Characterizing Technical Debt and Antipatterns in AI-Based Systems: A Systematic Mapping Study

Justus Bogner*
University of Stuttgart
Institute of Software Engineering
Stuttgart, Germany
justus.bogner@iste.uni-stuttgart.de

Roberto Verdecchia*
Vrije Universiteit Amsterdam
Department of Computer Science
Amsterdam, The Netherlands
r.verdecchia@vu.nl

Ilias Gerostathopoulos*
Vrije Universiteit Amsterdam
Department of Computer Science
Amsterdam, The Netherlands
i.g.gerostathopoulos@vu.nl

***Abstract*—*Background:* With the rising popularity of Artificial Intelligence (AI), there is a growing need to build large and complex AI-based systems in a cost-effective and manageable way. Like with traditional software, Technical Debt (TD) will emerge naturally over time in these systems, therefore leading to challenges and risks if not managed appropriately. The influence of data science and the stochastic nature of AI-based systems may also lead to new types of TD or antipatterns, which are not yet fully understood by researchers and practitioners. *Objective:* The goal of our study is to provide a clear overview and characterization of the types of TD (both established and new ones) that appear in AI-based systems, as well as the antipatterns and related solutions that have been proposed. *Method:* Following the process of a systematic mapping study, 21 primary studies are identified and analyzed. *Results:* Our results show that (i) established TD types, variations of them, and four new TD types (data, model, configuration, and ethics debt) are present in AI-based systems, (ii) 72 antipatterns are discussed in the literature, the majority related to data and model deficiencies, and (iii) 46 solutions have been proposed, either to address specific TD types, antipatterns, or TD in general. *Conclusions:* Our results can support AI professionals with reasoning about and communicating aspects of TD present in their systems. Additionally, they can serve as a foundation for future research to further our understanding of TD in AI-based systems.**

***Index Terms*—Artificial Intelligence, Machine Learning, Technical Debt, Antipatterns, Systematic Mapping Study**

I. Introduction

Artificial Intelligence (AI) covers different technologies for searching, reasoning, planning, problem solving, and learning with the overall aim of "automating intellectual tasks normally performed by humans" [1]. Its rise in popularity in recent years is mostly due to advancements in Machine Learning (ML), an area of AI focusing on algorithms and systems to identify rules and patterns in data based on statistical modeling techniques.

With more and more companies offering AI-powered products and using AI techniques to improve their internal processes, there is a need to build large, complex AI systems in a cost-effective and manageable way. At the surface level, this may not seem like a new problem: AI systems are software systems too, so we can use well-known, established software engineering principles, practices, and processes to build such systems (e.g., separation of concerns, component-based encapsulation, and agile delivery). However, recent studies show that the AI/ML domain possesses characteristics that make it distinct from other software application domains, such as an increased importance of data quality and management, unclear abstraction boundaries for complex models, and challenges in the customization and reuse of AI/ML components [2-4]. Such characteristics seem to necessitate adaptations of principles, practices, and processes successfully used in other domains, or even the adoption of new, AI-specific ones [5].

A successful practice when building software systems in an iterative fashion is the awareness and management of technical debt (TD) [6]. TD is a metaphor used to describe design or implementation constructs that may be expedient in the short term, but can make future changes more costly or even impossible [7]. Looking at the differences between AI/ML and other application domains from the TD perspective, researchers from Google proposed in 2015 new TD instances that are specific to the development of AI-based systems [4]. Since this seminal paper, various research works from both academia and industry followed up with the documentation of additional TD items and antipatterns in AI/ML systems [8-10].

Despite these efforts, there is still no comprehensive conceptual overview of TD in AI-based systems. It is unclear, for instance, whether these systems accrue more "traditional" TD than other types of systems, such as code, architecture, or documentation debt. It is also important to understand if AI-specific TD types emerge and what their characteristics, associated antipatterns, and proposed solutions are. Gaining such an overview would provide a foundation for future research, and support practitioners to better manage the maintenance and evolution of AI-based systems. To characterize TD in AI-based systems, we therefore conducted a systematic mapping study (SMS), and collected and analyzed relevant papers on the topic. Grounded in 21 primary studies, our contribution with this paper is the thorough analysis and discussion of the concepts of TD and antipatterns in AI-based systems.

II. Background

A. Technical Debt

Since its formulation by Cunningham in 1992 [6], the definition of *technical debt* has continuously evolved and broadened in scope. Nowadays, it encompasses a vast range of concepts, artifacts, and processes [11]. Among the current definitions

*All authors contributed equally to this study.

of TD, a widely adopted one was formulated during Dagstuhl seminar 16162 [7]. Simply referred to as the 16162 definition, it specifies TD as "design or implementation constructs that are expedient in the short term, but set up a technical context that can make a future change more costly or impossible". To structure the knowledge on TD, different *TD types* have been described, e.g. architectural debt, requirements debt, and test debt, allowing researchers and practitioners to effectively focus on specific technical issues where the debt metaphor applies [12]. Instances of such types vary considerably in nature, from suboptimal reuse of architectural components [13], to deferred testing [14], or the involvement of certain development communities [15]. In this research, we embrace the 16162 definition of TD, and build upon the TD type taxonomies presented by Li et al. [12] and Rios et al. [16].

B. Antipatterns

While design patterns constitute proven solution blueprints for specific problems, there is also the inverse concept of *antipatterns*, i.e. frequently occurring suboptimal solutions [17]. Developers may choose these solutions under time pressure, but antipatterns often appear due to insufficient expertise. They can have immediate negative effects on quality attributes such as maintainability, performance efficiency, or reliability, but may also hinder the sustainable evolution of a system, leading to the accumulation of TD. Antipatterns can exist at different levels of abstraction, such as code antipatterns, architectural antipatterns, or even project management antipatterns.

There is also the related concept of *bad smells*, e.g. code smells [18] or architectural smells [19]. Some authors keep these terms strictly separated [20], i.e. software smells are seen as potential indicators of bad quality that may require further investigation, while an antipattern is always supposed to be a bad practice that should be avoided. However, similar to patterns, many antipatterns can also be context-sensitive and may be perceived as "bad" only in specific cases. When collecting archetypes of suboptimal software practices, a clear distinction between the two concepts becomes less important and several studies have handled them uniformly [21,22]. For the purpose of this paper, we therefore do not differentiate between the terms *antipattern* and *smell*, i.e. we collect both concepts under the same umbrella. In this sense, we treat e.g. code smells as antipatterns on the implementation level.

C. Related Work

Technical debt and antipatterns have been the target of numerous reviews in different SE subfields and domains. However, to the best of our knowledge, there exists no comprehensive secondary study focusing on these concepts in the area of AI-based systems. TD has been studied in the context of databases [23] and data-intensive systems [24], but without a clear focus on AI or ML, as with Sculley et al. [4].

Nonetheless, several studies focus on general software quality aspects in AI-based systems. Humbatova et al. [25] conceptualized a fault taxonomy for deep learning systems by analyzing GitHub repositories, StackOverflow posts, and conducting interviews. While faults are not in the scope of our study, they can be related to TD and antipatterns in some cases, e.g. as symptoms of their existence. Concerning the quality assurance of AI-based systems, several position papers discuss differences compared to "traditional" systems and highlight the need for adapted quality assurance techniques [26,27]. Some empirical studies also went further than this and distilled effective techniques for ML system engineering, e.g. Serban et al. [28] derived general AI system development best practices, and Siebert et al. [29] provided guidelines for the quality assurance of such systems. Lastly, the study that comes closest to the goal of our own research is a preliminary multivocal literature review by Washizaki et al. [30]: they collected design patterns and antipatterns for ML systems from both white and grey literature. However, they only identified eight antipatterns, seven from Sculley et al. [4] and one from a company blog post, and did not provide any insights on how TD is characterized in AI-based systems. With our study, we therefore fill this gap by providing a detailed characterization of TD and antipatterns in AI-based systems.

III. Methodology

In this section, we document the research design, which was rigorously adhered to during study execution. We primarily followed the guidelines for conducting systematic literature studies in software engineering research by Kitchenham [31].

A. Research Objective and Questions

The aim of this research is to further the understanding of technical debt and antipatterns in AI-based software systems. To refine this goal, we derived the following research questions (RQs), which guided our mapping study:

RQ1: What are the characteristics of technical debt in AI-based systems?

- **RQ1.1:** Which established types of technical debt have been reported for AI-based systems?
- **RQ1.2:** Does the nature of established technical debt types change in AI-based systems?
- **RQ1.3:** Which new technical debt types have emerged in AI-based systems?
- **RQ1.4:** Which quality attributes are affected by technical debt in AI-based systems?

RQ2: Which antipatterns have been reported for AI-based systems?

RQ3: Which solutions have been reported to address technical debt and antipatterns in AI-based systems?

B. Research Process

An overview of the research process followed is depicted in Fig. 1. The process started with the execution of a conservative automated search query via *Google Scholar*, followed by an iterative forward- and backward-snowballing process, until theoretical saturation was reached. Following the methodology by Wohlin et al. [32], we based our search on a start set obtained via an automated search query executed on Google Scholar. This set was then used for exhaustive bidirectional snowballing. The use of Google Scholar allowed us to avoid

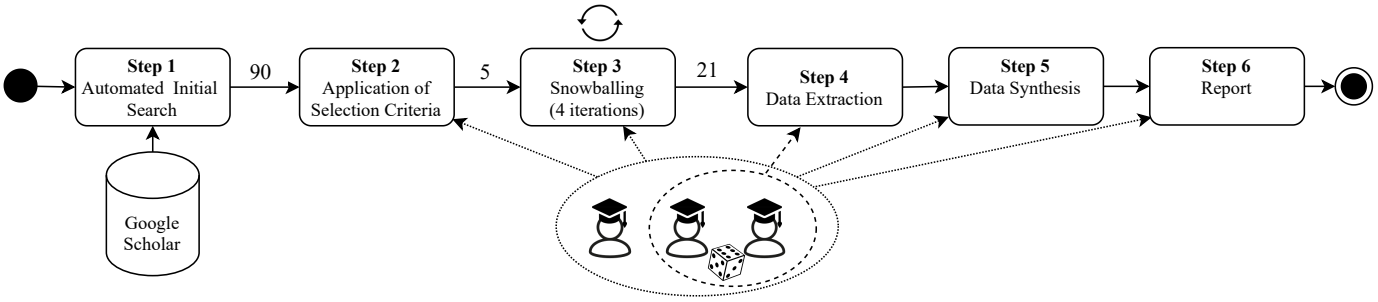

Fig. 1. Systematic mapping study process overview

bias in favor of any specific publisher [32]. Further details of each research step are reported in the following subsections.

1) Step 1: Automated Initial Search

To gather an initial set of potentially relevant studies, we executed a conservative automated query on the *Google Scholar* literature indexer. The title-focused query was designed to encompass related literature focusing specifically on the topic under investigation, and is formulated as follows:

Listing 1. Automated search query

```
1 ALLINTITLE: ("technical debt" OR "antipatterns" OR
2 "antipattern" OR "anti patterns" OR "anti pattern" OR
3 "smell" OR "smells") AND ("artificial intelligence" OR
4 "machine learning" OR "deep learning" OR "intelligent"
5 OR "smart" OR "AI" OR "ML" OR "DL")
```

This query identifies literature containing in their title keywords referring either to "technical debt", "antipatterns", or "smells" (Listing 1, Lines 1-3) and keywords referring to AI, or related synonyms and acronyms (Listing 1, Lines 3-5). The automated query was executed mid-June 2020, and yielded 90 potentially relevant studies. As we were not interested in publications regarding a specific timeframe, the publication date was purposely left unbounded in the query.

2) Step 2: Application of Selection Criteria

After identifying the set of potentially relevant studies via the automated query, we conducted a manual selection process. During this step, we evaluated the initial pool of studies based on pre-defined selection criteria. A paper was selected as a primary study if it satisfied all inclusion criteria and none of the exclusion ones. We used the following criteria:

- I1- Publications reporting technical debt, antipatterns, or suboptimal software engineering practices
- I2- Publications focusing on AI-based systems
- E1- Non-English publications
- E2- Publications for which the full text is not available to us
- E3- Duplicates or extensions of already included publications
- E4- Secondary or tertiary studies
- E5- Publications in the form of editorials, tutorials, books, extended abstracts, etc.
- E6- Non-scientific publications (i.e. grey literature)

The two inclusion criteria (I1, I2) were formulated to ensure that primary studies focused on the investigated topic, namely TD and antipatterns in AI-based systems, and hence provided relevant data to answer our RQs. The exclusion criteria instead were designed to guarantee that data could be extracted from papers (E1, E2), without duplication or redundancy (E3, E4), and consisted of scientific literature (E5, E6).

Given the fast pace at which the investigated topic evolves, we purposely included preprints during the selection process. However, preprints needed to possess a sufficient level of quality (reviewed by all three researchers), to have already been cited by high-quality academic literature, and to be from reputable authors who published other studies in the field.[1]

During the selection, adaptive reading depth [33] was used to efficiently assess potentially relevant studies. To mitigate subjective bias, all authors independently applied the selection criteria for the 90 candidate studies. Differences were jointly discussed until consensus was reached. This led to the selection of five primary studies, i.e. the snowballing start set.

3) Step 3: Snowballing

To obtain a sound and encompassing set of primary studies, the automated search was complemented by recursive backward and forward snowballing [32]. During this step, all studies either citing or cited by the primary studies were examined. Similar as for the initial selection, the snowballing process was conducted by three researchers: in each round, all researchers independently suggested new primary studies to be included, i.e. studies which fulfilled the selection criteria. Divergences were jointly discussed and resolved, after which the next iteration started. Overall, it took four rounds of backward and forward snowballing until no new studies were identified. Snowballing led to the inclusion of 16 studies, i.e. our SMS selection process led to the identification of 21 primary studies.

4) Step 4: Data Extraction

In the next step, we systematically analyzed the primary studies and extracted data related to our RQs. To gain a preliminary understanding, a data extraction pilot with four papers was conducted independently by all researchers. Subsequently, the extracted data was jointly discussed, leading to the extraction framework used in this study. Two researchers were randomly assigned to each primary study. They independently extracted

[1]This design decision led to the inclusion of two additional papers, namely a white paper by Microsoft research [P1], and a paper presented at the *AAAI Fall Symposium Series: Artificial Intelligence in Government and Public Sector* [P2].

the data and agreed on the final extractions per paper in a consensus meeting, with the intervention of the third researcher when required. The extraction framework is divided into one part for each specific RQ of our study (see Section III-A).

To *characterize TD in AI-based systems* (RQ1), data needed to be extracted according to its four sub-questions. For the recurrence of *established TD types in AI-based systems* (RQ1.1), we identified types based on the TD taxonomies of Li et al. [12] and Rios et al. [16] (e.g. code, test, or architectural debt). Using the same taxonomies, we also extracted *variations of established TD types* (RQ1.2), i.e. if the nature or scope of TD types changed in AI-based systems. An example for this is test debt, as it extends in AI-based systems to testing models and data. We also analyzed the primary studies for *new TD types in AI-based systems* (RQ1.3), i.e. debt types which (i) are documented in the context of AI-based systems and (ii) cannot be traced back to established TD types. Lastly, we extracted *quality attributes affected by TD in these systems* (RQ1.4), which was based on ISO/IEC 25010 [34], with the possibility to extend it with additional identified attributes.

To answer RQ2 (*antipatterns*), the primary studies were analyzed for recurrent suboptimal solutions in AI-based systems. Such suboptimal solutions could be explicitly referenced as "antipatterns" (e.g. correction cascades [4]), or reported as root causes of TD (e.g. unstable data dependencies [10]).

Finally, to answer RQ3 (*solutions*), we extracted the solutions proposed to mitigate or resolve TD and antipatterns in AI-based systems. Such solutions could be specific to a certain TD type or antipattern (e.g. model isolation to resolve entanglements) or general best practices to mitigate or prevent the introduction of TD in AI-based systems (e.g. periodically assessing assumptions during ML model evolution).

Note that the extractions did not have to explicitly mention "debt", "antipattern", or a specific quality attribute. The decision if a passage implicitly describing these concepts warranted extraction was up to the researchers' interpretation.

5) Step 5: Data Synthesis

As a final step, the extracted data was harmonized (e.g. merging identical or very similar antipatterns), and then analyzed to derive answers to the research questions. This analysis relied on open coding [35] to systematically identify recurrent concepts. Further axial coding [35] was required to reduce the growing complexity of some emerging concepts (e.g. antipattern subcategories). During the coding, emerging results were continuously discussed among the authors to keep codes and their abstraction level consistent. Finally, summary statistics were created to discuss general findings and their potential implications. For the sake of transparency and reproducibility, we make all study artifacts publicly available online[2].

IV. Results

Our results are extracted from 21 primary studies [P1]-[P21], which were published in conferences (12/21), workshops (5/21), journals (3/21), or distributed as white papers (1/21). Since the appearance of the first paper focusing on TD in AI-based systems in 2015 [P3], we observed a growing publication trend until 2020[3]. Interestingly, a large number of primary studies were co-authored by at least one industrial practitioner (13/21), including nine papers authored exclusively by practitioners. Google is the most recurrent company [P3-P7], while other prominent examples include Microsoft [P1], Amazon [P8], and IBM [P9]. The considerable involvement of industrial parties displays the industrial relevance of the topic, which has still to gain traction in academic environments.

In the remainder of this section, we present the results of our study, according to the four RQs guiding the investigation.

A. Characteristics of TD in AI-based Systems (RQ1)

This section reports the results for the sub-questions of RQ1, aiming to characterize the nature of TD in AI-based systems. An overview of the recurrence of all identified TD types is reported in Fig. 2 (both established and new types). Following we discuss the distribution of established TD types plus their variations, new types emerging in AI-based systems, and finally affected quality attributes. Mentioned antipatterns are explained in more detail in Section IV-B.

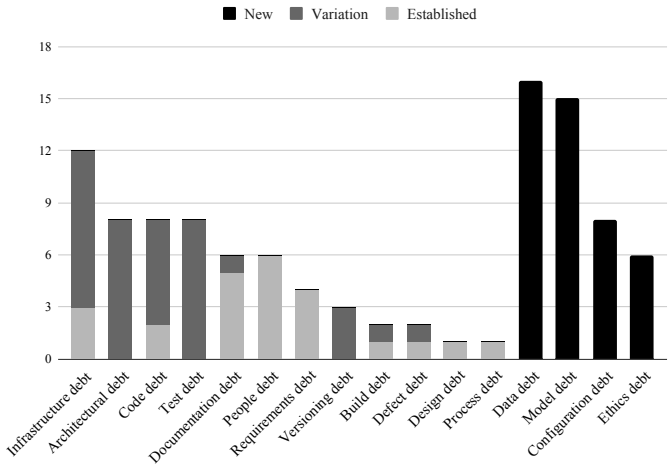

Fig. 2. Recurrence of TD types in AI-based systems

1) Established TD Types and Variations (RQ1.1 & RQ1.2)

As shown in Fig. 2, infrastructure debt is the most recurrent established TD type (12/21), followed by architectural, code, and test debt (8/21). Other types of debt, such as documentation (6/21), people (6/21), requirements (4/21), and versioning debt (3/21) are also reported, albeit less frequently. Lastly, build (2/21), defect (2/21), design (1/21), and process debt (1/21) are only sporadically mentioned.

We observed that this distribution is mostly due to new engineering challenges related to AI-based systems. Our primary studies link these systems to an inherently experimental development process, deep entanglement between architectural components and utilized data, and necessary data

[2]https://doi.org/10.5281/zenodo.4457216

[3]Year in which the primary study selection was executed.

transformation steps between components. Coping with such difficulties often leads to the introduction of TD, e.g. the evolutionary composition of AI pipelines may result in general-purpose components being precariously stitched together via *glue code* [P3]. Additionally, the data-driven nature of AI algorithms introduces novel difficulties for quality assurance, often manifested as suboptimal testing practices, such as under-tested or ill-tested functionalities.

Moreover, AI-specific variations or extensions of TD types, rather than their established scope [12,16], are often used in our primary studies. Specifically, out of the 61 established TD type extractions, 37 are such variations. While some TD types occur unchanged in AI-based systems, re-interpreting several existing debt types is necessary to accommodate the new characteristics of these systems. The most recurrent established debt types, namely infrastructure, architectural, code, and test debt, are also the types which frequently exhibit an extended or augmented scope in our primary studies.

Regarding *infrastructure debt*, this TD type is extended with deficiencies related to the implementation and operation of AI pipelines as well as to the management of AI models. In AI-based systems, this TD type often manifests in form of complex infrastructure comprising various AI pipelines [P3], suboptimal allocation of resources to train/test AI models [P2], and weak AI monitoring and debugging capabilities [P1], leading to major operations and reproducibility issues.

Architectural debt variations instead reflect the emphasis in AI-based systems on data, leading to a deep entanglement of architecture components with their underlying data. This may introduce debt items such as complex and non-deterministic dependencies between architectural components and datasets [P2], hard to assess compositions of architectural elements [P10], or *undeclared consumers* of AI models [P11].

Similarly, *test debt* extends to data testing, ranging from naive omission of basic sanity checks to the lack of more sophisticated tests to assess data quality or distributions [P3]. In addition, new facets include suboptimal practices in testing AI models and pipelines. Their deep connection to training data and the stochastic nature of some AI algorithms [P4,P6] make these artifacts increasingly complex to evaluate.

Code debt is shaped by the experimental nature of AI model development. This frequently emerges in form of *dead experimental code paths* in production code [P12] and the suboptimal refactoring of experimental models into deployable software [P5]. The algorithmic complexity of these systems also increases the likelihood of certain code deficiencies [P13].

For *versioning debt* as a less referenced TD type, we exclusively identified the AI-centric usage of the term. In AI-based systems, this now includes the versioning of AI models and training/testing data, which is often done in suboptimal fashion, if at all [P4,P8,P12].

Finally, documentation, people, requirements, build, defect, design, and process debt mostly appear in our studies according to their established scope, i.e. while such debt types also appear in AI-based systems, characteristics of AI do not have a prominent impact on their manifestation. This highlights that numerous commonalities are shared with software systems not employing AI. Missing documentation, insufficient developer skills, and unclear system requirements are all examples of TD which also frequently occur in non-AI software. Slight variations for some of these TD types are the extension of documentation debt to features and assumptions on the used data [P14], of build debt to suboptimal dependencies of internal and external AI models [P10], and of defect debt to ignored issues related to the quality of model predictions [P4].

2) New TD types in AI-based systems (RQ1.3)

With this RQ, we wanted to synthesize new TD types important for AI-based systems, i.e. types not included in the taxonomies of Li et al. [12] and Rios et al. [16]. Specifically, we found four such types of debt: *data debt*, *model debt*, *configuration debt*, and *ethics debt*, which we further describe below.

Data debt. The most recurrent new TD type regards suboptimal constructs around the data used in AI-based systems (16/21). Specifically, this TD type refers to deficiencies related to the collection, management, and usage of data, both for training and production [P7,P15,P16]. In addition to causing immediate issues, this TD type can also be latent, i.e. not manifesting itself immediately, but rather posing a risks for the long-term evolution of systems. Commonly referenced instances of data debt are data quality issues, unmanaged data dependencies and anomalies, or poor data relevance. Given their heavy reliance on data, this TD type can strongly impact AI systems, including reduced classification effectiveness, data loss due to *premature aggregation* [P15], and compatibility issues.

Model debt. The second most referenced new TD type is model debt (15/21). This AI-specific debt type regards suboptimal practices in the design, training, and management of AI models [P3,P4,P8]. As such, model debt manifests itself as deficiencies occurring exclusively in model-related constructs. Most prominently, model debt originates from suboptimal feature selection processes, neglected hyperparameter tuning, and poorly engineered model deployment strategies. Recurrent items of model debt are *feature entanglement* [P3], *hidden feedback loops* [P17], *unrecognized model staleness* [P8], and substantial differences between training and production performance, i.e. *training/serving skew* [P3]. As models constitute the logic kernel of AI-based systems, this TD type can have serious consequences, ranging from major challenges in maintaining a model, to severe deterioration of model accuracy.

Configuration debt. This debt type (8/21) describes deficiencies around the configuration mechanisms of AI-based systems [P3,P12,P17]. Often, configuration debt arises when the complexity of e.g. dynamic feature selection, hyperparameter tuning, and data pre- and post-processing makes it difficult to efficiently outsource machine- and human-readable configuration files for these activities. This encourages AI engineers to take shortcuts and to only consider the clean-up, restructuring, and commenting of configuration files as an afterthought. While the lines of configuration for AI-based systems may even exceed the lines of source code [P3], configurations are frequently not given the same level of quality control as code,

e.g. reviews or tests. Prominent instances of configuration debt include massive/complex, poorly documented (or simply undocumented), unversioned, or untested configuration files. As such, configuration debt may have some touch points with e.g. infrastructure or documentation debt, but its explicit description in the context of AI still warrants its own TD type.

Ethics debt. One less referenced new debt type is ethics debt (6/21), which comprises deficiencies around ethical aspects of AI-based systems, such as algorithmic fairness, prediction bias, or a lack of transparency and accountability [P14,P18,P19]. Specifically, ethics debt arises when socio-ethical concerns are deliberately or inadvertently neglected during the design or training phase of AI-based systems. While this can go hand in hand with reduced model accuracy, the resulting systems may also satisfy all technical requirements while leaving one or more ethical concerns unaddressed. This debt type can lead to ethical fallacies so deeply embedded into an AI-based system that they usually cannot be resolved with only minor data or software changes. Instead, they may require a complete restructuring and retraining of AI models or major source code updates. Depending on the relevant regulations, ethics debt can also have legal consequences.

3) Affected quality attributes (RQ1.4)

We extracted a total of 12 unique quality attributes impacted by TD in AI-based systems, for which the recurrence in our 21 primary studies is depicted in Fig. 3.

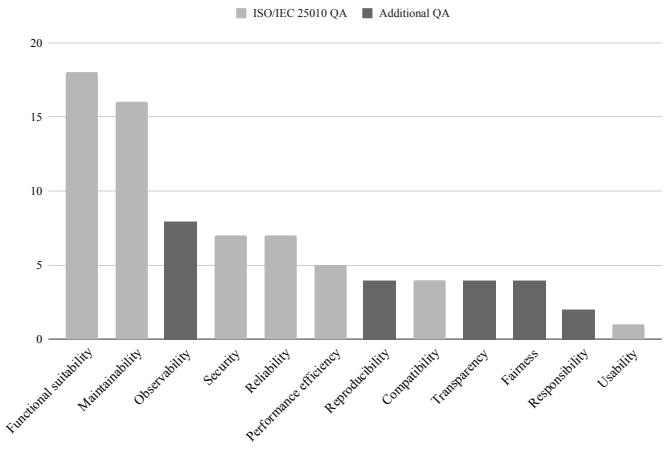

Fig. 3. Recurrence of quality attributes impacted by TD in AI-based systems

Overall, functional suitability is the most referenced quality attribute (18/21). Specifically, the introduction of TD in AI-based systems leads to issues in the functional correctness sub-characteristic (18/21), which, given the data-driven nature of AI-based systems, can be cumbersome to detect and resolve. Among the most mentioned reasons for this is diminished model accuracy, e.g., due to *training/serving skew* [P4]. Maintainability is the second most mentioned quality attribute (16/21), with emphasis on the sub-characteristics modifiability (14/21), testability (13/21), and reusability (11/21). In this case, we noted how TD frequently leads to quality issues specific to or caused by AI models, such as difficult model re-training or reuse, ripple effects on changes through harmful dependencies, or complex collections of scripts and pipelines that are hard to analyze. Observability (8/21) as additional quality attribute represents the degree to which the runtime behavior of a deployed AI-based system can be monitored. This quality attribute can be severely impacted by infrastructure and architecture debt. Security and reliability are referenced equally often (7/21). Security issues in AI-based systems often concern the sub-characteristics accountability (5/21) or confidentiality (4/21). Reliability, by contrast, is most often related to maturity of AI-based systems, i.e. the degree to which the system meets expectations under normal operation. Other quality attributes, such as performance efficiency (5/21), compatibility (4/21), and usability (1/21), are mentioned less frequently, i.e. are most likely less impacted by TD in AI-based systems. Similarly, new quality attributes emerging in AI-based systems such as reproducibility (4/21), fairness (4/21), transparency (4/21), and responsibility (3/21) are also not frequent. We believe the growing attention to topics related to these quality attributes in academia and industry will probably lead to a higher recurrence in the near future. Finally, the only attribute from ISO/IEC 25010 not mentioned at all is portability.

Main findings (RQ1): Infrastructure debt (12) is the most recurrent established TD type in AI-based systems, followed by architectural, code, and test debt (8). We identified four new debt types emerging in AI-based systems, namely data, model, configuration, and ethics debt. Functional suitability (18) and maintainability (16) are the most impacted quality attributes, followed by observability (8), security (7), and reliability (7).

B. Antipatterns (RQ2)

From the 21 primary studies, we extracted a total of 72 unique antipatterns for AI-based systems. To organize this large collection, we formed six categories (with an additional level of subcategories for the larger ones), where each antipattern is assigned to exactly one category. These categories, and associated number of antipatterns, are displayed in Table I, while Table II lists the 14 most prominent antipatterns mentioned in at least three publications. The largest categories are *model antipatterns* (29) and *data antipatterns* (22), which together account for ∼70% of all identified antipatterns. The four remaining categories – *design & architecture antipatterns* (8), *code antipatterns* (5), *infrastructure antipatterns* (4), and *ethics antipatterns* (4) – are all of similar size. In the following subsections, we briefly present each category with some antipattern examples.

Model Antipatterns. The largest group of identified antipatterns (29) describes deficiencies or suboptimal practices with AI models, mostly in the context of machine learning. Often, these refer to the training, validation, or management of models, i.e. to specific activities in the model life cycle. A concrete example is the training antipattern *direct feedback loops* [P3,P15,P20], which describes the unwanted state that a model impacts

TABLE I
ANTIPATTERN CATEGORIES, IN PARENTHESES: # OF UNIQUE ANTIPATTERNS PER (SUB)CATEGORY

Category	Subcategory	Sources
Model (29)	Training (9)	[P3,P4,P14,P19] [P15,P20,P21]
	Training/serving skew (6)	[P4,P12,P14,P19] [P2,P6,P15,P20]
	Features (6)	[P3,P4,P11,P17,P19,P20]
	Management (4)	[P4,P8,P14]
	Validation (4)	[P3,P4,P18-P20]
Data (22)	Management (7)	[P4-P6,P12,P15,P16]
	Anomalies (6)	[P7,P9,P21]
	Quality (4)	[P3,P7,P14,P15] [P9,P16,P21]
	Relevance (3)	[P14,P19]
	Dependencies (2)	[P3,P10,P11]
Design & architecture (8)	Modularity (4)	[P3,P5,P11,P14,P20]
	Integration (2)	[P2,P3,P17]
	Technology adoption (2)	[P3,P5,P8,P17]
Code (5)	Recurrent in AI (3)	[P13]
	AI-specific (2)	[P3,P12,P17,P19]
Infrastructure (4)	–	[P1,P2,P4,P12,P20]
Ethics (4)	–	[P4,P14,P17,P18]

TABLE II
MOST REFERENCED ANTIPATTERNS (OCCURRED IN AT LEAST 3 SOURCES)

Antipattern Name	Category	Sources
Training/serving skew	Model	[P4,P12,P19] [P2,P15,P20]
Data duplication	Data	[P7,P9,P15,P16]
Data miscoding smell	Data	[P3,P7,P15,P21]
Null/missing data values	Data	[P7,P9,P15,P16]
Feature entanglement	Model	[P3,P11,P17,P20]
Dead experimental codepaths	Code	[P3,P12,P17]
Unstable data dependencies	Data	[P3,P10,P11]
Unsound/missing metadata	Data	[P6,P15,P16]
Glue code	Design & architecture	[P2,P3,P17]
Undeclared consumers	Design & architecture	[P3,P11,P20]
Multiple-language smell	Design & architecture	[P3,P8,P17]
Correction cascades	Design & architecture	[P3,P11,P14]
Weak or missing monitoring	Infrastructure	[P1,P2,P4]
Direct feedback loops	Model	[P3,P15,P20]

its own future training data selection. This self-sustaining relationship may lead to wrong classifications and biased decisions, especially if feedback loops remain hidden or are not managed appropriately. An example for a model validation antipattern is *offline/online proxy metric divergence* [P4,P20]. Effectiveness of a production system is usually evaluated with metrics like user engagement or revenue (online), while models of AI components in such a system are validated with e.g. accuracy or mean squared error (offline). If offline metrics are not sufficiently aligned with the online metrics, e.g. via correlation, system effectiveness may be severely impacted. Lastly, a general model management antipattern is the absence of versioning and version control systems specifically for the model (*no version control for models* [P4]), which hinders sustainable evolution and potential rollbacks.

A different type of antipatterns in this category focuses on model features and their relationships. An example of this is *feature entanglement* [P3,P11,P17,P20], which refers to the interdependence of different model features. Adding, removing, or changing the distribution of one feature often has an impact on other features, therefore hindering incremental system improvement. This has also been described as the CACE principle [P3], i.e. "changing anything changes everything". A second example are *epsilon features* [P4,P17], which are features only leading to negligible model improvement. Since every feature comes with costs for maintenance and evolution (especially when considering *feature entanglement*), feature inclusion should be carefully considered based on merit.

Finally, a frequently mentioned type of model antipatterns is related to *training/serving skew* [P2,P4,P12,P15,P19,P20]. This is the most recurrent unique antipattern and, in general, describes substantial model accuracy divergences in the production system when compared to the training accuracy. Reasons for this can be different code paths to compute features in production, but also a non-representative or non-exhaustive training data set. More specialized variants of this antipattern have been called *distribution skew* or *scoring/serving skew* [P6]. The divergence between training and serving accuracy can also emerge slowly over time, which is called *data drift* [P15,P20] and leads to *stale models* [P4,P14].

Data Antipatterns. The second largest antipattern group (22) is related to deficiencies or suboptimal practices around the data of AI-based systems. Since data is the foundation for machine learning models, such antipatterns can substantially diminish system effectiveness. Many instances in this area are related to data quality or the existence of data anomalies. Examples of bad quality are *data duplication* [P7,P9,P15,P16], *null/missing data values* [P7,P9,P15,P16], or *data miscoding smells* [P3,P7,P15,P21], where an attribute is represented with an unsuitable data type or format. Similarly, examples for anomalies in machine learning data can be an *unnormalized feature* [P7,P21], where values exhibit large variance, or very few *extreme outliers* [P7], which may distort important aggregate values used by models. While of both these subcategories can also be important for data-intensive non-AI systems, these antipatterns have been specifically described in the AI context.

Data relevance is another mentioned property that can be subject to antipatterns. Selected examples are the *emphasis on available data* [P14] or the usage of *overcurated data* [P19], both of which can lead to *training/serving skew*.

One of the larger subcategories focuses on data management. Since it can be quite complex in some cases, an *undocumented data collection process* [P16] may negatively affect the long-term evolution of an AI-based system. Similarly, *premature data aggregation* [P15] during this collection process can destroy important data points that cannot be retrieved again. A third example, which is frequently mentioned, is *unsound/missing*

metadata [P6,P15,P16], i.e. a suboptimal or absent documentation and schema to describe the used data.

Lastly, a smaller subcategory is related to data dependencies. An example here is *unstable data dependencies* [P3,P10,P11]. Consuming data from other systems as input signals for model features may initially speed up development. However, if the external data is unstable and changes over time, these dependencies can have negative and hard to diagnose effects on the related ML component.

Design & Architecture Antipatterns. We also extracted eight antipatterns related to the design and architecture of AI-based systems. While it is one of the smaller categories, it contains several frequently mentioned antipatterns. An example related to modularity is *undeclared consumers* [P3,P11,P20], i.e. if the results of an AI component serve as input for a broad range of other systems or components. These undeclared or silent consumers constitute hidden coupling, which can have negative and obscure side effects during software evolution. Similarly, an antipattern where, instead of the output, the complete model is reused and slightly altered is called *correction cascades* [P3,P11,P14]. Changes in the original model then may lead to unintended ripple effects cascading to the "corrected" downstream models. In [P14], this is also described with the improper reuse of complete AI components or pipelines.

A second type of antipatterns is concerned with software integration. The prime example here is *glue code* [P2,P3,P17]. AI-based systems are often built with many generic packages or components, which are then connected with custom code, e.g. for data transformation or reading and writing data. This not only makes it difficult to keep an overview of the system but glue code may also tightly couple the system to specific external libraries. In the area of data collection and preparation, a specialized version of glue code is called *pipeline jungles* [P3,P17], i.e. the same stitching together but more on an architectural level and with ML pipelines.

Finally, a small subcategory regards technology adoption. An example, not fully AI-specific, but still mentioned as a consequence of the nature of AI systems, is the *multiple-language smell* [P3,P8,P17]. While using Python or R for ML models, and other languages for non-ML components, may enable using the best frameworks or libraries for the task at hand, it also entails disadvantages in maintaining, testing, or handing over a component to colleagues.

Code Antipatterns. We generally identified two subcategories of code antipatterns: those specific to AI-based systems and generic ones that occur more frequently in these systems. A frequently mentioned AI-specific example is *dead experimental codepaths* [P3,P12,P17]. The influence of data science leads to an iterative and experimental development process for AI components, where several conditional branches exist, which increase complexity and may also be forgotten, resulting in dead code. Examples of generic code antipatterns which occur more frequently in AI software are *long lambda functions* or *long ternary conditional expressions* [P13].

Infrastructure Antipatterns. We also identified a small number of antipatterns related to the infrastructure of AI-based systems. The most prominent example from this category is *weak or missing monitoring* [P1,P2,P4]. Using AI components leads to additional observability requirements, e.g. monitoring data sources or model accuracy to detect *training/serving skew*. Moreover, the black-box nature of AI components can make it difficult to perform root cause analysis without specialized tooling. Another infrastructure antipattern is hence *weak or missing debugging* [P1]. As a last example, *inadequate configuration management* [P12] describes missing or suboptimal tooling mechanisms to manage important model configuration, e.g., features, preprocessing settings, or hyperparameters.

Ethics Antipatterns. The last smaller category of antipatterns is concerned with ethical issues in AI-based systems. The obvious example are *biased models* [P4,P17], i.e. models that have been created based on incomplete or irrelevant data or with a prejudice-inducing algorithm or process. Such models not only produce inaccurate but also unfair results, which depending on the use case can have substantial negative societal effects, e.g. with predictive policing or recidivism models. For such usage scenarios, it is especially important to measure and control the consequences of the respective AI system. Failing to do so is described by the antipattern *unmanaged social impact* [P14]. A final example in this category is *undefined human accountability* [P14], i.e. when the role and responsibility of humans in AI-supported decisions is not clearly documented, allowing people to hide behind a machine.

Main findings (RQ2): We extracted 72 unique antipatterns in six categories. Largest categories are *model* (29) and *data* (22). *Design & architecture* only consists of eight antipatterns, but many of them occurred several times. The antipattern which was mentioned the most is *training/serving skew* (6).

C. Solutions (RQ3)

From the 21 primary studies, we identified 46 unique instances of solutions. Out of these, about a third of the instances explicitly mentions a TD type, another third mentions an antipattern, while the remaining third does not mention any specific TD type or antipattern addressed. In particular, the last group contains advices of broad and generic nature (e.g. *perform extensive testing* [P4]) and specific methods typically implemented in a tool or framework (e.g. *Data Quality Advisor* [P9]). Given the difficulty in categorizing solutions using a single dimension (TD type, antipattern, or specificity level), we focus instead on discussing the five solutions that are most referenced in our primary studies (Table III).

Manage model configuration prescribes that configuration changes in AI applications should be tracked, reviewed, and possibly tested in the same way as code [P12]. In this line, a suggested good practice is to externalize the configuration options from the code and to maintain them in human- and machine-readable files [P13]. *Use clear component and code APIs* relates instead to reducing design and architectural debt by encapsulating AI functionality in software components with

TABLE III
MOST REFERENCED SOLUTIONS (OCCURRED IN AT LEAST 2 SOURCES)

Solution Name	# of Sources	Sources
Manage model configuration	3	[P8,P12,P13]
Use clear component and code APIs	3	[P3,P5,P17]
Remove unnecessary features	2	[P3,P17]
Refactor the code	2	[P10,P13]
Monitor deployed models	2	[P1,P8]

clear required and provided interfaces [P5,P17]. At the code level, wrapping black-box packages into custom APIs can address the *glue code* antipattern [P3]. *Remove unnecessary features* prescribes to reduce model debt by periodically examining if all input features of a model are still needed [P3], e.g. with so-called "leave-one-feature-out evaluations". Additionally, new features should not be introduced if they do not significantly contribute to the prediction performance [P17]. *Refactor the code* is a straightforward and generic solution to deal with design and code debt, e.g. in the form of code-related antipatterns [P13]. While this is a general best practice, AI-based systems require collaboration with expert developers when refactoring is needed, e.g. to boost performance or re-implement complex ML algorithms [P10]. Finally, *monitor deployed models* suggests that ML models and their prediction performance should be closely monitored after deployment, e.g. to identify and address *training/serving skew* [P1,P8].

Main findings (RQ3): We extracted 46 unique solutions. The solutions either explicitly address a TD type or an antipattern, or present a general best-practice to resolve TD in AI-based systems. The most referenced solutions are *manage model configuration* (3) and *use clear component and code APIs* (3).

V. THREATS TO VALIDITY

Several limitations have to be mentioned for our study. *Internal validity* is influenced by the applied scientific rigor and potentially hidden confounding factors, both of which may impact the consistency and correctness of the results. Since selection, extraction, and synthesis activities of an SMS may rely partially on subjective interpretation, they may be prone to researcher bias. Although we diligently designed and adhered to our SMS protocol and always assigned at least two researchers to each paper, other researchers may have achieved slightly different results with our protocol.

External validity is concerned with the generalizability of the results. With 21 final papers, our SMS can be regarded as comparatively small, which indicates that research on this topic is just getting started. Moreover, many of our primary studies directly reference Sculley et al. [P3] and build on their findings. The majority of publications from industry is also from large software enterprises like Google, Amazon, or Microsoft. Several results of our study are therefore heavily skewed towards Internet-scale ML systems. As a consequence, reported facets of TD types or the relevance of certain antipatterns may slightly differ in other AI contexts.

VI. CONCLUSION

In this paper, we aimed at characterizing the notions of TD and antipatterns for AI-based systems by performing a systematic mapping study. Our research questions focused on both established and new types of TD in these systems, but also on reported antipatterns and solutions. We identified four new TD types emerging in AI-based systems (data, model, configuration, and ethics debt) and observed that several established types (e.g. infrastructure, architectural, code, and test debt) are frequently occurring, although their scope was extended to include AI-specific aspects, e.g., the management and monitoring of both AI pipelines and models to mitigate infrastructure debt. We also identified and categorized 72 unique antipatterns, the majority of which relate to data and models. Finally, we identified 46 solutions that can be used to reduce or prevent debt accumulation in AI-based systems.

For industry, our results can support AI/ML professionals to better communicate aspects of TD present in their systems, to raise awareness for common antipatterns, and to identify solutions to address both. From a research perspective, our contribution provides an encompassing overview and characterization of TD and antipatterns that can emerge in the development of AI-based systems. This study may also serve as a foundation for future research that both deepens our understanding of particular AI debt types and proposes more elaborate solutions to address them. In this respect, we see potential for follow-up grey literature or interview studies in this area, as well as the development of tools and techniques to identify, address, or avoid specific AI antipatterns.

REFERENCES

[1] François Chollet, *Deep Learning with Python*. Manning, 2017.

[2] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software Engineering for Machine Learning: A Case Study," in *International Conference on Software Engineering: Software Engineering in Practice*, IEEE/ACM, 2019.

[3] M. S. Rahman, E. Rivera, F. Khomh, Y.-G. Guéhéneuc, and B. Lehnert, "Machine Learning Software Engineering in Practice: An Industrial Case Study," *arXiv:1906.07154 [cs]*, June 2019. arXiv: 1906.07154.

[4] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden Technical Debt in Machine Learning Systems," in *International Conference on Neural Information Processing Systems*, 2015.

[5] J. Bosch, I. Crnkovic, and H. H. Olsson, *Engineering AI Systems: A Research Agenda*, pp. 1–19. IGI Global, 2021.

[6] W. Cunningham, "The WyCash Portfolio Management System," in *OOPSLA proceedings*, 1992.

[7] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, "Managing technical debt in software engineering (Dagstuhl Seminar 16162)," in *Dagstuhl Reports*, vol. 6, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[8] A. Agarwal, S. Bird, M. Cozowicz, L. Hoang, J. Langford, S. Lee, J. Li, D. Melamed, G. Oshri, O. Ribas, *et al.*, "Making contextual decisions with low technical debt," 2016.

[9] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, "The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction," in *International Conference on Big Data*, IEEE, 2017.

[10] J. Liu, Q. Huang, X. Xia, E. Shihab, D. Lo, and S. Li, "Is Using Deep Learning Frameworks Free? Characterizing Technical Debt in Deep Learning Frameworks," in *International Conference on Software Engineering: Software Engineering in Society*, ACM/IEEE, 2020.

[11] G. Fairbanks, "Ur-technical debt," *IEEE Annals of the History of Computing*, vol. 37, no. 04, 2020.

[12] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," Elsevier, 2015.
[13] R. Verdecchia, P. Kruchten, P. Lago, and I. Malavolta, "Building and evaluating a theory of architectural technical debt in software-intensive systems," Journal of Systems and Software. Elsevier, 2021.
[14] Y. Guo and C. Seaman, "A portfolio approach to technical debt management," in *Workshop on Managing Technical Debt*, ACM, 2011.
[15] D. A. Tamburri, P. Kruchten, P. Lago, and H. van Vliet, "What is social debt in software engineering?," in *International Workshop on Cooperative and Human Aspects of Software Engineering*, IEEE, 2013.
[16] N. Rios, M. G. de Mendonça Neto, and R. O. Spínola, "A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners," vol. 102, Elsevier, 2018.
[17] W. Brown, R. Malveau, H. S. McCormick, and T. Mowbray, *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. New York, NY, USA: John Wiley & Sons, Inc., 1st ed., 1998.
[18] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 1999.
[19] J. Garcia, D. Popescu, G. Edwards, and N. Medvidovic, "Identifying Architectural Bad Smells," in *2009 13th European Conference on Software Maintenance and Reengineering*, IEEE, 2009.
[20] F. Khomh, M. Di Penta, and Y.-G. Gueheneuc, "An Exploratory Study of the Impact of Code Smells on Software Change-proneness," in *Working Conference on Reverse Engineering*, IEEE, 2009.
[21] F. A. Fontana, J. Dietrich, B. Walter, A. Yamashita, and M. Zanoni, "Antipattern and Code Smell False Positives: Preliminary Conceptualization and Classification," in *International Conference on Software Analysis, Evolution, and Reengineering*, IEEE, mar 2016.
[22] J. Bogner, T. Boceck, M. Popp, D. Tschechlov, S. Wagner, and A. Zimmermann, "Towards a Collaborative Repository for the Documentation of Service-Based Antipatterns and Bad Smells," in *International Conference on Software Architecture Companion*, IEEE, 2019.
[23] M. Albarak and R. Bahsoon, "Prioritizing technical debt in database normalization using portfolio theory and data quality metrics," in *International Conference on Technical Debt*, ACM Press, 2018.
[24] H. Foidl, M. Felderer, and S. Biffl, "Technical Debt in Data-Intensive Software Systems," in *Euromicro Conference on Software Engineering and Advanced Applications*, IEEE, 2019.
[25] N. Humbatova, G. Jahangirova, G. Bavota, V. Riccio, A. Stocco, and P. Tonella, "Taxonomy of real faults in deep learning systems," in *International Conference on Software Engineering*, ACM, 2020.
[26] M. Felderer and R. Ramler, "Quality Assurance for AI-Based Systems: Overview and Challenges," in *Software Quality: Future Perspectives on Software Engineering Quality. SWQD 2021. Lecture Notes in Business Information Processing*, vol. 404, Springer International Publishing, 2021.
[27] P. Santhanam, "Quality Management of Machine Learning Systems," in *Communications in Computer and Information Science*, vol. 1272, Springer International Publishing, 2020.
[28] A. Serban, K. van der Blom, H. Hoos, and J. Visser, "Adoption and Effects of Software Engineering Best Practices in Machine Learning," in *International Symposium on Empirical Software Engineering and Measurement*, ACM, oct 2020.
[29] J. Siebert, L. Joeckel, J. Heidrich, K. Nakamichi, K. Ohashi, I. Namba, R. Yamamoto, and M. Aoyama, "Towards Guidelines for Assessing Qualities of Machine Learning Systems," in *Quality of Information and Communications Technology*, Springer, 2020.
[30] H. Washizaki, H. Uchida, F. Khomh, and Y.-G. Gueheneuc, "Studying Software Engineering Patterns for Designing Machine Learning Systems," in *International Workshop on Empirical Software Engineering in Practice*, IEEE, dec 2019.
[31] B. Kitchenham, "Procedures for performing systematic reviews," *Keele, UK, Keele University*, vol. 33, no. TR/SE-0401, 2004.
[32] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *International Conference on Evaluation and Assessment in Software Engineering*, ACM Press, 2014.
[33] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," 2008.
[34] International Organization For Standardization, "ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models," 2011.
[35] B. Jenner, U. Flick, E. von Kardoff, and I. Steinke, *A companion to qualitative research*. Sage, 2004.

PRIMARY STUDIES

[P1] A. Agarwal, S. Bird, M. Cozowicz, L. Hoang, J. Langford, S. Lee, J. Li, D. Melamed, G. Oshri, O. Ribas, *et al.*, "Making contextual decisions with low technical debt," 2016.
[P2] G. A. Lewis, S. Bellomo, and A. Galyardt, "Component Mismatches Are a Critical Bottleneck to Fielding AI-Enabled Systems in the Public Sector," in *AAAI Fall Symposium Series: Artificial Intelligence in Government and Public Sector*, 2019.
[P3] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden Technical Debt in Machine Learning Systems," in *International Conference on Neural Information Processing Systems*, 2015.
[P4] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, "The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction," in *International Conference on Big Data*, IEEE, 2017.
[P5] K. O'Leary and M. Uchida, "Common problems with creating machine learning pipelines from existing code," in *Machine Learning and Systems*, 2020.
[P6] N. Polyzotis, M. Zinkevich, S. Roy, E. Breck, and S. Whang, "Data validation for machine learning," vol. 1, 2019.
[P7] N. Hynes, D. Sculley, and M. Terry, "The data linter: Lightweight, automated sanity checking for ml data sets," in *MLSys Workshop*, 2017.
[P8] S. Schelter, F. Biessmann, T. Januschowski, D. Salinas, S. Seufert, and G. Szarvas, "On challenges in machine learning model management," 2018.
[P9] S. Shrivastava, D. Patel, A. Bhamidipaty, W. M. Gifford, S. A. Siegel, V. S. Ganapavarapu, and J. R. Kalagnanam, "Dqa: Scalable, automated and interactive data quality advisor," in *International Conference on Big Data*, IEEE, 2019.
[P10] J. Liu, Q. Huang, X. Xia, E. Shihab, D. Lo, and S. Li, "Is Using Deep Learning Frameworks Free? Characterizing Technical Debt in Deep Learning Frameworks," in *International Conference on Software Engineering: Software Engineering in Society*, ACM/IEEE, 2020.
[P11] H. Belani, M. Vukovic, and Ž. Car, "Requirements engineering challenges in building ai-based complex systems," in *International Requirements Engineering Conference Workshops*, IEEE, 2019.
[P12] A. Arpteg, B. Brinne, L. Crnkovic-Friis, and J. Bosch, "Software engineering challenges of deep learning," in *Euromicro Conference on Software Engineering and Advanced Applications*, IEEE, 2018.
[P13] H. Jebnoun, H. Ben Braiek, M. M. Rahman, and F. Khomh, "The scent of deep learning code: An empirical study," in *International Conference on Mining Software Repositories*, ACM, 2020.
[P14] J. Matthews, "Patterns and anti-patterns, principles and pitfalls: Accountability and transparency in ai," *AI Magazine, Association for the Advancement of Artificial Intelligence*, 2020.
[P15] A. Munappy, J. Bosch, H. H. Olsson, A. Arpteg, and B. Brinne, "Data Management Challenges for Deep Learning," in *Euromicro Conference on Software Engineering and Advanced Applications*, IEEE, 2019.
[P16] V. Gudivada, A. Apon, and J. Ding, "Data quality considerations for big data and machine learning: Going beyond data cleaning and transformations," vol. 10, 2017.
[P17] M. Alahdab and G. Çalıklı, "Empirical analysis of hidden technical debt patterns in machine learning software," in *International Conference on Product-Focused Software Process Improvement*, Springer, 2019.
[P18] V. Vakkuri, K.-K. Kemell, M. Jantunen, and P. Abrahamsson, ""This is Just a Prototype": How Ethics Are Ignored in Software Startup-Like Environments," in *International Conference on Agile Software Development*, Springer, 2020.
[P19] D. Roselli, J. Matthews, and N. Talagala, "Managing bias in AI," in *Companion Proceedings of the World Wide Web Conference*, 2019.
[P20] L. E. Lwakatare, A. Raj, J. Bosch, H. H. Olsson, and I. Crnkovic, "A taxonomy of software engineering challenges for machine learning systems: An empirical investigation," in *International Conference on Agile Software Development*, Springer, 2019.
[P21] H. Foidl and M. Felderer, "Risk-based data validation in machine learning-based software systems," in *International Workshop on Machine Learning Techniques for Software Quality Evaluation*, 2019.