0.0018	0.0035	0.002	**0.0003**	**0.0007**
		All + Kiwi	0.0013	**0.0026**	0.0018	0.004	0.002	**0.0003**	**0.0007**
		Basic	**0.0011**	0.003	**0.0014**	**0.0029**	0.0016	0.001	0.0012
		Basic + ContAware	**0.0011**	**0.0026**	0.0017	0.0035	0.0018	0.0005	0.001
		Basic + Kiwi	0.0012	0.0036	**0.0014**	0.0037	0.0011	0.0007	0.0009
		Basic + MTQual	**0.0011**	0.0034	0.0015	0.0036	**0.001**	0.0006	0.001
		ContAware	0.0048	0.004	0.0025	0.0086	0.0034	0.0004	0.0011
		ContAware-no-ngrams	0.0048	0.0043	0.0024	0.0085	0.0035	0.0004	0.0011
		ContAware + Kiwi	0.0031	**0.0026**	0.0019	0.004	0.002	**0.0003**	**0.0007**
		ContAware + MTQual	0.0029	**0.0026**	0.002	0.004	0.002	**0.0003**	**0.0007**
		MTQual	0.0027	0.0032	0.0018	0.0043	0.0012	0.0004	0.0008
	xgb	All	**0.0013**	0.0026	0.0016	0.003	0.002	**0.0003**	**0.0007**
		All + Kiwi	**0.0013**	0.0027	0.0016	0.0034	0.0021	**0.0003**	**0.0007**
		Basic	0.0015	0.003	0.0015	0.0034	0.0013	0.0005	0.001
		Basic + ContAware	**0.0013**	**0.0025**	0.0016	0.0031	0.0018	0.0004	0.001
		Basic + Kiwi	**0.0013**	0.0033	**0.0014**	0.0032	**0.001**	0.0005	0.0009
		Basic + MTQual	**0.0013**	0.0032	**0.0014**	**0.0029**	0.0011	0.0004	0.0009
		ContAware	0.0043	0.0037	0.0026	0.0083	0.0022	0.0004	0.0011
		ContAware-no-ngrams	0.0042	0.0037	0.0022	0.0082	0.0022	0.0004	0.0011
		ContAware + Kiwi	0.0028	0.0026	0.0017	0.0034	0.0022	**0.0003**	**0.0007**
		ContAware + MTQual	0.0024	**0.0025**	0.0016	0.003	0.0021	**0.0003**	**0.0007**
		MTQual	0.0026	0.0032	0.0015	0.0032	0.0011	0.0004	0.0008
	lin	All	0.0049	0.0023	0.0019	0.0112	0.0045	0.0004	0.0013
		All + Kiwi	0.005	0.0023	0.002	0.0105	0.0053	0.0004	0.0011
		Basic	**0.0023**	0.0041	0.0021	0.0044	0.0013	0.0007	0.0012
		Basic + ContAware	0.0028	0.0054	0.002	0.0052	0.0035	0.0007	0.0015
		Basic + Kiwi	0.0058	0.0027	0.0019	0.009	0.0022	**0.0003**	0.0014
		Basic + MTQual	0.0051	**0.0022**	0.0019	0.0087	0.0024	**0.0003**	0.0014
		ContAware	0.0067	0.0041	0.0034	0.0151	348473.9762	0.0005	0.0016
		ContAware-no-ngrams	0.0061	0.0044	0.0028	0.0129	0.0025	0.0005	0.0014
		ContAware + Kiwi	0.0054	0.0034	0.0018	0.0051	379450.0779	0.0004	0.0011
		ContAware + MTQual	0.0051	0.0032	0.0018	0.0054	10484.5242	0.0005	0.0012
		MTQual	0.0063	0.003	**0.0015**	**0.0039**	**0.0009**	0.0004	**0.0006**

Table A.9: pearson correlation coefficient (PCC) of the specific FIPs for the en-fr use-case and for the SacreBLEU and COMET22Kiwi MT metrics.

MT Metric	Pred.	Feature Set	Elitr	Elrc	EuBookshop	Php	Tanzil	Tedx-fr	Tedx-fr_ca
SacreBleu	rf	All	52.22	24.68	47.19	33.55	65.53	22.42	15.06
		All + Kiwi	45.82	-5.38	19.86	23.67	33.73	-12.55	-16.17
		Basic	39.58	**32.58**	5.71	12.02	-3.36	-3.46	-5.87
		Basic + ContAware	34.64	-6.99	11.89	33.61	-2.38	-8.39	-14.51
		Basic + Kiwi	**55.86**	18.7	20.13	30.54	58.49	-5.38	-9.81
		Basic + MTQual	55.3	29.99	45.68	34.63	**70.94**	46.33	18.83
		ContAware	18.24	-14.95	10.83	3.04	-4.9	1.04	7.33
		ContAware-no-ngrams	16.51	-1.29	-3.49	-11.45	-10.64	-24.02	-6.18
		ContAware + Kiwi	33.31	-13.9	19.08	19.64	35.26	-25.08	3.75
		ContAware + MTQual	30.54	13.74	**47.2**	**65.97**	55.02	23.8	38.46
		MTQual	32.26	25.26	40.1	62.75	46.58	**50.35**	**38.47**
	xgb	All	53.47	20.86	**44.43**	26.96	46.8	31.15	7.58
		All + Kiwi	41.67	13.54	14.14	27.33	36.49	-11.13	-5.37
		Basic	45.74	**28.95**	2.9	-5.61	3.14	-5.87	-6.94
		Basic + ContAware	33.65	-6.4	-2.5	16.3	3.11	-3.52	-5.51
		Basic + Kiwi	56.02	13.16	15.97	14.17	54.71	-7.55	-13.3
		Basic + MTQual	**59.17**	10.8	35.74	30.46	**67.76**	37.9	**26.88**
		ContAware	8.4	-16.21	-6.69	-18.13	7.27	0.06	4.11
		ContAware-no-ngrams	-5.55	15.1	9.12	-4.7	1.84	14.83	-15.78
		ContAware + Kiwi	34.12	-4.28	10.11	3.05	39.87	-30.4	-7.75
		ContAware + MTQual	28.76	5.63	37.66	42.26	43.61	9.52	17.4
		MTQual	37.59	15.18	42.35	**58.55**	39.99	**45.68**	1.09
	lin	All	46.65	20.17	63	34.63	45.92	30.76	30.01
		All + Kiwi	28.54	2.09	37.34	23.91	28.27	14.69	3.01
		Basic	51.18	6.71	-10.48	3.51	-3.43	-14.28	-7.8
		Basic + ContAware	44.85	2.67	27.62	18.42	15.39	-3.47	2.07
		Basic + Kiwi	41.73	10.63	13.21	6.49	25.57	21.83	-7.15
		Basic + MTQual	59.32	**56.92**	61.28	**35.3**	69.28	43.42	25.2
		ContAware	0.35	6.46	27.5	21.13	-21.92	3.84	7.75
		ContAware-no-ngrams	24.18	16.97	-14.22	-13.71	-13.14	-0.51	4.72
		ContAware + Kiwi	14.51	1.3	26.19	16.82	-21.92	13.49	9.52
		ContAware + MTQual	41.3	29.85	49.35	29	-21.92	24.77	34.58
		MTQual	**60.76**	50.48	**64.47**	26.44	**73.85**	**51.92**	**54.75**
COMET22Kiwi	rf	All	78.31	52.4	14.22	28.05	56.88	54.76	67.3
		All + Kiwi	76.99	52.27	13.18	19.77	61.78	54.2	68.81
		Basic	**81.4**	7.91	13.5	-19.47	2.23	-7.65	7.1
		Basic + ContAware	81.3	27.1	-28.17	14.14	6.12	-16.76	0.06
		Basic + Kiwi	80.32	43.12	36.96	20.5	**67.62**	**63.27**	**76.17**
		Basic + MTQual	80.44	48.44	**39.13**	**28.67**	60.22	52.57	62.22
		ContAware	-5.08	4.38	-34.86	11.31	9.89	-13.54	6.14
		ContAware-no-ngrams	-5.41	-3.73	-37.54	11.99	14.62	15.33	7.91
		ContAware + Kiwi	-4	51.26	5.79	20.05	66.06	51.15	68.96
		ContAware + MTQual	-8.29	**53.67**	3.46	16.43	63.47	51.82	66.72
		MTQual	2.01	41.41	25.76	22.62	48.39	37.45	67.43
	xgb	All	75.14	41.19	-0.26	24.12	36.84	56.96	70.1
		All + Kiwi	71.05	37.03	4.8	14.19	31.83	59.78	73.3
		Basic	**77.65**	22.64	13.88	-46.53	-14.34	nan	2.76
		Basic + ContAware	74.77	39.65	-10.59	13.56	-9.35	-20.5	12.62
		Basic + Kiwi	70.96	37.89	25.24	9.62	**63.56**	**63.27**	**76.57**
		Basic + MTQual	74.92	47.52	**33.64**	**24.84**	53.38	57.72	63.43
		ContAware	3.24	-13.44	-32.19	7.21	-7.66	-7.72	10.75
		ContAware-no-ngrams	-11.57	-18.9	-30.43	8.28	6.6	1.52	10.24
		ContAware + Kiwi	-14.03	36.65	-6.71	15.96	42.94	44.28	69.67
		ContAware + MTQual	7.71	**47.96**	-8.29	11.32	46.45	51.26	72.25
		MTQual	2.68	41.77	20.32	16.1	55.14	43.76	69.25
	lin	All	68.82	69.8	31.86	41.85	48.17	47.16	40.31
		All + Kiwi	70.16	64.83	29.28	50.51	48.72	55.72	43.43
		Basic	73.94	19.06	14.52	40.09	-14.58	6.67	8.52
		Basic + ContAware	68.57	26.28	-2.76	34.23	14.84	12.51	7.16
		Basic + Kiwi	**86.16**	77.1	**61**	76.3	**74.12**	**63.67**	76.33
		Basic + MTQual	85.95	**77.98**	41.82	**81.5**	72.71	57.99	80.86
		ContAware	10.69	24.54	-12.08	21.65	11.76	1.96	5.92
		ContAware-no-ngrams	12.33	11.78	-11.05	24.09	-2.94	-30.42	-4.21
		ContAware + Kiwi	40.86	56.58	23.81	22.77	11.76	51.84	42.6
		ContAware + MTQual	41.24	59	28.21	10.18	11.76	44.84	38.32
		MTQual	53.67	64.67	35.31	43.15	73.52	61.24	**81.53**



(a) Fine-tune cost 1. (b) Fine-tune cost 5. (c) Fine-tune cost 10.

Figure A.1: Total cost incurred by each baseline as a function of: the fine-tune cost, the target delta improvement for MT metrics.

Table A.10: MAE and PCC of the generic FIPs for the en-zh use-case and for the COMET22 and chrF MT metrics.

MT Metric	Predictor	Feature Set	MAE	PCC
COMET22	rf	All	0.0044	72.76
		All + Kiwi	0.0045	71.46
		Basic	0.0048	69.4
		Basic + ContAware	0.0045	69.34
		Basic + Kiwi	0.0051	69.72
		Basic + MTQual	0.0049	**76.62**
		ContAware	**0.0026**	67.9
		ContAware-no-ngrams	0.0027	57.78
		ContAware + Kiwi	**0.0026**	71.29
		ContAware + MTQual	**0.0026**	71.86
		MTQual	0.0035	17.04
	xgb	All	0.0045	72.03
		All + Kiwi	0.0044	64.94
		Basic	0.004	65.31
		Basic + ContAware	0.0044	60.25
		Basic + Kiwi	0.005	70.22
		Basic + MTQual	0.0053	**76.51**
		ContAware	**0.0023**	69.07
		ContAware-no-ngrams	0.0027	55.07
		ContAware + Kiwi	0.0026	68.47
		ContAware + MTQual	0.0032	68.48
		MTQual	0.004	18.06
	lin	All	0.0177	35.75
		All + Kiwi	0.0136	61.78
		Basic	0.0075	60.9
		Basic + ContAware	0.0141	61.64
		Basic + Kiwi	0.0066	**68.3**
		Basic + MTQual	0.0073	53.52
		ContAware	0.0062	57.25
		ContAware-no-ngrams	**0.0045**	44.28
		ContAware + Kiwi	0.0059	56.33
		ContAware + MTQual	0.0063	46.15
		MTQual	0.0057	10.19
CHRF	rf	All	0.6206	58.05
		All + Kiwi	0.6487	57.59
		Basic	**0.5614**	**72.8**
		Basic + ContAware	0.6717	56.18
		Basic + Kiwi	0.6202	66.08
		Basic + MTQual	0.6056	70.69
		ContAware	0.6958	51.42
		ContAware-no-ngrams	0.5648	55.5
		ContAware + Kiwi	0.6424	53.65
		ContAware + MTQual	0.5943	56.92
		MTQual	0.7524	28.06
	xgb	All	0.7377	53.23
		All + Kiwi	0.6939	57.47
		Basic	0.5472	69.74
		Basic + ContAware	0.7504	48.86
		Basic + Kiwi	**0.5455**	**74.07**
		Basic + MTQual	0.5739	69.38
		ContAware	0.7471	47.49
		ContAware-no-ngrams	0.5777	55.45
		ContAware + Kiwi	0.6743	52.12
		ContAware + MTQual	0.6908	44.17
		MTQual	0.8059	29.61
	lin	All	1.6522	41.38
		All + Kiwi	1.0963	64.43
		Basic	0.6292	62.72
		Basic + ContAware	1.1079	64.51
		Basic + Kiwi	**0.6258**	**71.87**
		Basic + MTQual	1.2031	42.53
		ContAware	1.3265	51.4
		ContAware-no-ngrams	1.2906	33.17
		ContAware + Kiwi	1.2834	49.45
		ContAware + MTQual	1.4369	47.16
		MTQual	1.3162	9.84

Table A.11: MAE and PCC of the generic FIPs for the en-zh use-case and for the SacreBLEU and COMET22Kiwi MT metrics.

MT Metric	Predictor	Feature Set	MAE	PCC
SacreBleu	rf	All	1.0156	55.24
		All + Kiwi	1.1691	**58.84**
		Basic	0.6732	58.37
		Basic + ContAware	1.1031	56.94
		Basic + Kiwi	1.0263	41.83
		Basic + MTQual	1.4243	47.68
		ContAware	1.0664	52.3
		ContAware-no-ngrams	**0.6172**	39.55
		ContAware + Kiwi	0.9828	47.83
		ContAware + MTQual	0.992	53.02
		MTQual	0.8047	12.42
	xgb	All	1.0205	49.95
		All + Kiwi	1.106	**58.98**
		Basic	0.6623	57.38
		Basic + ContAware	1.202	57.77
		Basic + Kiwi	1.0439	43.34
		Basic + MTQual	1.5072	44.38
		ContAware	1.1257	50.65
		ContAware-no-ngrams	**0.6595**	35.72
		ContAware + Kiwi	1.001	44.36
		ContAware + MTQual	0.9409	53.46
		MTQual	0.7512	11.52
	lin	All	2.1411	34.27
		All + Kiwi	4.9097	53.16
		Basic	2.9786	52.43
		Basic + ContAware	4.697	54.23
		Basic + Kiwi	3.2661	52.89
		Basic + MTQual	**1.1195**	**71.36**
		ContAware	1.3521	59.12
		ContAware-no-ngrams	1.5146	35.42
		ContAware + Kiwi	1.3328	57.54
		ContAware + MTQual	1.2565	46.69
		MTQual	1.2566	42.22
COMET22Kiwi	rf	All	0.0018	**26.1**
		All + Kiwi	0.0019	22.15
		Basic	**0.0013**	-31.58
		Basic + ContAware	0.002	-18.62
		Basic + Kiwi	0.002	15.32
		Basic + MTQual	0.0019	5.66
		ContAware	0.0021	-20.12
		ContAware-no-ngrams	0.0019	-17.03
		ContAware + Kiwi	0.002	12.13
		ContAware + MTQual	0.0022	4.03
		MTQual	0.0019	16.15
	xgb	All	0.0017	**31.5**
		All + Kiwi	0.0016	30.6
		Basic	**0.0012**	-19.91
		Basic + ContAware	0.0018	-12.42
		Basic + Kiwi	0.0017	24.36
		Basic + MTQual	0.0015	19.84
		ContAware	0.0018	-0.55
		ContAware-no-ngrams	0.0017	-15.95
		ContAware + Kiwi	0.0022	14.93
		ContAware + MTQual	0.002	9.65
		MTQual	0.0017	17.89
	lin	All	0.0041	-23.32
		All + Kiwi	0.0043	-28.39
		Basic	**0.002**	-14.79
		Basic + ContAware	0.0041	-25.7
		Basic + Kiwi	0.0025	-23.04
		Basic + MTQual	0.0024	-16.44
		ContAware	0.0025	-15.52
		ContAware-no-ngrams	0.0027	-9.3
		ContAware + Kiwi	0.0025	-19.3
		ContAware + MTQual	0.005	-16.56
		MTQual	0.0036	**-6.41**

Table A.12: MAE and PCC of the generic FIPs for the en-fr use-case and for the COMET22 and chrF MT metrics.

MT Metric	Predictor	Feature Set	MAE	PCC
COMET22	rf	All	0.002	3.12
		All + Kiwi	0.0019	9.43
		Basic	0.0013	2.87
		Basic + ContAware	0.0014	24.32
		Basic + Kiwi	**0.0012**	**39.79**
		Basic + MTQual	0.0013	29.9
		ContAware	0.0026	15.18
		ContAware-no-ngrams	0.0028	12.21
		ContAware + Kiwi	0.0024	1.13
		ContAware + MTQual	0.0021	0.67
		MTQual	0.0016	0.02
	xgb	All	0.0015	-2.64
		All + Kiwi	0.0015	12.42
		Basic	0.0013	3.74
		Basic + ContAware	0.0013	27.61
		Basic + Kiwi	**0.0012**	**39.29**
		Basic + MTQual	**0.0012**	33.72
		ContAware	0.0015	17.66
		ContAware-no-ngrams	0.0014	16.69
		ContAware + Kiwi	0.0015	5.66
		ContAware + MTQual	0.0015	-0.61
		MTQual	0.0014	7.61
	lin	All	0.002	5.76
		All + Kiwi	0.0019	25.17
		Basic	**0.0013**	2.51
		Basic + ContAware	0.0019	24.83
		Basic + Kiwi	**0.0013**	7.62
		Basic + MTQual	0.002	5.26
		ContAware	0.0018	25.11
		ContAware-no-ngrams	0.0018	24.83
		ContAware + Kiwi	0.0018	**25.74**
		ContAware + MTQual	0.0019	3.2
		MTQual	0.0019	4.5
CHRF	rf	All	0.2474	-1.01
		All + Kiwi	0.2147	13.04
		Basic	**0.1603**	-1.39
		Basic + ContAware	0.204	7.07
		Basic + Kiwi	0.1636	**37.79**
		Basic + MTQual	0.1982	5.26
		ContAware	0.3443	6.67
		ContAware-no-ngrams	0.3892	2.69
		ContAware + Kiwi	0.2877	7.23
		ContAware + MTQual	0.2732	-2.43
		MTQual	0.2002	-2.44
	xgb	All	0.1618	-7.46
		All + Kiwi	0.1563	4.46
		Basic	0.1508	-0.9
		Basic + ContAware	0.167	14.62
		Basic + Kiwi	**0.1474**	**37.82**
		Basic + MTQual	0.17	11.68
		ContAware	0.1699	13.94
		ContAware-no-ngrams	0.1893	-2.63
		ContAware + Kiwi	0.1657	2.83
		ContAware + MTQual	0.1632	-1.9
		MTQual	0.1726	-0.31
	lin	All	0.233	10.29
		All + Kiwi	0.2426	5.31
		Basic	**0.15**	-4.22
		Basic + ContAware	0.2339	**14.03**
		Basic + Kiwi	0.2009	-8.1
		Basic + MTQual	0.238	-0.4
		ContAware	0.2364	14.02
		ContAware-no-ngrams	0.2247	5.59
		ContAware + Kiwi	0.2517	5.07
		ContAware + MTQual	0.227	10.78
		MTQual	0.2563	0.46

Table A.13: MAE and PCC of the generic FIPs for the en-fr use-case and for the SacreBLEU and COMET22Kiwi MT metrics.

MT Metric	Predictor	Feature Set	MAE	PCC
SacreBleu	rf	All	0.281	13.59
		All + Kiwi	0.2446	11.4
		Basic	**0.2085**	-1.59
		Basic + ContAware	0.2403	8.57
		Basic + Kiwi	0.2102	**26.97**
		Basic + MTQual	0.2333	16.56
		ContAware	0.4918	18.92
		ContAware-no-ngrams	0.501	18.19
		ContAware + Kiwi	0.3139	10.25
		ContAware + MTQual	0.3054	10.21
		MTQual	0.2453	3.4
	xgb	All	0.2666	11.09
		All + Kiwi	0.237	11.87
		Basic	0.2139	-1.74
		Basic + ContAware	0.2328	18.57
		Basic + Kiwi	**0.211**	**27.33**
		Basic + MTQual	0.2174	15.08
		ContAware	0.3226	18.15
		ContAware-no-ngrams	0.3308	9.94
		ContAware + Kiwi	0.2669	12.31
		ContAware + MTQual	0.2572	15.25
		MTQual	0.2342	5.74
	lin	All	0.3381	26.64
		All + Kiwi	0.3696	17.96
		Basic	**0.2361**	-2.99
		Basic + ContAware	0.3563	24.5
		Basic + Kiwi	0.2726	1.81
		Basic + MTQual	0.3397	10.98
		ContAware	0.3913	25.5
		ContAware-no-ngrams	0.3693	15.93
		ContAware + Kiwi	0.4049	17.9
		ContAware + MTQual	0.3268	**26.9**
		MTQual	0.374	12.68
COMET22Kiwi	rf	All	0.0017	42.39
		All + Kiwi	0.0017	44.65
		Basic	0.0019	2.12
		Basic + ContAware	0.0019	28.34
		Basic + Kiwi	0.0017	37.19
		Basic + MTQual	**0.0016**	**52.06**
		ContAware	0.0035	41.94
		ContAware-no-ngrams	0.0036	41.88
		ContAware + Kiwi	0.0019	28.66
		ContAware + MTQual	0.0019	20.69
		MTQual	0.0019	19.6
	xgb	All	0.0017	38.18
		All + Kiwi	0.0018	35.84
		Basic	0.0019	-19.94
		Basic + ContAware	0.0019	25.42
		Basic + Kiwi	0.0019	25.64
		Basic + MTQual	**0.0016**	**49.44**
		ContAware	0.0029	32.45
		ContAware-no-ngrams	0.0029	36.31
		ContAware + Kiwi	0.002	16.3
		ContAware + MTQual	0.0021	-0.78
		MTQual	0.002	13.58
	lin	All	0.0026	47.12
		All + Kiwi	0.0028	**49.42**
		Basic	0.0023	20.2
		Basic + ContAware	0.0028	32.3
		Basic + Kiwi	**0.0022**	41.73
		Basic + MTQual	0.0028	45.93
		ContAware	0.0036	26.97
		ContAware-no-ngrams	0.0035	17.24
		ContAware + Kiwi	0.0032	46.42
		ContAware + MTQual	0.0028	35.3
		MTQual	0.0028	40.33

Table A.14: mean absolute error (MAE) of the generic FIPs when each dataset is left-out of the training set, for the en-zh use-case and for the COMET22 and chrF MT metrics.

MT Metric	Pred.	Feature Set	Entertainment	Env.	Finance	Gov.	Health & Wellness	Sports	Travel & Tourism
COMET22	rf	All	0.0065	0.0038	0.0033	0.0076	0.0056	0.0051	0.004
		All + Kiwi	0.0063	0.0049	0.0037	0.0075	0.0073	0.0049	0.0039
		Basic	0.0062	0.0044	0.004	0.007	0.0075	0.0056	**0.0031**
		Basic + ContAware	0.0064	0.0041	0.004	0.0087	0.0086	0.0051	0.0039
		Basic + Kiwi	0.0054	0.0059	0.0026	0.0045	0.0082	0.0072	0.005
		Basic + MTQual	0.0069	0.0029	0.0026	0.0045	0.0062	0.0075	0.006
		ContAware	0.0049	0.0026	0.0025	0.0034	**0.0019**	**0.0015**	0.0043
		ContAware-no-ngrams	**0.002**	**0.0016**	**0.0009**	0.0038	0.0041	**0.0015**	0.0063
		ContAware + Kiwi	0.0043	0.0031	0.0023	0.0039	0.002	0.0017	0.0045
		ContAware + MTQual	0.0035	0.0027	0.0023	0.004	**0.0019**	0.0016	0.0038
		MTQual	0.0038	0.0025	0.0026	**0.0024**	0.0028	0.0038	0.0062
	xgb	All	0.0058	0.0051	0.0024	0.0078	0.0044	0.0047	0.0049
		All + Kiwi	0.0056	0.0062	0.0034	0.0075	0.0059	0.004	0.0041
		Basic	0.0052	0.0047	0.0039	0.0065	0.0069	0.0054	**0.0033**
		Basic + ContAware	0.0063	0.0054	0.0034	0.0077	0.007	0.0041	0.0042
		Basic + Kiwi	0.0059	0.0063	0.0031	0.008	0.0081	0.0055	0.0041
		Basic + MTQual	0.0076	0.0029	0.0023	0.0049	0.0057	0.0073	0.0063
		ContAware	0.0066	0.0031	0.0038	**0.0022**	**0.0018**	**0.0017**	0.0053
		ContAware-no-ngrams	**0.0031**	0.0018	**0.0015**	0.0037	0.0038	0.0018	0.0058
		ContAware + Kiwi	0.0057	0.0029	0.0028	0.0042	0.0029	**0.0017**	0.0046
		ContAware + MTQual	0.0036	**0.0017**	0.0025	0.0044	0.0019	0.0018	0.0037
		MTQual	0.0072	0.0026	0.0029	0.0027	0.003	0.0047	0.0065
	lin	All	0.0065	0.0107	0.0058	0.0367	0.0369	0.007	0.029
		All + Kiwi	0.0256	0.0147	0.0068	0.0232	0.0152	0.0046	0.0296
		Basic	0.009	0.0094	0.008	0.0085	0.0084	0.0053	**0.0046**
		Basic + ContAware	0.0088	0.0186	0.0084	0.0195	0.0149	0.0048	0.0291
		Basic + Kiwi	**0.0055**	0.0094	0.0074	0.0061	0.0075	0.0054	0.0053
		Basic + MTQual	0.006	0.0157	0.0077	**0.0054**	0.0102	0.0056	0.0053
		ContAware	0.0309	0.0064	0.0036	0.0169	0.0072	0.0115	0.0196
		ContAware-no-ngrams	0.0184	0.003	**0.0031**	0.0068	0.0035	**0.0031**	0.0051
		ContAware + Kiwi	0.0398	**0.0027**	0.0037	0.0292	0.0058	0.0129	0.0215
		ContAware + MTQual	0.0313	0.0158	0.0155	0.0299	0.0087	0.0225	0.0051
		MTQual	0.0057	0.0033	0.0035	0.0166	**0.0033**	0.0051	0.011
CHRF	rf	All	0.4863	0.1696	0.2626	3.5816	0.9808	**0.4593**	1.3687
		All + Kiwi	0.4326	0.1657	0.5441	3.6109	0.8891	0.4603	1.3878
		Basic	0.896	0.1784	**0.2093**	0.7168	0.9245	0.5988	**0.795**
		Basic + ContAware	0.438	0.1715	0.5878	3.9327	1.0351	0.4607	1.3801
		Basic + Kiwi	0.7989	0.1851	0.2564	0.3614	0.785	1.1066	0.8681
		Basic + MTQual	0.8635	**0.1507**	0.2343	**0.293**	0.7631	1.0785	0.8519
		ContAware	0.4072	0.1981	0.568	3.7288	1.0523	0.5115	1.4758
		ContAware-no-ngrams	**0.2623**	0.1887	0.4377	0.4633	0.6197	0.5532	1.8174
		ContAware + Kiwi	0.4031	0.1985	0.5269	3.4099	0.8166	0.5098	1.5002
		ContAware + MTQual	0.4646	0.1891	0.2425	3.4313	0.855	0.5121	1.4696
		MTQual	0.8978	0.6018	0.6035	0.7731	**0.5792**	0.8224	1.5046
	xgb	All	0.6077	0.1767	0.302	3.6333	1.0299	**0.342**	1.262
		All + Kiwi	0.4028	**0.1723**	0.7083	3.0282	0.8754	0.4205	1.3798
		Basic	0.8756	0.2235	0.2892	0.6274	0.9882	0.6266	0.7277
		Basic + ContAware	0.4289	0.2171	0.631	3.1354	1.1088	0.3717	1.2906
		Basic + Kiwi	0.8052	0.2159	0.2632	**0.3151**	0.8863	1.1387	**0.7084**
		Basic + MTQual	1.3596	0.1861	**0.2598**	0.3397	0.8916	0.884	0.7404
		ContAware	**0.3368**	0.2597	0.7216	3.1658	1.309	0.5491	1.2696
		ContAware-no-ngrams	0.425	0.2067	0.4125	0.6174	0.6555	0.5567	1.6852
		ContAware + Kiwi	0.3968	0.2253	0.557	3.2444	1.2608	0.5138	1.2617
		ContAware + MTQual	0.7507	0.2322	0.2988	3.1005	1.101	0.4851	1.3469
		MTQual	1.4985	0.6052	0.6912	0.6981	**0.6349**	0.7797	1.5575
	lin	All	1.7703	0.8641	1.0688	2.2178	4.6592	1.547	3.2258
		All + Kiwi	4.2076	1.2048	0.9176	2.612	0.7953	1.648	1.5182
		Basic	**0.6833**	**0.4498**	0.4437	**0.3906**	0.5778	0.4688	1.4436
		Basic + ContAware	3.1302	1.6709	0.7315	2.2038	0.8389	1.4856	1.4308
		Basic + Kiwi	2.4533	0.4642	**0.4066**	0.6848	**0.498**	0.4464	**1.064**
		Basic + MTQual	0.7655	1.9908	0.7721	2.5733	1.9495	**0.4453**	2.8915
		ContAware	5.8912	0.9877	1.2187	2.9332	0.9726	2.8067	1.3736
		ContAware-no-ngrams	4.2173	0.9912	0.8794	1.965	1.012	0.8848	1.816
		ContAware + Kiwi	5.4418	0.594	1.1102	4.6959	0.7717	3.0684	1.6922
		ContAware + MTQual	2.5498	2.399	3.5792	1.9001	1.4178	4.9324	1.0864
		MTQual	3.1218	1.0199	0.7587	5.2759	1.0109	1.1878	3.1475

Table A.15: pearson correlation coefficient (PCC) of the generic FIPs when each dataset is left-out of the training set, for the en-zh use-case and for the COMET22 and chrF MT metrics.

MT Metric	Pred.	Feature Set	Entertainment	Env.	Finance	Gov.	Health & Wellness	Sports	Travel & Tourism
COMET22	rf	All	57.89	88.98	88.27	92.67	90.68	85.85	77.44
		All + Kiwi	56.89	89.72	90.26	93.01	89.08	85.81	77.31
		Basic	61.39	88.71	86.2	92.55	90.05	86.67	84.25
		Basic + ContAware	60.81	89.7	88.69	91.98	88.66	84.69	75.33
		Basic + Kiwi	60.13	84.84	84.61	92.66	91.31	86.1	83.5
		Basic + MTQual	59.64	83.98	80.96	94.16	90.65	86.15	83.72
		ContAware	67.02	93.82	94.41	76.62	87.22	**86.96**	85.53
		ContAware-no-ngrams	65.02	**95.62**	94.01	**94.44**	**92.25**	86.39	85.87
		ContAware + Kiwi	65.58	94.11	94.27	62.46	91.21	85.42	**86.76**
		ContAware + MTQual	**67.15**	93.1	**94.57**	65.07	91.08	86.16	84.96
		MTQual	13.51	17.06	16.42	11.38	21.25	7.8	9.54
	xgb	All	**64.17**	84.91	80.78	**94.5**	88.98	84.01	73.03
		All + Kiwi	57.06	85.99	79.1	94.3	87.59	84.5	75.78
		Basic	60.2	87.02	83.78	92	89.96	**86.69**	83.47
		Basic + ContAware	58.26	89.26	91.45	94.21	89.04	85	71.21
		Basic + Kiwi	58.55	80.64	88.83	92.78	89.44	85.48	83.14
		Basic + MTQual	50.51	85.19	83.03	94.1	90.24	84.49	81.19
		ContAware	61.96	92.18	90.78	89.19	86.44	83.71	79.51
		ContAware-no-ngrams	61.72	**93.75**	85.52	92.42	89.9	84.6	**84.25**
		ContAware + Kiwi	59.36	92.45	**92.23**	69.37	**90.46**	84.48	78.63
		ContAware + MTQual	56.69	91.77	91.91	51.32	88.06	85.42	83.38
		MTQual	12.33	5.43	16.18	13.47	19.15	12.06	0.63
	lin	All	26.99	45.38	40.63	63.9	39.91	73.12	72.29
		All + Kiwi	45.86	64.86	64.65	85.23	84.72	74.76	71.37
		Basic	47.56	76.17	75.97	86.79	86.64	84.48	75.39
		Basic + ContAware	38.2	60.9	67.43	82.44	85.02	74.13	69.72
		Basic + Kiwi	49.53	77.42	77.52	87.8	87.3	85.15	76.49
		Basic + MTQual	**61.18**	73.27	84.49	**91.66**	**89.26**	86.5	86.18
		ContAware	33.93	82.94	**88.93**	84.69	86.86	74.45	78.18
		ContAware-no-ngrams	38.41	67.76	47.84	47.91	45.63	30.12	46.84
		ContAware + Kiwi	37.38	**83.15**	88.74	82.27	85.37	72.45	77.37
		ContAware + MTQual	57.88	82.08	88.12	88.17	81	**86.52**	**90.93**
		MTQual	36.81	37.28	30.13	34.01	29.59	31.46	51.28
CHRF	rf	All	87.63	95.97	95.59	75.77	86.35	93.74	92.28
		All + Kiwi	87.7	96.22	94.63	76.08	83.59	93.87	92.04
		Basic	87.13	95.3	94.11	94.33	**95.68**	**95.24**	**93.76**
		Basic + ContAware	87.96	95.97	94.86	78.84	94.85	93.74	92.43
		Basic + Kiwi	**88.62**	94.58	92.93	95.36	95.36	94.09	91.25
		Basic + MTQual	88.08	**96.45**	94	**95.82**	95.56	94.32	73.25
		ContAware	86.92	94.59	91.27	58.31	93.14	92.74	91.06
		ContAware-no-ngrams	86.39	94.83	**96**	95.26	86.9	89.28	88.01
		ContAware + Kiwi	86.49	94.7	90.44	59.92	81.66	92.79	89.54
		ContAware + MTQual	85.83	95.02	94.55	60.22	86.84	93	90.62
		MTQual	19.88	22.28	37.65	19.35	27.28	20.77	17.32
	xgb	All	86.02	95.29	**95.78**	69.76	86.14	94.03	87.47
		All + Kiwi	81.8	**96.02**	94.42	77	85.72	91.84	90.79
		Basic	84.63	94.92	91.7	94.69	**94.27**	94.62	92.78
		Basic + ContAware	84.82	95.93	94.49	75.74	91.66	92.22	**92.79**
		Basic + Kiwi	**88.68**	93.13	92.94	90.65	93.53	92.26	90.69
		Basic + MTQual	81.62	95.8	93.71	**95.8**	94.06	**95.46**	78.51
		ContAware	83.87	93.44	90.2	52.98	84.07	88.42	82.99
		ContAware-no-ngrams	82.43	93.36	93.48	94.55	85.4	87.93	85.19
		ContAware + Kiwi	82.48	94.49	92.2	51.01	70.77	88.55	87.75
		ContAware + MTQual	83.24	94.9	90.5	62.47	86.43	90.43	83.48
		MTQual	10.5	22.5	24.31	19.78	20.09	19.86	6.16
	lin	All	2.25	62.02	64.72	82.57	35.57	74.46	**80.01**
		All + Kiwi	46.08	**86.06**	91.98	94.78	93.91	54.23	73.38
		Basic	71.65	70.02	78.97	89.01	82.83	**85.58**	79.64
		Basic + ContAware	45.28	84.13	**92.06**	94.2	**94.2**	51.17	73.02
		Basic + Kiwi	69.07	68.87	79.89	89.53	82.66	85.08	78.01
		Basic + MTQual	**79.52**	61.39	82.43	**95.62**	85.37	85.18	79.92
		ContAware	-8.65	84.5	86.15	86.91	88.68	38.65	60.93
		ContAware-no-ngrams	52.6	61.83	35.07	44.86	31.24	0.86	49.73
		ContAware + Kiwi	-14.15	83.77	84.15	86.77	86.26	6.74	53.25
		ContAware + MTQual	9.81	81.81	79.43	91.14	82.15	84.33	67.79
		MTQual	34.08	38.45	33.12	37.05	32.95	31.37	34.32

Table A.16: mean absolute error (MAE) of the generic FIPs when each dataset is left-out of the training set, for the en-zh use-case and for the SacreBLEU and COMET22Kiwi metrics.

MT Metric	Pred.	Feature Set	Entertainment	Env.	Finance	Gov.	Health & Wellness	Sports	Travel & Tourism
SacreBleu	rf	All	1.4196	0.7204	1.3441	2.5802	2.4049	**0.3908**	0.49
		All + Kiwi	0.6013	0.8928	1.1191	2.674	0.7504	0.3947	0.691
		Basic	1.447	0.5009	**0.6365**	**0.4364**	0.7165	0.7503	0.5392
		Basic + ContAware	0.5577	0.8352	0.9806	3.6974	0.9437	0.3925	**0.415**
		Basic + Kiwi	1.2902	0.5172	0.651	0.5881	0.8338	2.966	3.6925
		Basic + MTQual	3.553	0.3804	2.1117	0.8623	0.7699	2.6789	3.91
		ContAware	0.3674	0.4819	0.8559	5.4174	1.5684	0.5556	0.5149
		ContAware-no-ngrams	0.5655	**0.2977**	0.7223	1.2341	1.0838	0.5327	0.5881
		ContAware + Kiwi	**0.3312**	0.4716	0.8251	5.4139	1.1631	0.5511	0.5103
		ContAware + MTQual	1.3764	0.3269	0.9302	5.3578	2.0429	0.5464	0.5192
		MTQual	2.2232	0.4328	0.7192	0.787	**0.4438**	0.512	1.1132
	xgb	All	1.8858	0.6422	1.095	3.0708	2.3654	**0.3699**	0.5664
		All + Kiwi	1.1003	0.7991	1.1866	2.91	0.7985	0.3855	0.4646
		Basic	1.3824	0.4832	0.6571	**0.4431**	0.648	0.6989	0.4863
		Basic + ContAware	0.7195	0.8259	1.0739	3.3794	1.1885	0.4008	**0.411**
		Basic + Kiwi	1.7436	0.4796	**0.6469**	0.5584	1.1228	2.077	3.4383
		Basic + MTQual	2.9448	0.5843	1.988	0.8114	1.1153	2.8788	3.6937
		ContAware	0.3024	0.4744	0.8831	4.2216	1.3168	0.6063	0.7309
		ContAware-no-ngrams	0.6936	**0.3219**	0.8205	1.3188	1.3308	0.6059	0.6892
		ContAware + Kiwi	**0.2976**	0.5477	0.8426	4.7363	1.2136	0.5935	0.5227
		ContAware + MTQual	1.7673	0.3775	0.8416	4.489	2.0621	0.647	0.7139
		MTQual	2.8446	0.4396	0.7602	0.7944	**0.5716**	0.6743	1.6826
	lin	All	2.2983	1.028	2.3787	3.4833	5.7264	1.3493	7.0132
		All + Kiwi	5.8963	4.3728	8.3118	1.9794	5.3198	6.1651	3.1335
		Basic	2.7373	3.7887	3.0547	2.3415	2.9987	2.6858	3.3378
		Basic + ContAware	5.7692	4.0744	7.8768	2.06	5.1542	5.793	3.0997
		Basic + Kiwi	2.0295	4.1913	3.3211	2.9278	3.2856	2.8881	3.0178
		Basic + MTQual	3.346	1.1886	0.8205	**1.0541**	2.6599	**0.6939**	**0.9518**
		ContAware	3.5551	1.1132	3.5902	3.004	1.0769	2.0997	1.7738
		ContAware-no-ngrams	2.1056	1.734	**0.787**	2.5349	2.4592	0.8092	1.5936
		ContAware + Kiwi	1.8786	0.9139	3.7156	4.0674	**0.9638**	2.3554	2.1832
		ContAware + MTQual	4.6684	2.0505	3.9047	4.1426	3.5366	2.4456	5.5763
		MTQual	**1.0105**	**0.7258**	1.0353	2.5415	2.2811	0.984	2.1623
COMET22Kiwi	rf	All	0.0027	0.0023	0.0027	0.0021	0.0056	0.002	0.002
		All + Kiwi	0.0025	**0.0007**	0.0025	0.0033	0.0047	0.0028	0.0021
		Basic	0.0018	0.0018	**0.0011**	**0.0008**	**0.001**	**0.0015**	0.0019
		Basic + ContAware	0.0018	0.0014	0.0026	0.0023	0.0014	0.0022	0.0032
		Basic + Kiwi	0.0019	0.0032	0.0027	0.0036	0.0058	0.0026	**0.0018**
		Basic + MTQual	0.0019	0.0013	0.003	0.0018	0.006	0.0024	0.0026
		ContAware	0.002	0.0012	0.0034	0.0028	0.0019	0.0022	0.0022
		ContAware-no-ngrams	**0.0017**	0.0011	0.0018	0.0022	0.0018	0.002	0.0019
		ContAware + Kiwi	0.0038	0.0009	0.0036	0.0036	0.0052	0.0023	0.0027
		ContAware + MTQual	0.0028	0.0009	0.0037	0.0033	0.005	0.0022	0.0019
		MTQual	0.002	0.0012	0.0028	0.0028	0.0058	0.0027	0.0023
	xgb	All	0.0037	0.0015	0.0022	0.0028	0.0045	0.0026	**0.0018**
		All + Kiwi	0.0033	0.0017	0.0026	0.0036	0.003	0.0038	0.0019
		Basic	**0.0017**	0.0019	**0.0009**	**0.0009**	**0.0006**	**0.0013**	0.0019
		Basic + ContAware	**0.0017**	0.0012	0.0025	0.0014	0.001	0.0019	0.0028
		Basic + Kiwi	0.0018	**0.001**	0.0025	0.0032	0.0046	0.0028	**0.0018**
		Basic + MTQual	0.0022	0.0011	0.0025	0.0024	0.0047	0.0028	0.0021
		ContAware	0.0018	0.0015	0.0018	0.0019	0.001	0.0015	0.0021
		ContAware-no-ngrams	0.002	**0.001**	0.0017	0.0021	0.0015	0.0018	0.0019
		ContAware + Kiwi	0.0043	0.0014	0.0033	0.0038	0.0041	0.0035	0.0023
		ContAware + MTQual	0.0039	0.0011	0.0034	0.0036	0.0044	0.0031	0.0024
		MTQual	0.0019	0.0012	0.0024	0.0027	0.0045	0.0038	0.0021
	lin	All	0.0034	0.0048	0.004	0.0065	0.0049	0.0028	0.0035
		All + Kiwi	0.0076	0.0035	0.0068	0.007	0.0043	0.0055	0.0025
		Basic	0.0031	0.0017	0.0019	0.0022	**0.0015**	0.0022	0.0021
		Basic + ContAware	0.0096	0.0034	0.0071	0.0058	0.0038	0.0049	0.0026
		Basic + Kiwi	0.0035	0.0025	0.0024	**0.0018**	0.0017	0.0032	0.0041
		Basic + MTQual	0.0039	0.0019	**0.0017**	0.0046	0.002	**0.0018**	**0.002**
		ContAware	0.0038	**0.0014**	0.0049	0.0062	0.0022	0.0034	**0.002**
		ContAware-no-ngrams	**0.002**	0.0017	0.0031	0.006	0.0021	0.003	0.0024
		ContAware + Kiwi	0.0027	0.0016	0.0042	0.0075	0.0023	0.0035	0.0024
		ContAware + MTQual	0.0024	0.0088	0.0055	0.0034	0.0017	0.0054	0.0073
		MTQual	0.0037	0.0035	0.0038	0.0033	0.0054	0.0045	0.0039

Table A.17: pearson correlation coefficient (PCC) of the generic FIPs when each dataset is left-out of the training set, for the en-zh use-case and for the SacreBLEU and COMET22Kiwi metrics.

MT Metric	Pred.	Feature Set	Entertainment	Env.	Finance	Gov.	Health & Wellness	Sports	Travel & Tourism
SacreBleu	rf	All	**35.99**	73.74	70.42	82.88	75.65	80.53	78.09
		All + Kiwi	-14.16	75.25	70.1	83.6	64.07	**81.26**	**86.97**
		Basic	-10.32	73.77	68.69	81.53	72.39	76.69	82.87
		Basic + ContAware	-13.97	73.77	69.04	83.58	64.92	81.05	77.75
		Basic + Kiwi	-12.5	66.45	69.07	79.76	73.16	78.8	82.64
		Basic + MTQual	-14.97	**78.27**	69.84	-9.41	75.51	76.57	83.11
		ContAware	-8.84	69.52	69.1	58.66	77.05	57.41	64.44
		ContAware-no-ngrams	-12.66	76.87	55.98	**86.24**	57.7	51.75	37.36
		ContAware + Kiwi	-4.66	67.88	69.29	58.51	77.58	55.66	61.03
		ContAware + MTQual	1.04	68.3	**70.79**	58.49	**77.86**	56.49	63.87
		MTQual	2.39	46.98	44.78	-0.09	27.36	26.35	26.89
	xgb	All	**33.04**	**76.93**	71.04	82.66	74.58	75.61	72.74
		All + Kiwi	-17.09	74.92	71.34	84.14	70.7	75.76	74.51
		Basic	-10.23	76.32	67.43	77.45	71.42	74.81	79.95
		Basic + ContAware	-15.83	70.08	69.09	82.76	65.3	75.14	**85.82**
		Basic + Kiwi	-10.78	71.02	67.66	73.03	74.44	79.14	83.6
		Basic + MTQual	-10.94	75.05	**73.5**	-5.19	76.29	**80.89**	83.99
		ContAware	-5	62.72	63.88	57.58	74.74	31.74	15.87
		ContAware-no-ngrams	-11.21	68.03	44.46	**84.43**	48.34	31.39	20.3
		ContAware + Kiwi	-5.47	65.57	65.14	57.77	75.32	17.46	67.53
		ContAware + MTQual	-10.54	72.27	62.01	55.61	**76.64**	58.47	31.85
		MTQual	-1.99	24.06	20.93	-4.7	14.59	16.06	26.5
	lin	All	8.53	18.3	74.18	83.5	52.92	56.1	58.8
		All + Kiwi	-4.78	72.91	55.5	80.61	48.65	27.15	51.5
		Basic	-12.57	68.19	68.65	85.56	62.72	59.45	63.1
		Basic + ContAware	-5.15	73.87	56.08	82.08	50.11	26.7	53.67
		Basic + Kiwi	-12.42	66.64	67.05	84.49	60.14	56.35	60.21
		Basic + MTQual	-9.22	5.91	80.61	**89.01**	74.42	**76.59**	**75.33**
		ContAware	6.34	76.69	60.94	77.62	65.72	9.72	66.12
		ContAware-no-ngrams	-12.01	56.1	33.46	48.45	30.92	-28.86	50.08
		ContAware + Kiwi	9.06	76.75	59.6	77.57	62.71	-16.48	60.18
		ContAware + MTQual	**26.83**	**81.53**	**84.04**	84.2	**77.61**	45.22	73.01
		MTQual	24.24	30.26	37.96	38.76	30.84	20.07	5.58
COMET22Kiwi	rf	All	1.02	-32.32	-48.28	-32.16	1.85	-0.36	**29.33**
		All + Kiwi	0.74	**37.23**	-58.06	-26.22	3.76	-19.31	-30.91
		Basic	-12.47	-71.32	-58.97	-45.58	-46.69	-24.91	-24.18
		Basic + ContAware	-6.49	-26.91	-47.9	-41.48	-53.71	-23.48	-35.47
		Basic + Kiwi	-3.59	-51.25	-2.91	-52.37	5.2	-16.84	23.82
		Basic + MTQual	-5.24	-49.48	-74.7	-41.34	3.01	-15.56	-40.19
		ContAware	-9.24	15.67	-53.61	-38	-52.1	-24.69	10.63
		ContAware-no-ngrams	**16.04**	-2.14	-41.59	-50.71	-34.59	-22.33	14.47
		ContAware + Kiwi	-12.17	-16.27	-46.61	-34.89	-10.38	-21.5	-23.6
		ContAware + MTQual	-9.74	-40.89	-50.05	-35.92	-29.87	-5.73	8.17
		MTQual	-3.68	-34.16	**0.71**	**-4.37**	**8.82**	**16.92**	-1.2
	xgb	All	2.6	11.77	-34.7	-0.03	-45.89	-10.86	**41.12**
		All + Kiwi	0.2	10.68	-53.11	**11.83**	18.84	-21.95	-21.63
		Basic	-14.14	-64.56	-20.64	-30.49	**43.37**	-21.67	23.8
		Basic + ContAware	16.37	**20.58**	**-16.54**	-34.03	-43.34	-13.81	-10.31
		Basic + Kiwi	-4.03	-19.66	-65.88	-50.45	-41.26	-11.18	22.57
		Basic + MTQual	8.34	-4.82	-72.07	-23.43	-42.01	-8.99	-30.54
		ContAware	9.4	4.21	-31.44	-32.23	-37.33	5.14	8.41
		ContAware-no-ngrams	**17.52**	-1.8	-34.62	-47.66	-43.96	-14.78	16.09
		ContAware + Kiwi	-5.48	-40	-40.91	-27.92	-48.33	-26.69	17.59
		ContAware + MTQual	-0.52	-33.73	-47	-38.5	-46.46	1.15	16.28
		MTQual	-2.24	-8.17	-25.96	-2.62	-41.6	**7.22**	-5.77
	lin	All	-20.48	-43.42	-55.43	-36	-41.52	11.01	20.29
		All + Kiwi	-22.69	-46.4	-64.06	-43.8	-58.26	-7.33	9.35
		Basic	-25.2	-40.98	5.27	-46.86	-33.84	3.07	31.06
		Basic + ContAware	-21.73	-42.69	-61.24	-44.75	-53.48	-6.32	15.28
		Basic + Kiwi	-24.36	-46.3	1.26	-48.94	-40.12	-2.62	19.9
		Basic + MTQual	-23.77	-28.68	-7.34	-34.98	-28.22	4.5	**32.35**
		ContAware	-7.95	-18.26	-59.92	-51.44	-51.57	14.02	23.29
		ContAware-no-ngrams	-2.26	-4.56	9.06	-5.5	-0.67	10.45	-3.43
		ContAware + Kiwi	-9.68	-23.22	-64	-47.01	-53.55	9.76	16.94
		ContAware + MTQual	**27.75**	-12.83	-58.87	-37.75	-30.77	**24.58**	-9.97
		MTQual	22.29	**42.65**	**17.99**	**15.11**	**38.49**	-0.59	-12.13

Table A.18: mean absolute error (MAE) of the generic FIPs when each dataset is left-out of the training set, for the en-fr use-case and for the COMET22 and chrF MT metrics.

MT Metric	Pred.	Feature Set	Elitr	Elrc	EuBookshop	Php	Tanzil	Tedx-fr	Tedx-fr_ca
COMET22	rf	All	0.002	0.0034	0.0021	0.0074	0.0024	0.0007	**0.0009**
		All + Kiwi	0.0027	**0.0017**	0.0016	0.006	0.0018	0.0011	0.0011
		Basic	0.0021	0.0018	0.0012	**0.0011**	0.0016	**0.0006**	0.0012
		Basic + ContAware	0.002	0.0019	0.0019	0.0033	0.0017	0.001	0.0011
		Basic + Kiwi	0.0026	**0.0017**	**0.0011**	**0.0011**	**0.0014**	0.0008	0.0011
		Basic + MTQual	0.0022	0.0038	0.0026	0.0014	0.0017	0.0008	**0.0009**
		ContAware	0.0024	0.0023	0.0046	0.0089	0.0019	0.0036	0.0012
		ContAware-no-ngrams	**0.0019**	0.0023	0.0047	0.0095	0.0016	0.0039	0.0012
		ContAware + Kiwi	0.0062	0.0019	0.0012	0.0097	0.0022	0.0014	0.0015
		ContAware + MTQual	0.0025	0.008	0.0031	0.0117	0.0038	0.0007	0.001
		MTQual	0.0031	0.0053	0.0027	0.0034	0.0034	0.0016	0.0015
	xgb	All	0.0019	0.002	0.0058	0.0061	0.0018	**0.0007**	**0.001**
		All + Kiwi	0.002	0.0018	0.0018	0.0051	0.0015	0.0046	0.0013
		Basic	0.0019	**0.0017**	0.0011	0.001	0.0015	**0.0007**	0.0011
		Basic + ContAware	0.0018	0.0019	0.0017	0.003	**0.0014**	0.0011	0.0011
		Basic + Kiwi	0.0028	0.0018	**0.001**	0.001	**0.0014**	0.0009	**0.001**
		Basic + MTQual	0.0024	0.0038	0.0032	**0.0009**	0.0018	0.0009	**0.001**
		ContAware	0.0019	0.0018	0.0024	0.0037	**0.0014**	0.0018	0.0015
		ContAware-no-ngrams	0.002	**0.0017**	0.0024	0.0032	**0.0014**	0.0018	0.0011
		ContAware + Kiwi	**0.0015**	**0.0017**	0.0021	0.004	0.0015	0.0042	0.0014
		ContAware + MTQual	0.0021	0.0035	0.008	0.0091	0.002	0.0009	**0.001**
		MTQual	0.0034	0.0045	0.0054	0.0014	0.0024	**0.0007**	0.0011
	lin	All	0.0087	0.0076	0.0028	0.01	0.0216	0.0034	0.006
		All + Kiwi	**0.0017**	0.003	0.0019	0.0037	**0.0014**	0.0009	0.0053
		Basic	0.0025	**0.0018**	0.0011	**0.001**	0.0015	0.0007	**0.0012**
		Basic + ContAware	0.0022	0.0028	0.002	0.0026	0.0031	0.0011	0.0051
		Basic + Kiwi	0.0044	**0.0018**	0.0012	**0.001**	0.0037	0.0007	**0.0012**
		Basic + MTQual	0.0035	0.003	0.0026	0.002	0.0037	0.0046	0.0016
		ContAware	0.0024	0.003	0.0012	0.0014	0.0032	0.0008	0.0037
		ContAware-no-ngrams	0.0018	0.0028	**0.001**	0.0017	0.0033	**0.0006**	0.0037
		ContAware + Kiwi	0.0018	0.0032	0.0012	0.0018	**0.0014**	0.001	0.004
		ContAware + MTQual	0.0081	0.0071	0.0034	0.0104	0.0206	0.004	0.0034
		MTQual	0.003	0.0026	0.0026	0.0018	0.0032	0.0052	0.0016
CHRF	rf	All	0.1523	0.3013	0.9624	1.1405	0.2725	0.5578	0.4004
		All + Kiwi	0.2572	0.3506	0.1612	0.7054	0.255	0.3445	0.1407
		Basic	**0.0967**	0.334	0.1429	0.1242	0.237	**0.0968**	0.156
		Basic + ContAware	0.2208	0.2891	0.1694	0.4743	**0.228**	0.1092	0.1273
		Basic + Kiwi	0.2102	0.4884	**0.1334**	**0.1208**	0.2834	0.1162	0.1116
		Basic + MTQual	0.2382	0.3064	0.892	0.1315	0.2847	0.446	0.1659
		ContAware	0.1369	0.3549	0.3463	1.1604	0.4685	0.2848	0.1693
		ContAware-no-ngrams	0.1504	0.3908	0.3394	1.1923	0.4673	0.1238	0.2124
		ContAware + Kiwi	0.7496	0.4197	0.2094	1.0876	0.561	0.4198	0.1367
		ContAware + MTQual	0.1969	**0.2239**	0.8046	2.138	0.5865	1.0043	0.1218
		MTQual	0.2744	0.2291	0.9182	0.1288	0.4173	0.2431	**0.1095**
	xgb	All	**0.0879**	0.3074	0.3147	0.3546	0.2813	**0.0656**	0.2805
		All + Kiwi	0.1124	0.3149	0.1356	0.5182	0.2394	0.2097	0.136
		Basic	0.1032	0.3078	0.1311	**0.122**	**0.2299**	0.069	0.1318
		Basic + ContAware	0.1294	0.2883	**0.1303**	0.1314	0.2363	0.0713	0.1368
		Basic + Kiwi	0.1178	0.4696	0.1338	0.1221	0.2652	0.0943	0.1152
		Basic + MTQual	0.1529	0.3038	0.4271	0.1632	0.2862	0.168	0.1688
		ContAware	0.1145	0.3217	0.1566	0.4768	0.2956	0.0957	0.154
		ContAware-no-ngrams	0.1125	0.3287	0.1635	0.3443	0.3001	0.1078	0.1578
		ContAware + Kiwi	0.3262	0.3115	0.1417	0.4375	0.4092	0.3828	0.1413
		ContAware + MTQual	**0.0879**	0.2717	0.559	0.6244	0.539	0.0728	**0.1111**
		MTQual	0.1498	**0.2487**	0.901	0.1413	0.3795	0.1849	0.12
	lin	All	0.2442	0.3192	0.2666	0.3092	0.716	1.1444	1.2397
		All + Kiwi	0.3957	0.2294	0.3917	0.1887	1.3303	0.1232	0.63
		Basic	0.1264	0.2723	**0.1321**	**0.1208**	0.2693	**0.079**	**0.1121**
		Basic + ContAware	0.8395	0.3259	0.5955	0.475	0.4459	0.352	0.3143
		Basic + Kiwi	0.2541	0.3806	0.1404	0.1232	0.3031	0.1409	0.1172
		Basic + MTQual	0.138	0.3943	0.7105	0.9222	0.7665	0.699	0.1198
		ContAware	0.7074	0.2273	0.7169	0.422	0.2451	0.5105	0.1997
		ContAware-no-ngrams	0.5892	**0.2183**	0.9531	0.2147	**0.2232**	0.1565	0.4308
		ContAware + Kiwi	0.2749	0.27	0.3133	0.1828	1.1247	0.1869	0.53
		ContAware + MTQual	0.1928	0.2808	0.3057	0.2615	1.4585	1.1476	1.1595
		MTQual	**0.1065**	0.458	0.7811	0.976	0.2314	0.7755	0.1463

Table A.19: pearson correlation coefficient (PCC) of the generic FIPs when each dataset is left-out of the training set, for the en-fr use-case and for the COMET22 and chrF MT metrics.

MT Metric	Pred.	Feature Set	Elitr	Elrc	EuBookshop	Php	Tanzil	Tedx-fr	Tedx-fr_ca
COMET22	rf	All	15.74	11.8	-14.9	-4.34	-4.58	-5.7	**24.72**
		All + Kiwi	21.66	**15.91**	-9.84	-5.77	-18.72	-6.44	-17.09
		Basic	5.9	12.04	-17.9	-1.84	-22.97	-4.86	-25.49
		Basic + ContAware	-17.61	-21.42	-26.26	-4.13	-20.72	-4.7	5.51
		Basic + Kiwi	18.81	-2.43	2.82	-11.09	**21.16**	-4.8	-17.06
		Basic + MTQual	12.71	11.45	11.27	0.04	9.9	-30.02	21.14
		ContAware	-30.47	-22.79	-5.02	-6.92	-1.11	**4.53**	9.05
		ContAware-no-ngrams	-11.04	-8.87	-4.24	-2.3	0.02	3.99	7.39
		ContAware + Kiwi	-0.77	14.52	**21.03**	6.31	4.33	3.98	-29.7
		ContAware + MTQual	2.01	-1.69	2.42	-4.44	9.97	-3.38	3.78
		MTQual	**25.56**	-9.93	11.45	**7.64**	6.8	-29.94	6.6
	xgb	All	nan	**11.41**	-7.29	-10.17	-4.82	-0.67	**20.44**
		All + Kiwi	2.22	-25.24	-13.8	-9.49	-16.03	-8.31	-1.46
		Basic	10.21	10.46	-7.84	-2.41	-19.73	-4.51	-24.92
		Basic + ContAware	-12.09	-19.32	-23.41	-5.84	7.08	4.12	16.05
		Basic + Kiwi	**23.65**	2.94	13.52	-13.15	**15.49**	-6.71	-14.22
		Basic + MTQual	22.81	10.33	-13.42	**8.25**	4.79	**22.19**	5.94
		ContAware	-14.4	-26.65	-6.09	-6.63	2.23	-3.27	11.58
		ContAware-no-ngrams	2.49	-1.88	6.34	3.07	12.5	-22.93	11.09
		ContAware + Kiwi	nan	nan	**22.4**	-3.9	12.31	-8.02	-8.97
		ContAware + MTQual	-10.94	nan	3.2	-4.32	2.36	-10.35	10.24
		MTQual	22.94	9.69	3.37	-1.81	nan	-15.22	10.73
	lin	All	30.42	37.48	-7.83	**27.74**	40.4	19.49	15.2
		All + Kiwi	14.35	-3.53	-4.21	19.39	12.82	14.33	-9.99
		Basic	8.94	-1.48	-14.69	14.96	-4.96	6.58	-23.28
		Basic + ContAware	14.76	-5.13	-4.68	19.87	3.04	8.71	-11
		Basic + Kiwi	12.24	-3.32	-18.02	9.36	11.9	5.53	-25.31
		Basic + MTQual	37.58	24.51	-2.86	21.62	32.36	12.43	-6.66
		ContAware	10.29	-8.77	11.82	15.99	0.36	8.58	12.3
		ContAware-no-ngrams	-13.32	15.69	17.77	15.95	5.98	**23.83**	4.25
		ContAware + Kiwi	7.82	-3.56	12.29	14.74	16.05	15.04	14.98
		ContAware + MTQual	21.79	**48.79**	7.62	15.18	**40.75**	21.47	**38.24**
		MTQual	**49.52**	44.05	**23.48**	23.7	39.09	10.59	28.19
CHRF	rf	All	-17.68	-15.2	12.06	-7.03	5.8	8.32	**23.83**
		All + Kiwi	-42.51	-29.78	7.22	-11.78	4.56	-5.45	-1.94
		Basic	-37.04	-5.66	1.79	-1.02	-3.85	-1.36	-16.31
		Basic + ContAware	**5.43**	-35.35	3.4	-31.78	13.6	-3.93	2.81
		Basic + Kiwi	-42.01	-32.12	-6.39	**31.98**	**18.72**	-9.27	-0.12
		Basic + MTQual	-41.44	-17.96	7.26	-36.38	1.04	0.17	7.5
		ContAware	-3.43	**25.69**	**19.03**	-9.89	0.2	**13.34**	-2.67
		ContAware-no-ngrams	-6.46	20.79	-2.64	25.35	-1.72	-14.06	-3
		ContAware + Kiwi	-11.02	-7.1	-7.57	14.29	0.61	11.01	-1.06
		ContAware + MTQual	-5.41	-5.12	-8.75	8.21	-7.52	12.81	-1.77
		MTQual	-8.44	-3.69	10.69	-11.68	3.49	-1.52	-0.85
	xgb	All	-7.73	-23.59	13.73	-21.75	7.37	nan	22.8
		All + Kiwi	-21.58	-31.2	15.86	-19.42	2.77	-13.47	-17.65
		Basic	-5.36	nan	-0.5	nan	-2.3	2.27	-4.07
		Basic + ContAware	11.95	5.41	16.94	-13.62	9.7	-6.47	-15.99
		Basic + Kiwi	-37.24	-31.95	-7.72	**38.76**	**19.72**	-2.61	-25.36
		Basic + MTQual	**21.75**	-30.34	11.64	-13.54	-9.51	7.75	11.15
		ContAware	0.66	8.57	**34.26**	-20.03	-4.62	**23.1**	-14.01
		ContAware-no-ngrams	-5.82	nan	-35.62	31.69	-3.59	0.85	**25.47**
		ContAware + Kiwi	-8.02	-29.56	8.86	0.6	nan	3.79	-2.84
		ContAware + MTQual	-7.73	**13.37**	19.18	-27.56	2.38	6.66	nan
		MTQual	-2.75	-6.31	28.7	-13.23	4.09	-1.62	nan
	lin	All	-23.21	-0.59	-26.19	1.47	13.14	11.71	**17.68**
		All + Kiwi	-30.74	-10.26	-14.33	-11.46	13.41	11.02	4.91
		Basic	-32.54	-28.7	4.95	15.18	-5.77	2.38	12.7
		Basic + ContAware	-23.41	14.14	1	-11.66	7.99	0.27	5.61
		Basic + Kiwi	-37.55	-31.02	**6.28**	6.42	-9.05	1.34	17.42
		Basic + MTQual	-48.85	-26.72	-12.18	8.8	-14.85	2.97	-0.38
		ContAware	-7.87	16.72	-3.08	-10.99	8.76	-0.52	-3.62
		ContAware-no-ngrams	**14.19**	**24.94**	-41.13	**21.89**	**18.54**	7.02	0.41
		ContAware + Kiwi	11.68	-17.56	-21.04	-10.91	14.7	**13.94**	-0.67
		ContAware + MTQual	-9.89	1.9	-30.59	-2.21	18.29	11.19	9.67
		MTQual	-40.77	8.98	-29.19	-2.3	-13.52	-5.15	-26.62

Table A.20: mean absolute error (MAE) of the generic FIPs when each dataset is left-out of the training set, for the en-fr use-case and for the SacreBleu and COMET22Kiwi MT metrics.

MT Metric	Pred.	Feature Set	Elitr	Elrc	EuBookshop	Php	Tanzil	Tedx-fr	Tedx-fr_ca
SacreBleu	rf	All	0.4161	0.3792	0.3658	0.959	0.3344	0.1409	0.711
		All + Kiwi	0.1937	0.4387	0.234	0.2889	0.2547	0.1821	0.1939
		Basic	0.1629	0.4141	**0.2243**	0.1894	**0.2065**	0.1137	0.1881
		Basic + ContAware	0.1966	0.4249	0.2412	0.3897	0.2813	0.1755	0.226
		Basic + Kiwi	0.2164	0.4666	0.2362	**0.1869**	0.2274	0.1168	**0.1871**
		Basic + MTQual	0.5303	0.3788	0.27	0.8106	0.3742	**0.1094**	0.2383
		ContAware	0.1898	0.5893	0.743	0.8517	0.5394	0.4778	0.2832
		ContAware-no-ngrams	0.2153	0.6019	0.6961	0.8398	0.4969	0.5589	0.2977
		ContAware + Kiwi	1.6878	0.5476	0.2772	0.7083	0.6768	0.2554	0.3269
		ContAware + MTQual	**0.1556**	**0.3383**	0.8511	1.0385	0.9607	0.293	1.0565
		MTQual	0.4473	0.3386	0.2374	0.7007	0.8466	0.1222	0.2358
	xgb	All	0.2575	0.3901	0.3658	0.2609	0.5312	0.1427	0.5279
		All + Kiwi	0.1455	0.4858	0.2303	0.2094	0.2769	0.1487	0.1994
		Basic	**0.1329**	0.4042	0.2255	0.1927	**0.2222**	**0.1104**	**0.1843**
		Basic + ContAware	0.1888	0.4838	0.2514	**0.1898**	0.2892	0.134	0.22
		Basic + Kiwi	0.1654	0.4654	0.266	0.1924	0.2422	0.1673	0.2006
		Basic + MTQual	0.1846	0.3746	0.281	0.3288	0.5592	0.1872	0.2006
		ContAware	0.1403	0.5712	0.4306	0.3231	0.3877	0.1257	0.2901
		ContAware-no-ngrams	0.1591	0.5731	0.3914	0.2318	0.3564	0.1157	0.3332
		ContAware + Kiwi	0.3821	0.5589	0.312	0.2595	0.6211	0.1857	0.3078
		ContAware + MTQual	0.175	**0.3533**	0.7241	0.2496	1.0032	0.1276	0.757
		MTQual	0.2981	0.362	**0.2221**	0.4742	0.728	0.1938	0.2714
	lin	All	0.1558	0.5775	0.4462	2.3257	**0.2388**	1.828	2.8919
		All + Kiwi	0.7259	0.3784	0.9199	0.7411	2.0817	0.1836	1.2491
		Basic	0.2291	**0.3718**	0.2491	**0.1872**	0.2435	**0.1362**	0.2262
		Basic + ContAware	1.0266	0.5501	0.4305	0.5268	0.427	0.4243	0.8895
		Basic + Kiwi	**0.1454**	0.4246	**0.2396**	0.1922	0.3695	0.2442	**0.2082**
		Basic + MTQual	0.4705	0.4984	0.965	1.5737	1.6767	1.3751	0.2698
		ContAware	0.5751	0.3756	0.8969	0.3476	0.2913	0.9285	0.3986
		ContAware-no-ngrams	0.7478	0.4392	1.2247	0.2002	0.3691	0.495	0.7075
		ContAware + Kiwi	0.3502	0.6185	0.5574	0.4294	1.7818	0.5148	0.8705
		ContAware + MTQual	0.5664	0.4058	0.3752	2.2058	1.4915	1.8708	3.6077
		MTQual	0.1648	0.8713	1.3536	1.5139	0.2849	1.3741	0.2536
COMET22Kiwi	rf	All	0.0028	0.0051	0.0015	0.0036	0.0017	0.0012	0.004
		All + Kiwi	0.003	0.0032	**0.0014**	0.0034	0.0013	0.001	0.004
		Basic	0.0031	**0.0027**	0.0015	0.0035	0.0015	**0.0004**	0.001
		Basic + ContAware	0.0027	0.0031	0.0022	0.0031	**0.0012**	**0.0004**	**0.0009**
		Basic + Kiwi	0.0028	0.0062	0.0028	0.0038	0.0013	0.0009	0.0038
		Basic + MTQual	0.0027	0.0072	0.0028	0.004	0.0016	0.0018	0.0042
		ContAware	**0.002**	0.0028	0.0066	**0.003**	0.0014	0.0015	0.0011
		ContAware-no-ngrams	**0.002**	0.0028	0.0067	0.0031	0.0013	0.0012	0.001
		ContAware + Kiwi	0.0097	0.0045	**0.0014**	0.0035	0.0013	0.0014	0.0041
		ContAware + MTQual	0.0092	**0.0027**	0.0015	0.0035	0.0018	0.0013	0.0041
		MTQual	0.0054	**0.0027**	0.0036	0.005	0.0018	0.0018	0.0035
	xgb	All	0.0031	0.0052	0.0015	0.0034	0.0016	0.0021	0.0044
		All + Kiwi	0.003	0.0032	**0.0014**	0.0034	0.0013	0.0017	0.0046
		Basic	0.0031	0.0028	**0.0014**	0.0035	0.0016	**0.0004**	**0.001**
		Basic + ContAware	0.0027	0.0028	0.0023	0.0031	0.0013	0.002	**0.001**
		Basic + Kiwi	0.0031	0.0061	0.0035	0.0041	**0.0012**	0.0018	0.0039
		Basic + MTQual	0.0027	0.007	0.0032	0.0035	0.0014	0.0024	0.0041
		ContAware	**0.0019**	0.0029	0.0042	**0.0029**	0.0033	0.0038	0.0032
		ContAware-no-ngrams	0.002	0.0028	0.0043	**0.0029**	0.0032	0.0033	0.0031
		ContAware + Kiwi	0.0077	0.0038	0.0015	0.0032	**0.0012**	0.0025	0.0042
		ContAware + MTQual	0.0079	**0.0026**	**0.0014**	0.0034	0.0015	0.0019	0.0041
		MTQual	0.006	**0.0026**	0.0032	0.0046	0.0017	0.0011	0.0029
	lin	All	0.0027	**0.0024**	0.0033	0.0101	0.0055	0.0195	0.0174
		All + Kiwi	0.0032	0.003	0.0059	0.0046	0.0095	0.0036	0.0051
		Basic	0.0036	0.0028	0.0017	0.0037	0.0019	**0.0014**	**0.0016**
		Basic + ContAware	0.011	0.0035	0.0104	0.0094	0.0064	0.0027	0.0024
		Basic + Kiwi	**0.0016**	0.0025	**0.0016**	0.0035	0.0057	0.0021	**0.0016**
		Basic + MTQual	0.0049	0.0036	0.0017	**0.0026**	0.0116	0.0131	0.002
		ContAware	0.007	0.003	0.0144	0.0069	0.011	0.0068	0.0052
		ContAware-no-ngrams	0.0042	0.0028	0.0086	0.0068	0.0096	0.0059	0.0067
		ContAware + Kiwi	0.0059	0.0056	0.0038	0.0028	0.0079	**0.0014**	0.0017
		ContAware + MTQual	0.0065	0.0026	0.0074	0.0102	0.0202	0.0217	0.0298
		MTQual	0.002	0.0057	0.0035	0.0029	**0.0013**	0.0131	0.0018

Table A.21: pearson correlation coefficient (PCC) of the generic FIPs when each dataset is left-out of the training set, for the en-fr use-case and for the SacreBleu and COMET22Kiwi MT metrics.

MT Metric	Pred.	Feature Set	Elitr	Elrc	EuBookshop	Php	Tanzil	Tedx-fr	Tedx-fr_ca
SacreBleu	rf	All	-9.73	-17.34	0.6	-4.75	25.21	**24.37**	-8.5
		All + Kiwi	-15.78	-18.73	3.01	2.88	9.37	-6.37	-6.99
		Basic	**23.06**	-32.54	-3.34	9.31	-0.11	1.04	-2.42
		Basic + ContAware	8.62	-27.6	9.31	7.6	4.28	-2.11	-5.46
		Basic + Kiwi	-4	-27.55	-6.92	21.34	8.93	20.26	-27.64
		Basic + MTQual	-8.86	-13.46	-16.84	-5.04	16.1	-4.4	-16.68
		ContAware	14.46	14.17	18.17	-5.01	12.31	-2.24	**9.79**
		ContAware-no-ngrams	-1.03	**16.62**	**21.31**	**26.92**	6.28	11.49	-7.74
		ContAware + Kiwi	-10.97	-22.72	-0.62	2.85	-5.16	18.24	3.65
		ContAware + MTQual	1.23	-3.47	-20.38	3.65	-3.2	16.29	-7.04
		MTQual	0.4	1.76	-12.25	5.99	**31**	10.5	-14.26
	xgb	All	-9.59	-14.11	16.45	-31.04	12.29	24.93	-14.16
		All + Kiwi	-13.35	-1.59	14.7	-6.51	-1.56	10.73	**31.99**
		Basic	nan	nan	nan	-2.26	0.06	-0.33	2.82
		Basic + ContAware	**22.13**	-22.86	16.89	-8.06	-0.47	27.94	1.39
		Basic + Kiwi	17.47	-19.23	-5.57	16.68	9.87	-3.25	-3.37
		Basic + MTQual	-0.48	**19.28**	10.31	1.39	13.48	18.2	-20.57
		ContAware	3.13	-1.2	9.61	-16.39	2.04	15.92	9.6
		ContAware-no-ngrams	-11.35	-7.87	**26.47**	**28.66**	-1.36	20.68	-4.6
		ContAware + Kiwi	nan	-23.67	-24.16	-14.71	-10	-14.15	20.67
		ContAware + MTQual	nan	nan	-4.87	-27.46	12.46	14.42	12.85
		MTQual	-9.29	5.25	19.78	-27.22	**16.63**	**29.46**	0.07
	lin	All	-4.79	-14.11	-14.49	**32.4**	25.97	-15.08	-5.29
		All + Kiwi	21.65	-26.16	2.05	1.96	23.96	**11.38**	11.05
		Basic	21.24	-27.04	-10.94	21.07	3.7	6.19	-13.73
		Basic + ContAware	**26.08**	2.98	5.05	1.37	19.08	10.89	7.27
		Basic + Kiwi	8.89	-27.78	-9.38	14.91	1.74	4.19	-8.66
		Basic + MTQual	-14.57	-31.38	-10.79	26.51	0.53	3.29	-12.94
		ContAware	23.46	**15.55**	**7.67**	-3.5	17.98	8.74	11.98
		ContAware-no-ngrams	24.4	14.53	-22.79	23.4	**32.38**	-5.16	**15.9**
		ContAware + Kiwi	19.88	-21.81	4.73	-2.05	20.64	4.57	14.91
		ContAware + MTQual	8.98	0.58	-4.59	26.7	24.94	-20.65	-1.49
		MTQual	-36.25	0.46	-8.43	11.8	-11.58	-8.21	-7.16
COMET22Kiwi	rf	All	28.27	26.09	-15.39	8.3	4.94	10.73	26.35
		All + Kiwi	**34.14**	26.06	-27.65	-8.12	-3.73	**17.08**	6.02
		Basic	-27.25	-5.26	-4.97	9.79	-17.39	-8.42	2.01
		Basic + ContAware	19.8	-2.18	**16.82**	-0.69	**40.24**	10.88	24.04
		Basic + Kiwi	28.43	25.91	-36.22	9.04	9.69	14.36	**69.33**
		Basic + MTQual	21.49	**27.07**	-22.17	9.91	-3.07	-19.92	46.88
		ContAware	6.08	2.03	-12.35	-4.54	-10.03	-4.73	13.72
		ContAware-no-ngrams	3.98	9.58	-2.03	-8.31	-32.43	-25.05	15.08
		ContAware + Kiwi	-9.61	2.89	3.66	-9.88	-9.74	13.32	11.06
		ContAware + MTQual	-10.51	24.26	-7.42	5.6	6.07	10.23	33.37
		MTQual	8.36	9.81	-38.29	**23.16**	-7.68	-7.27	45.95
	xgb	All	48.06	20.16	-21.07	**18.9**	3.55	-13.7	57.68
		All + Kiwi	**57.1**	18.03	6.25	-4.63	1.48	-16.77	57.78
		Basic	-70.39	-9.77	-4.9	-39.88	-20.48	-11.74	1.67
		Basic + ContAware	1.5	2.98	**27.49**	-2.98	-37.7	19.98	6.14
		Basic + Kiwi	12.06	26.07	-28.4	12.65	3.64	13.66	**69.95**
		Basic + MTQual	14.46	26.58	-4.1	-4.09	-30.38	0.24	67.08
		ContAware	5.58	-3.22	-41.86	-6.08	nan	**22.7**	6.23
		ContAware-no-ngrams	-16.34	-0.08	-1.32	-9.8	nan	-4.8	7.98
		ContAware + Kiwi	-11	1.99	-6.67	-7.37	**22**	-0.21	46.34
		ContAware + MTQual	-16.92	**28.52**	-10.37	-20.08	10.5	-3.1	44.32
		MTQual	17.18	18.85	-16.42	11.42	-1.7	-5.21	44.22
	lin	All	15.5	**38.07**	**30.45**	18.8	15.09	-8.79	6.46
		All + Kiwi	24.34	30.57	-3.18	35.67	3.02	-15.69	-0.38
		Basic	56.33	21.59	20.72	30.87	-16.08	2.55	-3.94
		Basic + ContAware	26.82	13.15	-3.48	39.69	-18.4	-0.57	-3.98
		Basic + Kiwi	**57.89**	27.57	22.91	36.98	-12.12	**3.63**	-3.36
		Basic + MTQual	52.21	35.61	24.07	**41.62**	-8.87	-6.03	5.59
		ContAware	-1.46	2.76	-12.47	23.31	-5.16	-3.54	-1.39
		ContAware-no-ngrams	-5.07	-1.14	20.15	21.94	11.27	-30.79	-4.09
		ContAware + Kiwi	9.35	32.94	-18.92	9.6	28.87	-28.13	7.1
		ContAware + MTQual	1.71	25.71	26.32	-22.23	**32.98**	-18.61	8.37
		MTQual	12.49	34.24	15.46	-3.78	29.78	-2.27	**34.04**

Bibliography

[1] Moloud Abdar, Farhad Pourpanah, Sadiq Hussain, Dana Rezazadegan, Li Liu, Mohammad Ghavamzadeh, Paul Fieguth, Xiaochun Cao, Abbas Khosravi, U Rajendra Acharya, et al. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information fusion*, 76:243–297, 2021. 7.3.1

[2] Ziawasch Abedjan et al. Detecting data errors: Where are we and what needs to be done? *Procs. of VLDB*, 9(12), 2016. 4.3

[3] Milind Agarwal, Sweta Agrawal, Antonios Anastasopoulos, Luisa Bentivogli, Ondřej Bojar, Claudia Borg, Marine Carpuat, Roldano Cattoni, Mauro Cettolo, Mingda Chen, William Chen, Khalid Choukri, Alexandra Chronopoulou, Anna Currey, Thierry Declerck, Qianqian Dong, Kevin Duh, Yannick Estève, Marcello Federico, Souhir Gahbiche, Barry Haddow, Benjamin Hsu, Phu Mon Htut, Hirofumi Inaguma, Dávid Javorský, John Judge, Yasumasa Kano, Tom Ko, Rishu Kumar, Pengwei Li, Xutai Ma, Prashant Mathur, Evgeny Matusov, Paul McNamee, John P. McCrae, Kenton Murray, Maria Nadejde, Satoshi Nakamura, Matteo Negri, Ha Nguyen, Jan Niehues, Xing Niu, Atul Kr. Ojha, John E. Ortega, Proyag Pal, Juan Pino, Lonneke van der Plas, Peter Polák, Elijah Rippeth, Elizabeth Salesky, Jiatong Shi, Matthias Sperber, Sebastian Stüker, Katsuhito Sudoh, Yun Tang, Brian Thompson, Kevin Tran, Marco Turchi, Alex Waibel, Mingxuan Wang, Shinji Watanabe, and Rodolfo Zevallos. FINDINGS OF THE IWSLT 2023 EVALUATION CAMPAIGN. In Elizabeth Salesky, Marcello Federico, and Marine Carpuat, editors, *Proceedings of the 20th International Conference on Spoken Language Translation (IWSLT 2023)*, pages 1–61, July 2023. 6.3

[4] Pangkuh Ajisoko. The use of duolingo apps to improve english vocabulary learning. *International Journal of Emerging Technologies in Learning (iJET)*, 15(7):149–155, 2020. 6

[5] Khetam Al Sharou and Lucia Specia. A taxonomy and study of critical errors in machine translation. In *Proceedings of the 23rd annual conference of the European Association for Machine Translation*, pages 171–180, 2022. 1

[6] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 469–482, 2017. 3, 3, 4.1, 4.3

[7] Lasse F Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. *arXiv preprint arXiv:2007.03051*, 2020. 2.3.5

[8] David Aparício et al. Arms: Automated rules management system for fraud detection. *arXiv preprint arXiv:2002.06075*, 2020. 4.2.1, 4.2.2, 5, 5.2, 6.2.1

[9] Language Technology Research Group at the University of Helsinki. Huggingface helsinki-nlp/opus-mt-en-zh, 2020. URL `https://huggingface.co/Helsinki-NLP/opus-mt-en-zh`. Accessed: 2024-12-11. 6.3

[10] Claudine Badue, Rânik Guidolini, et al. Self-driving cars: A survey. *Expert Systems with Applications*, 165:113816, 2021. ISSN 0957-4174. 4

[11] Ezio Bartocci et al. Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications. In *Lectures on Runtime Verification*. Springer, 2018. 4.3

[12] Barış Bayram, Bilge Köroğlu, and Mehmet Gönen. Improving fraud detection and concept drift adaptation in credit card transactions using incremental gradient boosting trees. In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 545–550, 2020. doi: 10.1109/ICMLA51294.2020.00091. 2.3.3

[13] Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*, pages 610–623, 2021. 2.3.5

[14] Lucas Bernardi, Themistoklis Mavridis, and Pablo Estevez. 150 successful machine learning models: 6 lessons learned at booking. com. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1743–1751, 2019. 2.3.1

[15] Hiya Bhatt, Shrikara Arun, Adyansh Kakran, and Karthik Vaidhyanathan. Towards architecting sustainable mlops: A self-adaptation approach. *arXiv preprint arXiv:2404.04572*, 2024. 2.3.5

[16] Christopher M Bishop. Pattern recognition and machine learning. *Springer google schola*, 2:1122–1128, 2006. 5.5.1

[17] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738. 5.3

[18] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016. 1

[19] Damien Bouchabou, Sao Mai Nguyen, Christophe Lohr, Benoit LeDuc, and Ioannis Kanellos. A survey of human activity recognition in smart homes based on iot sensors

algorithms: Taxonomies, challenges, and opportunities with deep learning. *Sensors*, 21(18), 2021. 2.2.1, 4

[20] Lucas Bourtoule, Varun Chandrasekaran, Christopher A Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 141–159. IEEE, 2021. 1, 2.3.3

[21] Bernardo Branco et al. Interleaved sequence rnns for fraud detection. In *Procs. of KDD*, 2020. 1, 4, 4.2.1, 4.2.3

[22] Leo Breiman. Bagging predictors. *Machine learning*, 24:123–140, 1996. 4.3

[23] Leo Breiman. *Classification and regression trees*. Routledge, 2017. 5.5.1

[24] Tomáš Bureš. Self-adaptation 2.0. In *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 262–263. IEEE, 2021. 2.3.1

[25] Radu Calinescu, Raffaela Mirandola, Diego Perez-Palacin, and Danny Weyns. Understanding uncertainty in self-adaptive systems. In *2020 ieee international conference on autonomic computing and self-organizing systems (ACSOS)*, pages 242–251. IEEE, 2020. 7.2.4

[26] Radu Calinescu et al. Synthesis and verification of self-aware computing systems. In *Self-Aware Computing Systems*. Springer, 2017. 1, 2, 2.1

[27] Jose Camacho-Collados and Mohammad Taher Pilehvar. Embeddings in natural language processing. In *Proceedings of the 28th international conference on computational linguistics: tutorial abstracts*, pages 10–15, 2020. 6.3

[28] J. Cámara, W. Peng, D. Garlan, and B. Schmerl. Reasoning about sensing uncertainty and its reduction in decision-making for self-adaptation. *Science of Computer Programming*, 167, 2018. 1, 2, 2.1, 2.1, 5, 5.1, 7.2.3

[29] Yinzhi Cao and Junfeng Yang. Towards making systems forget with machine unlearning. In *Procs. of S&P*. IEEE, 2015. 1.1, 2.3.3, 4, 4.1, 4.1

[30] Yinzhi Cao et al. Efficient repair of polluted machine learning systems via causal unlearning. In *Procs. of Asia CCS*, 2018. 4.1

[31] Maria Casimiro, Diego Didona, Paolo Romano, Luis Rodrigues, Willy Zwaenepoel, and David Garlan. Lynceus: Cost-efficient tuning and provisioning of data analytic jobs. In *Procs. of ICDCS*, 2020. 1, 3, 3, 4, 4.1, 4.3, 5, 5.3, 6.2, 6.2.4

[32] Maria Casimiro, Paolo Romano, David Garlan, Gabriel Moreno, Eunsuk Kang, and Mark Klein. Self-adaptation for machine learning based systems. In *ECSA (Companion)*, 2021. 5.1, 6.2

[33] Maria Casimiro, Paolo Romano, David Garlan, and Luis Rodrigues. Towards a framework for adapting machine learning components. In *2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pages 131–140. IEEE, 2022. 5.5.1

[34] Jiefeng Chen, Frederick Liu, Besim Avci, Xi Wu, Yingyu Liang, and Somesh Jha. Detecting errors and estimating accuracy on unlabeled data with self-training ensembles. *Advances in Neural Information Processing Systems*, 34:14980–14992, 2021. 2.3.2

[35] Min Chen, Zhikun Zhang, Tianhao Wang, Michael Backes, Mathias Humbert, and Yang Zhang. Graph unlearning. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 499–513, 2022. 1, 2.3.3

[36] Tao Chen. All versus one: An empirical comparison on retrained and incremental machine learning for modeling performance of adaptable software. In *Procs. of SEAMS*. IEEE, 2019. 2.3.3, 2.3.4, 4

[37] Tao Chen. Lifelong dynamic optimization for self-adaptive systems: fact or fiction? In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 78–89. IEEE, 2022. 2.3.4

[38] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016. 6.3

[39] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016. 6.2.1

[40] Zhilu Chen and Xinming Huang. End-to-end learning for lane keeping of self-driving cars. In *IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017. 1

[41] Zhiyuan Chen and Bing Liu. Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 12(3):1–207, 2018. 2.2

[42] Betty H. C. Cheng et al. *Software Engineering for Self-Adaptive Systems: A Research Roadmap*, volume 5525, pages 1–26. Springer Berlin Heidelberg, 2009. 2.1

[43] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Procs. of the 1st workshop on deep learning for recommender systems*, 2016. 4

[44] Shang-Wen Cheng, David Garlan, and Bradley Schmerl. Architecture-based self-adaptation in the presence of multiple objectives. In *ICSE 2006 Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, Shanghai, China, 21-22 May 2006. 6.3

[45] A. Christi et al. Evaluating fault localization for resource adaptation via test-based software modification. In *Procs. of QRS*, 2019. 4.3

[46] Matteo Ciniselli, Nathan Cooper, Luca Pascarella, Antonio Mastropaolo, Emad Aghajani, Denys Poshyvanyk, Massimiliano Di Penta, and Gabriele Bavota. An empirical study on the usage of transformer models for code completion. *IEEE Transactions on Software Engineering*, 48(12):4818–4837, 2021. 6

[47] Jürgen Cito, Isil Dillig, Seohyun Kim, Vijayaraghavan Murali, and Satish Chandra. Explaining mispredictions of machine learning models using rule induction. In *Procs. of ESEC/FSE*, 2021. 1, 2.3.2, 4

[48] Edmund M. Clarke, E Allen Emerson, and A Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986. 2.1

[49] Edmund M Clarke, William Klieber, Miloš Nováček, and Paolo Zuliani. Model checking and the state explosion problem. In *LASER Summer School on Software Engineering*, pages 1–30. Springer, 2011. 5.5

[50] André F. Cruz et al. A bandit-based algorithm for fairness-aware hyperparameter optimization. *CoRR*, abs/2010.03665, 2020. 4.2.1

[51] Per-Erik Danielsson. Euclidean distance mapping. *Computer Graphics and image processing*, 14(3):227–248, 1980. 6.3

[52] Rogério De Lemos, Holger Giese, Hausi A Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M Villegas, Thomas Vogel, et al. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*. Springer, 2013. 1, 2.1

[53] Chris Deotte. Ieee-cis fraud detection winner solution, 2019. URL `https://www.kaggle.com/code/cdeotte/xgb-fraud-with-magic-0-9600`. 6.2, 6.2.1

[54] Josu Diaz-De-Arcaya, Ana I Torre-Bastida, Gorka Zárate, Raúl Miñón, and Aitor Almeida. A joint study of the challenges, opportunities, and roadmap of mlops and aiops: A systematic survey. *ACM Computing Surveys*, 56(4):1–30, 2023. 2.3.1

[55] Diego Didona, Francesco Quaglia, Paolo Romano, and Ennio Torre. Enhancing performance prediction robustness by combining analytical modeling and machine learning. In *Proceedings of the 6th ACM/SPEC international conference on performance engineering (ICPE)*, pages 145–156, 2015. 6.2.1

[56] Tom Diethe, Tom Borchert, Eno Thereska, Borja Balle, and Neil Lawrence. Continual learning in practice. *arXiv preprint arXiv:1903.05202*, 2019. 2.3.1

[57] Annette J Dobson and Adrian G Barnett. *An introduction to generalized linear models*. Chapman and Hall/CRC, 2018. 5.5.1

[58] Raghav Donakanti, Prakhar Jain, Shubham Kulkarni, and Karthik Vaidhyanathan. Reimagining self-adaptation in the age of large language models. *arXiv preprint arXiv:2404.09866*, 2024. 2.3.4

[59] ELRC. Elra-w0138: Elrc_1075_euipo - ip case law french-english, 2018. URL `https://opus.nlpl.eu/ELRA-W0138/corpus/version/ELRA-W0138`. 6.11

[60] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019. 4.1

[61] EMNLP. News translation task of the conference on machine translation, 2024. URL `https://www2.statmt.org/wmt24/translation-task.html`. 6.3

[62] Bradley J Erickson, Panagiotis Korfiatis, Zeynettin Akkus, and Timothy L Kline. Machine learning for medical imaging. *Radiographics*, 37(2), 2017. 1, 4, 6

[63] Mohammad Esrafilian-Najafabadi and Fariborz Haghighat. Occupancy-based hvac control systems in buildings: A state-of-the-art review. *Building and Environment*, 197:107810, 2021. 4

[64] Ugo Fiore, Alfredo De Santis, Francesca Perla, Paolo Zanetti, and Francesco Palmieri. Using generative adversarial networks for improving classification effectiveness in credit card fraud detection. *Information Sciences*, 479, 2019. 1

[65] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM CSUR*, 46(4), 2014. 2.2.1

[66] Dashan Gao, Yang Liu, Anbu Huang, Ce Ju, Han Yu, and Qiang Yang. Privacy-preserving heterogeneous federated transfer learning. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 2552–2559. IEEE, 2019. 4.2.3

[67] Saurabh Garg, Sivaraman Balakrishnan, Zachary C Lipton, Behnam Neyshabur, and Hanie Sedghi. Leveraging unlabeled data to predict out-of-distribution performance. *arXiv preprint arXiv:2201.04234*, 2022. 2.3.2, 5.4, 5.4, 7.2.1

[68] David Garlan, S-W Cheng, A-C Huang, Bradley Schmerl, and Peter Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37 (10):46–54, 2004. 1

[69] Jakob Gawlikowski, Cedrique Rovile Njieutcheu Tassi, Mohsin Ali, Jongseok Lee, Matthias Humt, Jianxiang Feng, Anna Kruspe, Rudolph Triebel, Peter Jung, Ribana Roscher, et al. A survey of uncertainty in deep neural networks. *Artificial Intelligence Review*, 56(Suppl 1):1513–1589, 2023. 7.3.1

[70] Ilias Gerostathopoulos, Dominik Skoda, Frantisek Plasil, Tomas Bures, and Alessia Knauss. Architectural homeostasis in self-adaptive software-intensive cyber-physical systems. In *Software Architecture*, ECSA 2016, pages 113–128. Springer International Publishing, November 28–December 2 2016. 2.1

[71] Omid Gheibi and Danny Weyns. Lifelong self-adaptation: Self-adaptation meets lifelong machine learning. In *Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 1–12, 2022. 2.3.4

[72] Omid Gheibi, Danny Weyns, and Federico Quin. Applying machine learning in self-adaptive systems: A systematic literature review. *ACM TAAS*, 15(3), 2021. 1, 2.3.1

[73] github. Github, 2020. URL `https://github.com/`. 7.3.3

[74] Tianyu Gu et al. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7, 2019. 2.2.1, 4

[75] María José Hernández Guerrero. News translation strategies. In *The Routledge handbook of translation and media*, pages 232–249. Routledge, 2021. 6

[76] Devin Guillory, Vaishaal Shankar, Sayna Ebrahimi, Trevor Darrell, and Ludwig Schmidt. Predicting with confidence on unseen distributions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1134–1144, 2021.

2.3.2

[77] Xiao Guo, Zhenjiang Shen, Yajing Zhang, and Teng Wu. Review on the application of artificial intelligence in smart homes. *Smart Cities*, 2(3):402–420, 2019. 4

[78] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6:512–535, 1994. 2.1

[79] Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. Towards the systematic reporting of the energy and carbon footprints of machine learning. *Journal of Machine Learning Research*, 21(248):1–43, 2020. 2.3.5

[80] Sara M Hezavehi, Danny Weyns, Paris Avgeriou, Radu Calinescu, Raffaela Mirandola, and Diego Perez-Palacin. Uncertainty in self-adaptive systems: A research community perspective. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 15(4):1–36, 2021. 7.2.4

[81] Ling Huang et al. Adversarial machine learning. In *Procs. of AISec*, 2011. 2.2.1, 4

[82] Brent Huchuk, Scott Sanner, and William O'Brien. Comparison of machine learning models for occupancy prediction in residential buildings using connected thermostat data. *Building and Environment*, 160:106177, 2019. 4

[83] HuggingFace. Huggingface, 2016. URL `https://huggingface.co/`. 7.3.3

[84] Pooyan Jamshidi, Javier Cámara, Bradley Schmerl, Christian Käestner, and David Garlan. Machine learning meets quantitative planning: Enabling self-adaptation in autonomous robots. In *Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS 2019, pages 39–50. IEEE, May 2019. 1, 2.3.1, 4

[85] Pooyan Jamshidi et al. Transfer learning for improving model predictions in highly configurable software. In *Procs. of SEAMS*, 2017. 1, 2.3.3

[86] Heli Järvenpää, Patricia Lago, Justus Bogner, Grace Lewis, Henry Muccini, and Ipek Ozkaya. A synthesis of green architectural tactics for ml-enabled systems. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Society*, pages 130–141, 2024. 2.3.5

[87] Nan Jiang, Thibaud Lutellier, and Lin Tan. Cure: Code-aware neural machine translation for automatic program repair. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1161–1173. IEEE, 2021. 6

[88] Yiding Jiang, Vaishnavh Nagarajan, Christina Baek, and J Zico Kolter. Assessing generalization of sgd via disagreement. *arXiv preprint arXiv:2106.13799*, 2021. 2.3.2

[89] Dahyun Jung, Sugyeong Eo, and Heui-Seok Lim. Towards precise localization of critical errors in machine translation. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 3000–3012, 2024. 1

[90] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36 (1), 2003. 1, 2.1

[91] Cody Kinneer, Ryan Wagner, Fei Fang, Claire Le Goues, and David Garlan. Modeling observability in adaptive systems to defend against advanced persistent threats. In *Proceedings of the 17th ACM-IEEE International Conference on Formal Methods and Models for System Design*, pages 1–11, 2019. 7.2.3

[92] Tom Kocmi, Eleftherios Avramidis, Rachel Bawden, Ondřej Bojar, Anton Dvorkovich, Christian Federmann, Mark Fishel, Markus Freitag, Thamme Gowda, Roman Grundkiewicz, Barry Haddow, Philipp Koehn, Benjamin Marie, Christof Monz, Makoto Morishita, Kenton Murray, Makoto Nagata, Toshiaki Nakazawa, Martin Popel, Maja Popović, and Mariya Shmatova. Findings of the 2023 conference on machine translation (WMT23): LLMs are here but not quite there yet. In Philipp Koehn, Barry Haddow, Tom Kocmi, and Christof Monz, editors, *Proceedings of the Eighth Conference on Machine Translation*, pages 1–42, December 2023. 6.3

[93] Philipp Koehn, Franz J. Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 127–133, 2003. URL `https://aclanthology.org/N03-1017`. 6.3

[94] Pang Wei Koh et al. Wilds: A benchmark of in-the-wild distribution shifts. In *Procs. of ICML*. PMLR, 2021. 3

[95] Panagiotis Kourouklidis, Dimitris Kolovos, Joost Noppen, and Nicholas Matragkas. A model-driven engineering approach for monitoring machine learning models. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 160–164, 2021. doi: 10.1109/MODELS-C53483.2021.00028. 1, 2.3.2

[96] Christian Krupitzer, Felix Maximilian Roth, Sebastian VanSyckel, Gregor Schiele, and Christian Becker. A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, 17, 2015. 2.1

[97] Shubham Kulkarni, Arya Marda, and Karthik Vaidhyanathan. Towards self-adaptive machine learning-enabled systems through qos-aware model switching. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1721–1725. IEEE, 2023. 2.3.4

[98] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic model checking and autonomy. *Annual Review of Control, Robotics, and Autonomous Systems*, 5, 2022. 2, 2.1, 2.1

[99] Marta Kwiatkowska et al. Prism: Probabilistic symbolic model checker. In *Procs. of TOOLS*, 2002. 2.1, 5, 5.1, 5.3, 5.5

[100] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*, 2019. 2.3.5

[101] Remi Lam and Karen Willcox. Lookahead bayesian optimization with inequality constraints. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. 6.2.4

[102] Michael Austin Langford and Betty H.C. Cheng. "know what you know": Predicting behavior for learning-enabled systems when facing uncertainty. In *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2021. 2.3.2

[103] Michael Austin Langford and Betty H.C. Cheng. "know what you know": Predicting behavior for learning-enabled systems when facing uncertainty. In *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 78–89. IEEE, 2021. 2.3.2, 4

[104] Bertrand Lebichot, Gian Paldino, Wissam Siblini, Liyun He-Guelton, Frédéric Oblé, and Gianluca Bontempi. Incremental learning strategies for credit cards fraud detection. *International Journal of Data Science and Analytics*, 12(2):165–174, 2021. 1, 1.1, 2.3.3

[105] Grace A. Lewis, Ipek Ozkaya, and Xiwei Xu. Software architecture challenges for ml systems. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 634–638. IEEE, 2021. doi: 10.1109/ICSME52107.2021.00071. 2.2

[106] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P Trevino, Jiliang Tang, and Huan Liu. Feature selection: A data perspective. *ACM computing surveys (CSUR)*, 50(6):1–45, 2017. 5.5.1

[107] Nianyu Li, Mingyue Zhang, Jialong Li, Sridhar Adepu, Eunsuk Kang, and Zhi Jin. A game-theoretical self-adaptation framework for securing software-intensive systems. *ACM Transactions on Autonomous and Adaptive Systems*, 19(2):1–49, 2024. 7.2.3

[108] Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory*, 37(1):145–151, 1991. 6.3

[109] Bing Liu. Learning on the job: Online lifelong and continual learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(09):13544–13549, 2020. 2.2, 2.3.1

[110] Yang Liu et al. A secure federated transfer learning framework. *Procs. of IS*, 35(4), 2020. 4.1, 4.2.3

[111] Yvan Lucas and Johannes Jurgovsky. Credit card fraud detection using machine learning: A survey. *CoRR*, abs/2010.06479, 2020. 2.2.1, 4.2.1, 4.2.2, 4.2.2, 6.2, 7.2.1

[112] Xiaoyi Ma. Hong kong news parallel text, 2000. URL `https://catalog.ldc.upenn.edu/LDC2000T46`. 6.3

[113] Piergiuseppe Mallozzi, Patrizio Pelliccione, Alessia Knauss, Christian Berger, and Nassar Mohammadiha. Autonomous vehicles: state of the art, future trends, and challenges. *Automotive Systems and Software Engineering*, pages 347–367, 2019. 4

[114] Rafael G Mantovani, André LD Rossi, Joaquin Vanschoren, Bernd Bischl, and André CPLF Carvalho. To tune or not to tune: recommending when to adjust svm hyper-parameters via meta-learning. In *2015 International joint conference on neural networks (IJCNN)*, pages 1–8. Ieee, 2015. 1, 2.3.3

[115] Rafael Gomes Mantovani, Tomáš Horváth, Ricardo Cerri, Sylvio Barbon Junior, Joaquin Vanschoren, and André Carlos Ponce de Leon Ferreira de Carvalho. An empirical study on hyperparameter tuning of decision trees. *arXiv preprint arXiv:1812.02207*, 2018. 1, 2.3.3

[116] Arya Marda, Shubham Kulkarni, and Karthik Vaidhyanathan. Switch: An exemplar for evaluating self-adaptive ml-enabled systems. In *Proceedings of the 19th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '24, page 143–149, New York, NY, USA, 2024. Association for Computing Machinery. 2.3.4

[117] Market.us. Global ai in fraud detection market report, 2024. URL `https://market.us/report/ai-in-fraud-detection-market/`. 6

[118] Pedro Mendes, Maria Casimiro, Paolo Romano, and David Garlan. Trimtuner: Efficient optimization of machine learning jobs in the cloud via sub-sampling. In *2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 1–8. IEEE, 2020. 4, 4.1

[119] Pedro Mendes, Paolo Romano, and David Garlan. Error-driven uncertainty aware training. In *27th European Conference on Artificial Intelligence*, 19-24 October 2024. 7.3.1

[120] ML Menéndez, JA Pardo, L Pardo, and MC Pardo. The jensen-shannon divergence. *Journal of the Franklin Institute*, 334(2), 1997. 5.3

[121] Jean-Baptiste Michel, Yuan Kui Shen, Aviva Presser Aiden, Adrian Veres, Matthew K Gray, Google Books Team, Joseph P Pickett, Dale Hoiberg, Dan Clancy, Peter Norvig, et al. Quantitative analysis of culture using millions of digitized books. *science*, 331(6014):176–182, 2011. 6.3

[122] Brad Miller et al. Reviewer integration and performance measurement for malware detection. In *Procs. of DIMVA*, 2016. 4, 4.1, 4.1

[123] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. Model cards for model reporting. In *Proceedings of the conference on fairness, accountability, and transparency*, pages 220–229, 2019. 7.3.3

[124] Gabriel A Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. Proactive self-adaptation under uncertainty: A probabilistic model checking approach. In *Proceedings of the 2015 10th joint meeting on foundations of software engineering (FSE)*, pages 1–12, 2015. 5, 5.1, 5.5.1, 6.2.1, 6.2.4, 7.2.4

[125] Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. Proactive self-adaptation under uncertainty: A probabilistic model checking approach. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, page 1–12, August 2015. 2.1, 2.1, 7.2.2

[126] Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. Efficient decision-making under uncertainty for proactive self-adaptation. In *Proceedings of the*

2016 IEEE International Conference on Autonomic Computing, ICAC 2016, pages 147–156, July 2016. 2.1, 2.1, 7.2.2

[127] Gabriel A. Moreno, Javier Cámara, David Garlan, and Mark Klein. Uncertainty reduction in self-adaptive systems. In *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS 2018, page 51–57, May 2018. 2.1, 5, 5.1, 7.2.4

[128] Gabriel A. Moreno et al. Flexible and efficient decision-making for proactive latency-aware self-adaptation. *ACM Trans. Auton. Adapt. Syst.*, 13(1), 2018. 4.3

[129] Jose G Moreno-Torres, Troy Raeder, Rocío Alaiz-Rodríguez, Nitesh V Chawla, and Francisco Herrera. A unifying view on dataset shift in classification. *Pattern recognition*, 45(1):521–530, 2012. 2.2.1, 2.2.1, 2.2.1

[130] Cuong Nguyen, Tal Hassner, Matthias Seeger, and Cedric Archambeau. Leep: A new measure to evaluate transferability of learned representations. In *Procs. of ICML*. PMLR, 2020. 4, 4.1

[131] Jeremy Nixon, Michael W. Dusenberry, Linchuan Zhang, Ghassen Jerfel, and Dustin Tran. Measuring calibration in deep learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019. 7.3.1

[132] OpenAI. Chatgpt, 2022. URL `https://openai.com/chatgpt/`. 6.3

[133] Michael A. Osborne et al. Gaussian processes for global optimization. In *LION*, 2009. 4.3

[134] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022. 7.4

[135] Yaniv Ovadia et al. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In *Procs. of NIPS*, 2019. 2.2.1, 4.3

[136] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE TKDE*, 22 (10), 2009. 4, 4.1, 4.1, 4.2.3

[137] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE TKDE*, 22 (10), 2009. 2.3.3

[138] Ashutosh Pandey, Gabriel A. Moreno, Javier Cámara, and David Garlan. Hybrid planning for decision making in self-adaptive systems. In *Procs. of SASO*, 2016. 4.2.3

[139] Dimitrios Papamartzivanos et al. Introducing deep learning self-adaptive misuse network intrusion detection systems. *IEEE Access*, 7, 2019. 4.3, 6

[140] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011. 6.2.1, 6.3

[141] PHP. Php parallel corpus, 2012. URL https://opus.nlpl.eu/PHP/corpus/version/PHP. 6.11

[142] Fábio Pinto et al. Automatic model monitoring for data streams. *arXiv preprint arXiv:1908.04240*, 2019. 2.2.1, 2.3.2, 4, 4.2.1, 4.3, 4.3, 4.3, 5.2, 5.3

[143] Mark Pluymaekers. How well do real-time machine translation apps perform in practice? insights from a literature review. In *Proceedings of the 23rd Annual Conference of the European Association for Machine Translation*, pages 51–60, 2022. 6

[144] Maja Popović. chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal, September 2015. Association for Computational Linguistics. 6.3, 6.3, 6.3, 6.3, 7.2.1

[145] Matt Post. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Brussels, Belgium, October 2018. Association for Computational Linguistics. 6.3, 6.3, 7.2.1

[146] The Tanzil Project. Collection of quran translations, 2007. URL https://opus.nlpl.eu/Tanzil/corpus/version/Tanzil. 6.11

[147] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014. 2.1

[148] Federico Quin, Danny Weyns, and Omid Gheibi. Reducing large adaptation spaces in self-adaptive systems using classical machine learning. *Journal of Systems and Software*, 190:111341, 2022. 2.3.1

[149] Joaquin Quionero-Candela et al. *Dataset shift in machine learning*. The MIT Press, 2009. 1, 2.2.1, 2.2.1, 3, 4.3, 5.2

[150] Stephan Rabanser et al. Failing loudly: An empirical study of methods for detecting dataset shift. In *Procs. of NIPS*, 2019. 2.2.1, 2.3.2, 4, 4.3

[151] Ricardo Rei, José G. C. de Souza, Duarte Alves, Chrysoula Zerva, Ana C Farinha, Taisiya Glushkova, Alon Lavie, Luisa Coheur, and André F. T. Martins. COMET-22: Unbabel-IST 2022 submission for the metrics shared task. In *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 578–585, Abu Dhabi, United Arab Emirates (Hybrid), December 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.wmt-1.52. 6.3, 6.3, 6.3, 6.3, 7.2.1

[152] Ricardo Rei, Marcos Treviso, Nuno M. Guerreiro, Chrysoula Zerva, Ana C Farinha, Christine Maroti, José G. C. de Souza, Taisiya Glushkova, Duarte Alves, Luisa Coheur, Alon Lavie, and André F. T. Martins. CometKiwi: IST-unbabel 2022 submission for the quality estimation shared task. In *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 634–645, Abu Dhabi, United Arab Emirates (Hybrid), December 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.wmt-1.60. 6.3

[153] Nils Reimers and Iryna Gurevych. Making monolingual sentence embeddings multilingual using knowledge distillation. *CoRR*, abs/2004.09813, 2020. URL https:

//arxiv.org/abs/2004.09813. 6.11

[154] Parker Ridd and Christophe G Giraud-Carrier. Using metalearning to predict when parameter optimization is likely to improve classification accuracy. In *MetaSel@ ECAI*, pages 18–23, 2014. 1, 1.1, 2.3.3

[155] Samantha Sanders and Christophe Giraud-Carrier. Informing the use of hyperparameter optimization through metalearning. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 1051–1056, 2017. doi: 10.1109/ICDM.2017.137. 1, 2.3.3

[156] T. Saputri and S.-W. Lee. The application of machine learning in self-adaptive systems: A systematic literature review. *IEEE Access*, 8, 2020. 1, 2.3.1

[157] Iqbal H Sarker. Machine learning: Algorithms, real-world applications and research directions. *SN computer science*, 2(3):160, 2021. 1

[158] Danielle Saunders. Domain adaptation and multi-domain adaptation for neural machine translation: A survey. *Journal of Artificial Intelligence Research*, 75:351–424, 2022. 6.3

[159] Bradley Schmerl, Javier Cámara, Jeffrey Gennari, David Garlan, Paulo Casanova, Gabriel A. Moreno, Thomas J. Glazier, and Jeffrey M. Barnes. Architecture-based self-protection: Composing and reasoning about denial-of-service mitigations. In *HotSoS 2014: 2014 Symposium and Bootcamp on the Science of Security*, Raleigh, NC, USA, 8-9 April 2014. 6.3

[160] Skipper Seabold and Josef Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010. 6.2.4

[161] Khetam Al Sharou and Lucia Specia. A taxonomy and study of critical errors in machine translation. In Helena Moniz, Lieve Macken, Andrew Rufener, Loïc Barrault, Marta R. Costa-jussà, Christophe Declercq, Maarit Koponen, Ellie Kemp, Spyridon Pilos, Mikel L. Forcada, Carolina Scarton, Joachim Van den Bogaert, Joke Daems, Arda Tezcan, Bram Vanroy, and Margot Fonteyne, editors, *Proceedings of the 23rd Annual Conference of the European Association for Machine Translation*, pages 171–180, June 2022. 1

[162] Yong-Jun Shin, Eunho Cho, and Doo-Hwan Bae. Pasta: An efficient proactive adaptation approach based on statistical model checking for self-adaptive systems. In *Fundamental Approaches to Software Engineering*, pages 292–312, Cham, 2021. Springer International Publishing. 5.5.1

[163] Daniel L Silver, Qiang Yang, and Lianghao Li. Lifelong machine learning systems: Beyond learning algorithms. In *2013 AAAI spring symposium series*, 2013. 4.1

[164] IEEE Computational Intelligence Society. Ieee-cis fraud detection, 2019. URL https://www.kaggle.com/competitions/ieee-fraud-detection/overview. 6.2.1, 6.2.4

[165] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for modern deep learning research. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(09):13693–13696, 2020. 2.3.5

[166] Nico Surantha and Wingky R Wicaksono. Design of smart home security system using object recognition and pir sensor. *Procedia computer science*, 135:465–472, 2018. 4

[167] TED Talks. Ted2020, 2020. URL `https://opus.nlpl.eu/TED2020/corpus/version/TED2020`. 6.11

[168] Tatoeba. Opus-eng-fra mt model, 2021. URL `https://github.com/Helsinki-NLP/Tatoeba-Challenge/blob/master/models/eng-fra/opus-2021-02-22.yml`. 6.3

[169] Niek Tax, Kees Jan de Vries, Mathijs de Jong, Nikoleta Dosoula, Bram van den Akker, Jon Smith, Olivier Thuong, and Lucas Bernardi. Machine learning for fraud detection in e-commerce: A research agenda. In *Deployable Machine Learning for Security Defense: Second International Workshop, MLHat 2021, Virtual Event, August 15, 2021, Proceedings 2*, pages 30–54. Springer, 2021. 7.2.1

[170] Meghana Tedla, Shubham Kulkarni, and Karthik Vaidhyanathan. Ecomls: A self-adaptation approach for architecting green ml-enabled systems. *arXiv preprint arXiv:2404.11411*, 2024. 2.3.5

[171] Jörg Tiedemann. The Tatoeba Translation Challenge – Realistic data sets for low resource and multilingual MT. In *Proceedings of the Fifth Conference on Machine Translation*, pages 1174–1182, Online, November 2020. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/2020.wmt-1.139`. 6.3

[172] Jörg Tiedemann and Santhosh Thottingal. OPUS-MT — Building open translation services for the World. In *Proceedings of the 22nd Annual Conferenec of the European Association for Machine Translation (EAMT)*, Lisbon, Portugal, 2020. 6.3

[173] Jörg Tiedemann. Parallel data, tools and interfaces in opus. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Mehmet Ugur Dogan, Bente Maegaard, Joseph Mariani, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey, may 2012. European Language Resources Association (ELRA). ISBN 978-2-9517408-7-7. 6.3, 6.11

[174] Tilde. Corpus of documents from the eu bookshop, 2012. URL `https://opus.nlpl.eu/EUbookshop/corpus/version/EUbookshop`. 6.11

[175] J. Townsend. Theoretical analysis of an alphabetic confusion matrix. *Perception & Psychophysics*, 9(1), 1971. 5.2

[176] Ngoc Tran, Jean-Guy Schneider, Ingo Weber, and A Kai Qin. Hyper-parameter optimization in classification: To-do or not-to-do. *Pattern Recognition*, 103:107245, 2020. 2.3.3

[177] Michele Tufano, Jevgenija Pantiuchina, Cody Watson, Gabriele Bavota, and Denys Poshyvanyk. On learning meaningful code changes via neural machine translation. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 25–36. IEEE, 2019. 6

[178] Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, and Denys Poshyvanyk. An empirical study on learning bug-fixing patches in the wild via neural machine translation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 28(4):1–29, 2019. 6

[179] Aimee Van Wynsberghe. Sustainable ai: Ai for sustainability and the sustainability of ai. *AI and Ethics*, 1(3):213–218, 2021. 2.3.5

[180] Roberto Verdecchia, Luís Cruz, June Sallou, Michelle Lin, James Wickenden, and Estelle Hotellier. Data-centric green ai an exploratory empirical study. In *2022 international conference on ICT for sustainability (ICT4S)*, pages 35–45. IEEE, 2022. 2.3.5

[181] Kiri L. Wagstaff. Machine learning that matters. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, 2012. 2.3, 2.3.1

[182] Yoav Wald, Amir Feder, Daniel Greenfeld, and Uri Shalit. On calibration and out-of-domain generalization. *Advances in neural information processing systems*, 34: 2215–2227, 2021. 7.3.1

[183] Zijie J Wang, Dongjin Choi, Shenyu Xu, and Diyi Yang. Putting humans in the natural language processing loop: A survey. *arXiv preprint arXiv:2103.04044*, 2021. 4.1

[184] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):1–40, 2016. 2.3.3

[185] Danny Weyns, Bradley Schmerl, Masako Kishida, Alberto Leva, Marin Litoiu, Necmiye Ozay, Colin Paterson, and Kenji Tei. Towards better adaptive systems by combining mape, control theory, and machine learning. In *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 217–223. IEEE, 2021. 2.3.1

[186] Danny Weyns, Omid Gheibi, Federico Quin, and Jeroen Van Der Donckt. Deep learning for effective and efficient reduction of large adaptation spaces in self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 17(1-2): 1–42, 2022. 2.3.1

[187] Philip Williams and Barry Haddow. Elitr eca multilingual corpus, 1995. URL `https://opus.nlpl.eu/ELITR-ECA/corpus/version/ELITR-ECA`. 6.11

[188] Philip Williams and Barry Haddow. The ELITR ECA corpus. *CoRR*, abs/2109.07351, 2021. URL `https://arxiv.org/abs/2109.07351`. 6.11

[189] D. Wu et al. A highly accurate framework for self-labeled semisupervised classification in industrial applications. *IEEE TII*, 14(3), 2018. 2.2.1

[190] Yinjun Wu et al. DeltaGrad: Rapid retraining of machine learning models. In *Procs. of ICML*, 2020. 3, 4, 4.1

[191] Yinjun Wu et al. DeltaGrad: Rapid retraining of machine learning models. In *Procs. of ICML*, 2020. 1.1, 2.3.3

[192] Y. Xiao, I. Beschastnikh, D. S. Rosenblum, C. Sun, S. Elbaum, Y. Lin, and J. S. Dong. Self-checking deep neural networks in deployment. In *Procs. of ICSE*, 2021. 2.3.2

[193] Yan Xiao, Ivan Beschastnikh, David S Rosenblum, Changsheng Sun, Sebastian Elbaum, Yun Lin, and Jin Song Dong. Self-checking deep neural networks in deployment. In *Procs. of ICSE*, 2021. 4

[194] N. Yadwadkar, B. Hariharan, J. Gonzalez, B. Smith, and R. Katz. Selecting the best vm across multiple public clouds: A data-driven performance modeling approach. In *SoCC*, 2017. 1, 4.1, 4.3, 5.3, 6.2

[195] Jingkang Yang, Kaiyang Zhou, Yixuan Li, and Ziwei Liu. Generalized out-of-distribution detection: A survey. *arXiv preprint arXiv:2110.11334*, 2021. 2.2.1, 2.2.1, 2.3.2

[196] Zhou Yang, Muhammad Hilmi Asyrofi, and David Lo. BiasRV: Uncovering biased sentiment predictions at runtime. *CoRR*, abs/2105.14874, 2021. 4.3

[197] Tianqi Zhao, Wei Zhang, Haiyan Zhao, and Zhi Jin. A reinforcement learning-based framework for the generation and evolution of adaptation rules. In *2017 IEEE International Conference on Autonomic Computing (ICAC)*, pages 103–112. IEEE, 2017. 2.3.1

[198] Xianzhe Zhou, Wally Lo Faro, Xiaoying Zhang, and Ravi Santosh Arvapally. A framework to monitor machine learning systems using concept drift detection. In *Business Information Systems: 22nd International Conference, Proceedings, Part I 22*, pages 218–231. Springer, 2019. 4.3, 4.3

[199] Xianzhe Zhou, Wally Lo Faro, Xiaoying Zhang, and Ravi Santosh Arvapally. A framework to monitor machine learning systems using concept drift detection. In *Business Information Systems: 22nd International Conference, BIS 2019, Seville, Spain, June 26–28, 2019, Proceedings, Part I 22*, pages 218–231. Springer, 2019. 2.3.2

[200] Yongchun Zhu, Dongbo Xi, Bowen Song, Fuzhen Zhuang, Shuai Chen, Xi Gu, and Qing He. Modeling users' behavior sequences with hierarchical explainable network for cross-domain fraud detection. In *Proceedings of The Web Conference 2020*, pages 928–938, 2020. 6.2.1, 6.2.4

[201] I. Žliobaitė. Learning under concept drift: an overview. *arXiv preprint arXiv:1010.4784*, 2010. 2.2.1