From Machine Learning to Large Language Models: AI-Based Test Oracle Generation

Lidija Vincekovič[1], Tina Beranič[1] and Lili Nemec Zlatolas[1]

[1]Faculty of Electrical Engineering and Computer Science - University of Maribor, Koroška cesta 46, Maribor, Slovenia

Abstract

A test oracle is the mechanism that decides whether the observed behaviour of a program is acceptable for a given input. Producing reliable oracles remains challenging in practice, and the oracle problem has been studied in software testing for more than forty years without a general solution. In recent years, machine learning and language-model-based techniques have increasingly been used to support oracle generation. This paper presents a focused systematic mapping based on an ongoing systematic literature review of AI-based test oracle generation. The synthesis covers 143 primary studies retrieved from three indexing databases and published up to the retrieval cut-off in early May 2026. We organise the paper around three questions: which types of oracles are generated, which AI/ML techniques and input artefacts are used, and how the generated oracles are evaluated. The three most frequent oracle labels are metamorphic-relation oracles, assertion-style oracles, and expected-output oracles; together, they account for 120 coded occurrences in the corpus, although the coding is multi-label and some studies generate more than one oracle type. We also observe a sharp increase in language-model and transformer-based approaches after 2022, from 7.2% of studies up to and including 2022 to 68.9% of studies from 2023 onward. Evaluation practice, however, remains uneven: only 21.7% of studies report a mutation-grounded metric, fewer than 8% report a statistical significance test, and active mitigation of data contamination is rare. We use these observations to identify open challenges for future research on AI-based test oracle generation.

Keywords

test oracle, oracle problem, large language models, prompt engineering, metamorphic testing, software testing

1. Introduction

Software testing relies on a mechanism that decides whether the observed output of a program is acceptable for a given input. This mechanism is called the *test oracle*, and it is a central component of any testing activity [1]. When such an oracle is missing, incomplete, or too expensive to obtain, testing becomes difficult or even unreliable. This situation is known as the *oracle problem*, and it was first described by Weyuker in her work on non-testable programs more than forty years ago [2].

Although the problem has been studied for a long time, it remains open in practice [1, 3]. In many real systems, the correct output is not known in advance, or it can only be described in part. As a consequence, developers still write most assertions manually, which makes the testing process slow and costly [4]. Several research directions have tried to reduce this effort. Some approaches derive oracles from formal specifications, others rely on program properties such as invariants or metamorphic relations [5], and some compare the output of a program against a reference version or a similar system [1]. Each direction has produced useful results, but none of them has solved the oracle problem in general.

In the last decade, machine learning (ML) has been progressively introduced into oracle generation [1, 6]. Early work used classical models to learn oracles from existing tests, while later work moved to neural networks (NNs) and, more recently, to large language models (LLMs) [7, 8, 9]. From around 2023 onward, the volume of LLM-based oracle generation work has grown rapidly. LLMs can generate assertions, expected outputs, and even metamorphic relations from relatively short prompts, and the broader availability of instruction-tuned models over 2022–2023 has coincided with their wider adoption

SQAMIA 2026: Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications, September 16–18, 2026. Niš, Serbia

lidija.vincekovic@um.si (L. Vincekovič); tina.beranic@um.si (T. Beranič); lili.nemeczlatolas@um.si (L. N. Zlatolas)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-19

CEUR-WS.org/Vol-4336/paper5.pdf

for oracle generation [10, 11]. As a result, the number of papers on AI-based test oracles has grown considerably in a short period of time.

This rapid growth raises several questions that current secondary studies do not fully answer. The survey by Barr et al. [1] was written before LLMs became widely available, while the review by Chen et al. [5] focuses only on metamorphic testing. Other reviews look at AI in software testing more broadly, without discussing oracle generation in detail [12]. Because of this, it is not yet fully clear what kinds of oracles AI-based approaches commonly produce, which AI techniques and input artefacts they build on, and how authors generally assess the quality of the generated oracles.

To address this gap, we are conducting a systematic literature review (SLR) on AI-based test oracle generation. The broader review covers several research questions and a wider set of extracted variables. In this paper, however, we report a focused systematic mapping of the parts of the review that are most relevant for an introductory overview of the field. The paper should therefore be read as a preliminary synthesis from an ongoing SLR rather than as the final report of the complete review.

In this paper, we present an introductory and more focused version of that work. We concentrate on three research questions that we consider central for an entry point into the topic:

- **RQ1:** What types of test oracles are generated by AI-based approaches?
- **RQ2:** What AI/ML techniques and input artefacts are used to generate them?
- **RQ3:** How are AI-generated oracles evaluated?

Together, these three questions describe *what* is generated, *how* it is generated, and *how* the generated oracle is assessed. We deliberately limit the scope of this paper to these three dimensions in order to keep the contribution suitable for a shorter workshop paper. Other aspects of the full SLR, including tooling and integration, author-reported threats to validity, and targeted software qualities are left for the extended version.

The rest of the paper is organised as follows. Section 2 presents the background and related secondary studies. Section 3 describes the methodology of the SLR in a compact form. Section 4 reports the results and discusses them around the three research questions. Section 5 concludes the paper and indicates directions for future work.

2. Background and Related Work

2.1. Test oracles and the oracle problem

A *test oracle* is the component of a testing process that decides whether the observed behaviour of a program is acceptable for a given input [1]. In practice, the oracle can take many forms. It can be a human tester, an executable assertion, a formal specification, a previous version of the same program, or even a similar system used as a reference [4, 1]. When the oracle is unavailable or only partially specified, the tester is left without a reliable way to decide whether the observed behaviour is correct, and the testing process becomes considerably harder. Weyuker discussed this situation already in her work on non-testable programs [2], and it has since been studied under the name of the *oracle problem*.

In the literature, oracles are usually grouped into a small number of broad categories [1]. *Specified oracles* are derived from formal or semi-formal specifications and check the output against expected properties. *Derived oracles* are obtained from program-internal information, such as invariants, contracts, or metamorphic relations [5]. *Implicit oracles* detect generic failures, such as crashes or assertion violations, without requiring a specification. The fourth category in Barr et al.'s framework is conceptually different: it covers the situation in which *no automatable oracle is available*, so that research focuses on reducing the effort of the human who has to act as the oracle. Following their terminology, we refer to this category as the *human oracle cost problem* [1]. It is therefore not an oracle type on the same level as the first three; rather, it groups techniques such as test-suite and test-case reduction, readability-improving input generation, and crowdsourcing of oracle decisions to humans [1].

These four categories are conceptual: in practice, primary studies report oracles at a finer-grained operational level, and our synthesis in Section 4.1 counts oracles at that level. Table 1 summarises how

the concrete operational oracle types observed in the corpus map onto Barr et al.'s four conceptual categories.

Table 1
Mapping between Barr et al.'s four conceptual oracle categories [1] and the concrete operational oracle types counted in Section 4.1

Conceptual category	Concrete operational types in this review
Specified	Expected-output oracles; specification-based oracles
Derived	Metamorphic-relation oracles; invariant-based oracles; contract-based oracles; statistical pseudo-oracles; natural-language pseudo-oracles
Implicit	Assertion-style oracles; exception oracles; verdict oracles
Human (residual)	No automatable oracle available; research focuses on reducing the human oracle cost via test-suite or test-case reduction, readability heuristics for human-checked outputs, or crowdsourcing the oracle to humans [1].

AI-based approaches that we discuss later in the paper can, in principle, generate oracles in several of these categories, in particular assertions, expected outputs, and metamorphic relations [7, 9, 11]. By construction of our inclusion criteria, almost all of the studies in our pool produce some form of automatable oracle and therefore fall into Barr et al.'s first three categories; the fourth category is included for completeness and is covered by a single outlier paper that targets human oracle cost reduction rather than oracle automation [13].

2.2. Related secondary studies

Several secondary studies have addressed the oracle problem and related topics. We briefly summarise the works that are most relevant for our review.

The survey by Barr et al. [1] is one of the most cited secondary studies on the oracle problem. It provides a comprehensive taxonomy of oracle types and discusses the main techniques proposed up to that point. However, it was published before the rise of deep learning and LLMs, and therefore does not cover the recent wave of AI-based approaches. An earlier survey by Harman et al. [3] follows a similar direction and discusses trends in oracle research, but it also predates the LLM era and addresses ML only marginally.

The survey by Pezzè and Zhang [4] focuses specifically on automated test oracles. It reviews techniques for deriving oracles from specifications, models, and program executions, and it provides a useful classification of evaluation criteria. As with the previous surveys, it does not yet cover the more recent wave of deep-learning and LLM-based oracle generation that has emerged in the last few years.

A more focused review is the one by Chen et al. [5], which addresses metamorphic testing. Metamorphic testing is an important sub-category of oracle research, because it provides a way to test programs even when the expected output is not known. The review by Chen et al. discusses challenges, opportunities, and application domains, but by design it covers only one class of oracles. It also predates the more recent wave of deep-learning and LLM-based oracle generation, and therefore does not discuss how these techniques have been applied to metamorphic relations in the last few years.

The most recent secondary study directly on our topic is the roadmap article by Molina et al. [6], published in 2025. It reviews 37 LLM-based oracle studies (of which 20 specifically target oracle generation and the remaining 17 produce oracles inside a broader test-generation pipeline), and it organises them along three axes: the kinds of oracles produced, the way the LLM is used (pre-training/fine-tuning versus prompt engineering), and the input artefacts provided to the model. The article identifies eight challenges for the field – oracle quality assessment, bug-detection capability, prompt and training data, reproducibility, dataset bias, and oracle leakage – and proposes a research roadmap for each. The pool, however, is restricted to the LLM era (2020–2025) and is dominated by assertion generation (29 of the 37 studies, ~78%); pre-LLM techniques are summarised narratively rather than counted.

More recent reviews look at the use of AI in software testing in a broader sense. The multivocal

literature review by Garousi et al. [12] analyses AI-assisted test automation across academic and industrial sources. It provides a useful overview of the field, but oracle generation is treated only as one of many sub-topics and is not analysed in depth. Other studies have focused on specific aspects of LLM-based testing, such as prompt engineering for unit test generation [14], or on the evaluation of neural oracle generators [15, 16]. These works are closer to our topic, but they typically address a single technique or a single evaluation question, and they do not provide a systematic synthesis of the AI-based oracle generation field as a whole.

2.3. Positioning of this work

Compared to the secondary studies summarised above, our review differs in three main aspects.

First, the *scope* is explicitly focused on AI-based test oracle generation. We do not cover testing in general, nor do we limit ourselves to a single oracle class such as metamorphic testing. Instead, we consider approaches in which AI or ML techniques are used to generate, infer, or support the evaluation of test oracles. In contrast to recent work that focuses specifically on LLM-based oracle automation [6], we cover a broader methodological spectrum, including classical machine learning, non-LLM neural networks, encoder and encoder–decoder transformer models, and instruction-tuned LLM-based workflows.

Second, the *time window* of the review extends to 2026 and therefore covers the recent rise of large language models. This allows us to analyse how the field has changed in the last few years, in particular after the broad adoption of LLMs in software engineering tasks [10, 11]. By spanning 2010–2026, our review covers not only the LLM era addressed by the Molina et al. [6] roadmap but also the preceding classical-ML and transformer waves, which lets us trace the transition within a single corpus.

Third, the *focus of analysis* in this paper is on three specific questions: the types of oracles that AI-based approaches generate, the AI/ML techniques and input artefacts they use, and the way in which the generated oracles are evaluated. These three questions are central to understanding the current state of the field and to identifying open problems for future research.

3. Methodology

This paper draws on an ongoing systematic literature review on AI-based test oracle generation. The broader review follows the guidelines of Kitchenham and Charters [17] and the PRISMA reporting standard [18]. Since the full review will be reported separately, this paper reports only the part of the protocol that is needed to interpret the focused synthesis in Section 4. To avoid presenting the paper as a complete SLR report, we refer to the present contribution as a focused systematic mapping based on the ongoing review.

The search query was built as a conjunction of three semantic blocks. The first block contains terms related to test oracles and testing artefacts (for example, *oracle*, *assertion*, *metamorphic relation*, *expected output*). The second block contains terms related to artificial intelligence and machine learning (for example, *machine learning*, *deep learning*, *neural network*, *large language model*, *transformer*). The third block contains terms related to the software-testing context (for example, *software testing*, *unit test*, *test generation*). The database-specific search strings were adapted to the syntax of each database. The complete search strings, in the database-specific syntax, shall be available in the extended artefact.

We searched three primary databases: IEEE Xplore, Scopus, and Web of Science. These sources are commonly used in software-engineering SLRs and provide broad coverage of the field [17]. The initial search returned 7,670 records. After de-duplication, 4,539 unique records remained. Title and abstract screening against our inclusion and exclusion criteria reduced this set to 483 candidates, and full-text screening produced 237 studies for the broader review. The criteria were first calibrated on a random pilot sub-sample; the ambiguity observed during calibration concerned almost exclusively how strictly the "AI-based oracle generation" condition should be read when abstracts were short or generic. The pilot therefore led to an explicit *AI-generates-the-oracle* rule, as opposed to a weaker rule in which AI

appears anywhere in the testing pipeline, and all borderline records were re-examined at full-text level with a recorded exclusion reason attached to each decision.

The focused synthesis reported in this paper applies that stricter rule and uses a core pool of 143 studies. This core pool contains studies in which the AI/ML component is used to generate, infer, or support the assessment of a test oracle for a software system under test. Of the 94 studies from the broader pool that are not counted in the present synthesis, the dominant reason (70 / 94, 74%) was that the study applied a testing methodology *to* an AI or ML system, treating the model itself as the subject under test, rather than *using* AI to generate an oracle for arbitrary software. The remaining exclusions covered adjacent testing tasks such as fault localisation or test selection (8), out-of-scope domains such as hardware and VLSI design (8), studies with no AI or ML component in the oracle itself (6), and duplicates or vision papers that did not provide enough information to code the oracle-generation component (2). This narrower scope matches the focus of the recent survey by Molina et al. [6] on AI-generated oracles and keeps our synthesis comparable with the state of the art in that direction. The PRISMA flow is reported in Figure 1.

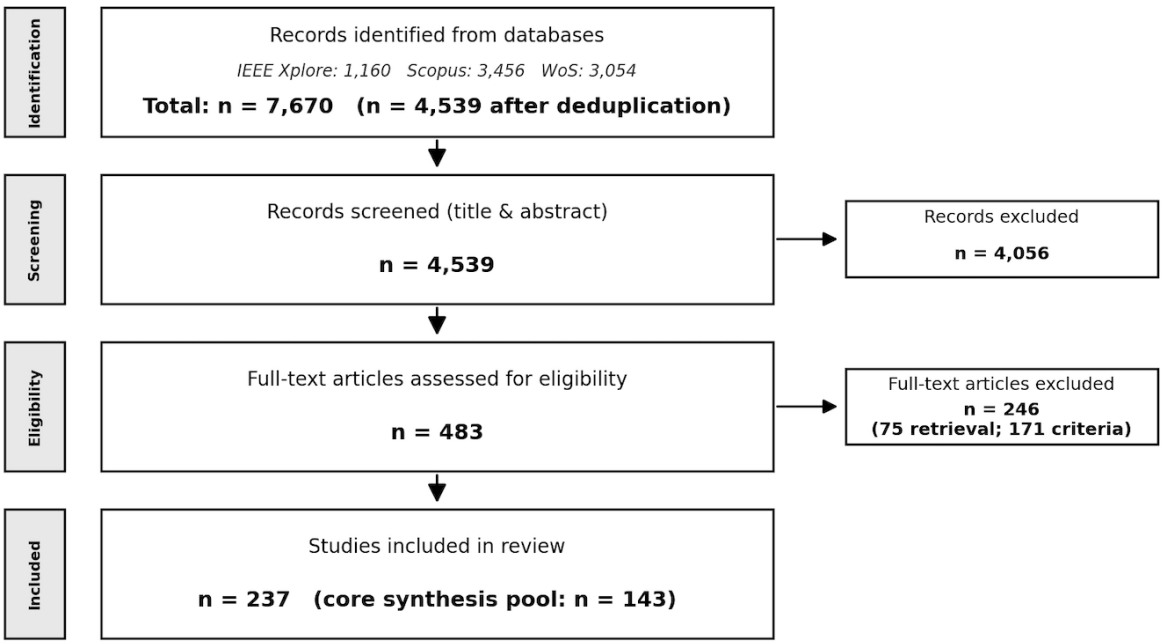


Figure 1: PRISMA 2020 flow for the SLR.

For each of the 143 primary studies, we extracted the variables needed for the three research questions of this paper: oracle type, AI/ML technique, input artefacts, prompting or generation strategy, evaluation setup, benchmark or dataset, reported metrics, statistical reporting, treatment of data contamination, and the role of the human in the pipeline. Several variables were coded using multi-label coding, because a single paper can generate more than one oracle type, use more than one input artefact, or report more than one evaluation metric. The resulting counts should therefore be interpreted as coded occurrences rather than mutually exclusive paper categories. Additional variables related to tooling, integration, threats to validity, targeted software qualities, and adjacent testing tasks were extracted for the broader review but are not analysed in this paper.

4. Results and Discussion

This section reports the focused synthesis of the 143-study core pool. We organise the results around the three questions introduced in Section 3: the oracle types generated by AI-based approaches, the AI/ML techniques and input artefacts used to generate them, and the evaluation practices used to assess the

generated oracles. Where useful, we refer to representative studies from the pool in order to illustrate the observed trends without listing every primary study individually.

When we read the 143 primary studies side by side, we observed that the changes brought by AI to test oracle generation are not concentrated in a single dimension. The oracle types that authors target, the input artefacts that feed the generators, and the way in which the produced oracles are evaluated have all shifted together, in roughly three overlapping eras: an early machine-learning era (2010–2019, $n = 39$), a transformer-based era around models such as TOGA (2020–2022, $n = 30$), and the current LLM era (2023–2026, $n = 74$). Studies that predate the AI-only inclusion criterion of this review (classical, specification-only or reference-only work) are outside the pool by construction. Table 2 summarises this parallel evolution along the three axes that correspond to our research questions. We use it as a reading map for the rest of the section: each of the following subsections zooms in on one column of the table and reports the underlying numbers and representative studies.

Table 2
Evolution of AI-based test oracle generation across the 143 studies, grouped into three overlapping eras.

Era (years)	n	Typical oracle types	Typical input artefacts	Typical evaluation
Early ML era (2010–2019)	39	Expected-output, verdict, and statistical pseudo-oracles	Labelled executions, input/output pairs, hand-crafted features	Accuracy, F1, classification performance
Transformer era (2020–2022)	30	Assertion and exception oracles	Test prefix, method under test, documentation	Exact match, Acc@k, occasional mutation score
LLM era (2023–2026)	74	Assertion, metamorphic, exception, contract-based, and system-level oracles	Test prefix, method signature, source code, comments, documentation, sampled traces	Mutation score, compilability and pass rate, fault-detection rate, hallucination rate

4.1. Three oracle classes account for most of the field

We code oracle types at the concrete operational level, using the mapping to Barr et al.'s four conceptual categories summarised in Table 1. By construction, our inclusion criteria require an AI-based oracle-generation component, so all but one of the 143 studies fall into Barr et al.'s first three categories: specified, derived, or implicit oracles. The single exception is a cost-reduction paper that targets the human oracle cost problem rather than automating the oracle itself [13]. As illustrated in Figure 2, the distribution of coded oracle types is uneven. The three most frequent labels are metamorphic-relation oracles, which appear in 43 studies (30.1%), assertion-style oracles, which appear in 42 studies (29.4%), and expected-output oracles, which appear in 35 studies (24.5%). Together, these three labels account for 120 coded occurrences, which is equivalent to 83.9% of the corpus size. Because the coding is multi-label, however, this number should not be interpreted as the percentage of unique papers that belong exclusively to one of these classes. Some studies generate more than one oracle type within the same pipeline, and the totals therefore exceed 143.

Metamorphic relations were originally proposed as a way to test programs in the absence of a complete oracle [5], and they remain the most studied class in our pool. Recent work in this area is still active, ranging from domain-specific applications such as the OpenStreetMap study by Mendez et al. [19], to techniques for inferring or transforming metamorphic relations automatically [20, 21]. Assertion-style oracles, on the other hand, become dominant specifically in the LLM era, since most recent LLM-based pipelines emit a JUnit-style assertion together with the test input. Expected-output oracles are more typical of pre-2020 work, where ML was used as a trained classifier over input/output features, that is, the oracle itself was a trained model that decided whether a given output was acceptable. The early neural-network oracle proposed by Shahamiri et al. [22] is a representative example.

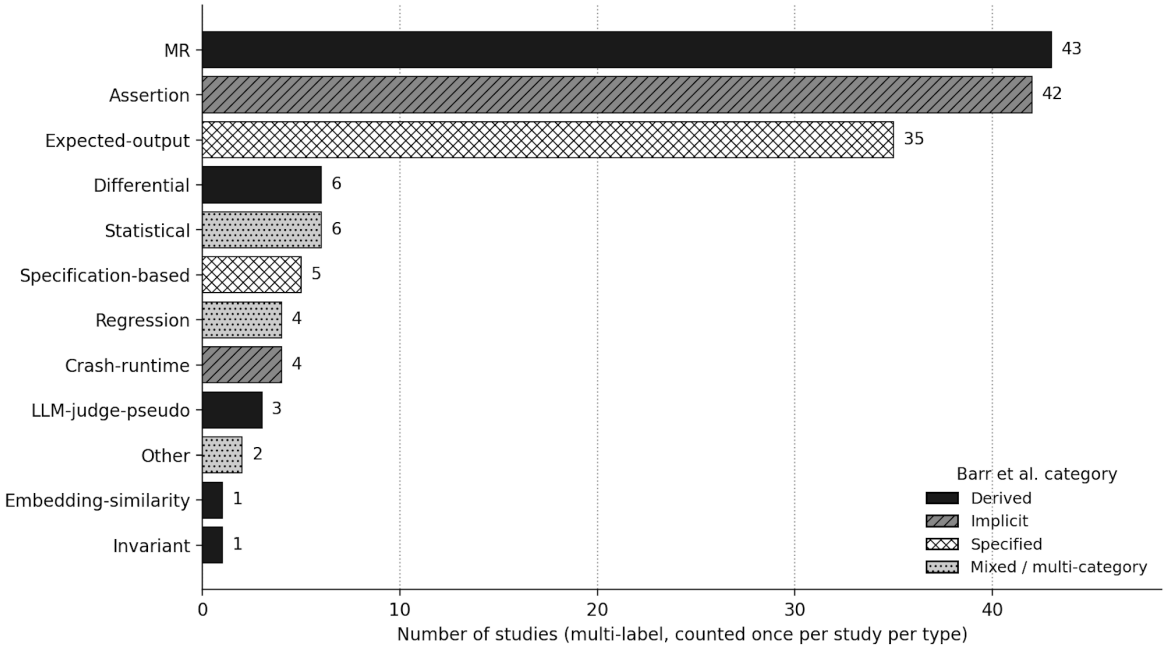


Figure 2: Distribution of oracle types across the 143 primary studies (multi-label coding).

Most studies in the pool target unit-level testing (71 papers, 49.7%) or system-level testing (52 papers, 36.4%). The remaining 14% distributes across GUI and end-to-end testing (4), mixed-granularity studies (4), regression testing (4), API testing (3), property-based testing (2), integration (1), and mutation testing (1). The near-absence of explicit integration-test work (a single paper out of 143) is itself a notable finding: the literature either narrows to a single function or method, or broadens to a full-system black-box harness, with little attention to the inter-component oracle problem.

In terms of application domain, general software-engineering contexts dominate the pool with 102 studies (71.3%), followed by scientific computing (14; 9.8%), automotive software (8; 5.6%), other engineering domains (7; 4.9%), game and graphics systems (4; 2.8%), web and e-commerce (3; 2.1%), and security/fuzzing (2; 1.4%). DSL, aerospace, and healthcare each contribute a single paper. The automotive sub-stream is concentrated in European industrial collaborations, such as [23, 24], and it is one of the few areas where specification-based input artefacts are still visible.

Overall, our observation is that AI-based approaches do not introduce a fundamentally new oracle category. Rather, they automate or reduce the cost of producing oracle types that already fit established taxonomies. This is particularly visible for assertion-style oracles: recent language-model-based pipelines can generate many candidate assertions from a short prompt and can combine generation with compilation or execution feedback [8, 9, 10]. The novelty of the recent literature is therefore less conceptual than methodological: AI changes the process, cost, and scale of oracle generation more than it changes the underlying definition of what an oracle is. This finding is consistent with broader secondary studies on AI-assisted software testing [12].

4.2. From classical machine learning to language-model-based generation

Looking at the publication trend of the 143 studies, we observe a clear inflection at the boundary between 2022 and 2023, followed by a strong acceleration in publication volume. Work up to and including 2022 (69 papers) is dominated by classical ML and non-LLM neural networks: only 5 of these 69 studies (7.2%) belong to the broader language-model and transformer-based family. From 2023 onward (74 papers), this share rises to 68.9% (51/74), a difference of more than 60 percentage points between the two periods. The year-by-year trajectory increases from 2021 onward: 12.5% in 2021, 33% in 2022, 43% in 2023, 67% in 2024, 77% in 2025, and 88% in early 2026. The 2026 figure is partial because

the retrieval cut-off was in early May 2026. The 2022–2023 boundary coincides with the broader public availability of instruction-tuned LLMs over the same period, including ChatGPT in late 2022, GPT-4 in early 2023, and the first widely used open-weight Llama and Mistral variants shortly after. Our data are descriptive, however: we observe a before/after pattern but do not attribute the shift to any single model release.

For the purposes of this paper, we use the term *language-model and transformer-based family* as an umbrella category rather than as a synonym for instruction-tuned LLMs. This family includes encoder-only transformer models, encoder–decoder and sequence-to-sequence models such as T5 and CodeT5, small fine-tuned transformer models, and instruction-tuned decoder LLMs used through prompting, retrieval augmentation, or agentic workflows. This terminology avoids treating all transformer-based models as equivalent to modern conversational LLMs, while still allowing us to describe the broader transition from classical ML and non-LLM neural networks toward language-model-based oracle generation. Across the full pool, this consolidated family is the largest AI family with 56 studies (39.2%), followed by non-LLM neural networks (36; 25.2%), classical ML (30; 21.0%), hybrid approaches (9; 6.3%), symbolic techniques (5; 3.5%), genetic or search-based approaches (3; 2.1%), and reinforcement learning (2; 1.4%).

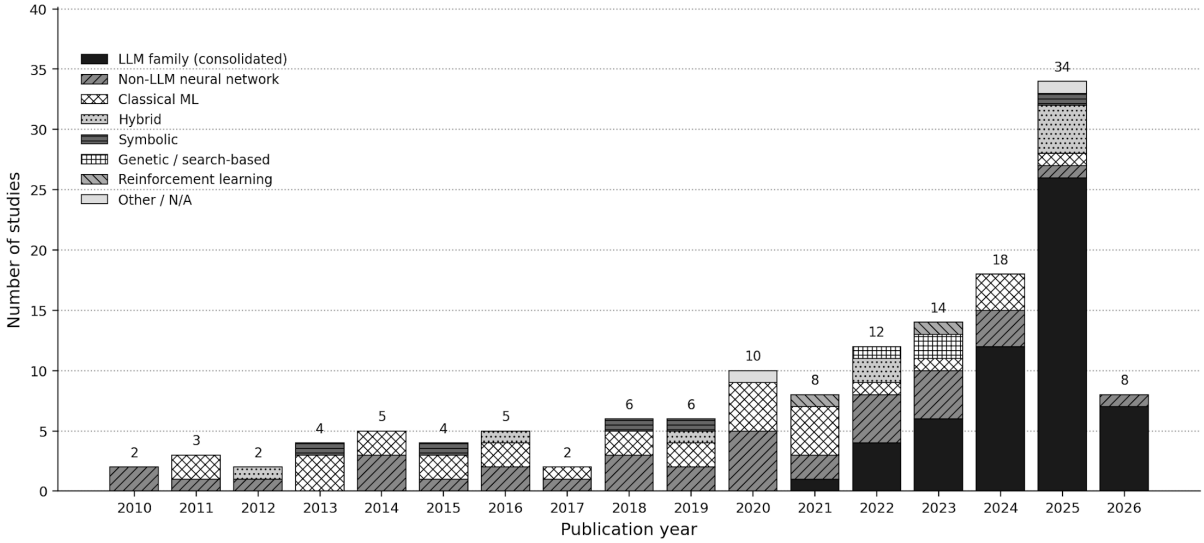


Figure 3: Publication trend by AI family across the 143 primary studies; the 2026 figure is partial due to the retrieval cut-off in early 2026.

Figure 3 shows the year-by-year distribution of studies across the seven AI families in the pool. The plot makes visible the near-flat volume of oracle-generation research before 2020, the gradual growth during the transformer era (2020–2022), and the pronounced concentration of activity in the language-model family from 2023 onwards.

Within the language-model and transformer-based cohort, a small number of model families account for much of the recent activity. The GPT-4 family, including GPT-4o, GPT-4.1, and the *mini* variants, appears in 26 papers; the GPT-3.5 family in 21; Llama in 8; CodeT5 in 5; Claude in 4; Gemini in 3; DeepSeek in 3; Mistral or Mixtral in 3; OpenAI reasoning models such as o1, o3, and o4-mini in 3; and Qwen in 2. These counts are not mutually exclusive, since some studies compare several models. We interpret this distribution as reflecting both technical interest and practical availability: public APIs and open-weight model releases strongly influence which models can be evaluated by the research community.

The input artefacts consumed by these techniques are concentrated on a few categories (multi-label coding). Source code is by far the most common input (70 papers, 49.0%), followed by execution traces (39, 27.3%), existing tests (34, 23.8%), natural-language requirements (17, 11.9%), API documentation (9, 6.3%), formal specifications (8, 5.6%), UI logs (5, 3.5%), and input–output pairs (5, 3.5%). This distribution

mirrors the dominance of unit-test assertion generation: most pipelines start from the code of the method under test, sometimes augmented with the signature of the test method or with a small set of example assertions. The substantial share of execution traces is driven by hybrid and reinforcement-learning approaches that observe runtime behaviour to derive expected-output or invariant-based oracles, and by recent LLM-family studies that feed sampled traces or coverage data into the prompt as additional context. Formal specifications and natural-language requirements, although classical inputs for oracle generation, remain less frequently used and are mostly present in the cyber-physical, automotive, and formal-methods sub-streams.

For studies that use instruction-tuned LLMs or promptable code models, the prompting strategy becomes a central design dimension (multi-label coding). The most common strategies are zero-shot prompting (22 papers), few-shot prompting (16), template-based prompting (16), and chain-of-thought prompting (12). Validate-fix loops, in which the model output is checked against a feedback signal such as compilation errors, test-execution results, or a semantic check and then refined, appear in 7 papers. These studies often report improvements over single-shot baselines within their own experimental settings. However, because the studies differ in benchmarks, models, oracle types, and metrics, the current evidence does not support a robust cross-study ranking of prompting strategies. Retrieval-augmented generation appears in 6 papers, while tree-of-thought reasoning and multi-agent consensus appear in 2 papers each. Ten studies in this family do not report a recognisable prompting strategy, typically because they fine-tune the model rather than use prompt engineering as the main design variable.

A small set of representative systems illustrates how the design space has matured over time. TOGA [9] is a neural baseline that infers the type of oracle and generates a single assertion or exception oracle from a test prefix and the method under test. TOGLL [11] keeps the same task but uses a fine-tuned code LLM and proposes several candidates, which raises the chances of obtaining a strong oracle. ChatAssert [16] adds a tool-augmented prompting loop, in which the LLM output is compiled and executed and the feedback is fed back into the prompt until a runnable assertion is produced. More recent multi-agent designs, such as the CANDOR-style pipeline reported by Cañizares et al. [21], push this idea further by having several LLM agents propose, critique, and reconcile a candidate oracle before it is returned to the developer. Across these four representative systems, the main novelty reported by the authors is not the underlying model but the surrounding generation process, which moves from single-shot inference to fine-tuning, then to tool feedback, and finally to multi-agent consensus. Because each system is evaluated on different benchmarks and metrics, this observation is a qualitative pattern in the design space rather than a quantitative performance ranking.

The three eras identified in Table 2 differ not only in the AI technique used but also in the challenges they raise for oracle generation. Early machine-learning approaches (2010–2019) relied heavily on manual feature engineering and were constrained by small, single-project datasets; the resulting oracles were often expressed as classifiers over hand-crafted features and did not generalise well across projects. Transformer-based systems such as TOGA [9] lifted the ceiling on assertion generation by learning directly from paired code–assertion corpora, but their outputs remained closed-box and required standard developer review before use.

The language-model era (2023–2026) introduces a qualitatively different set of challenges. First, hallucination: Ouédraogo et al. [14] report substantial hallucination rates in unfavourable zero-shot configurations, and hallucinated API calls or non-compilable code are the leading reason tool-augmented and multi-agent pipelines have emerged. Second, data contamination: because widely used benchmarks such as Defects4J were published long before the training cut-offs of modern LLMs, benchmark-only evaluations may over-estimate the true generalisation capability of the technique. In our pool, only 4 of the 143 studies actively mitigate this risk. Third, prompt-sensitivity and closed-model opacity: LLM behaviour depends on prompt phrasing, temperature, and model version, none of which are always reported in enough detail to reproduce. Fourth, the separation between oracle-for-implementation and oracle-for-specification becomes especially relevant when the model is prompted with buggy code and produces an oracle that is correct with respect to the current implementation but not with respect to the intended behaviour. These four challenges are largely aligned with the eight challenges identified

by Molina et al. [6].

What has not changed across the three eras is the evaluation practice: the metrics and the reporting conventions inherited from the transformer era are still applied largely unchanged to a setting in which the model, the prompt, and the benchmark can all vary between two runs.

4.3. Evaluation practice

The benchmarks used to evaluate the generated oracles are concentrated on a small number of well-known subjects. Defects4J appears in 14 papers, Apache Commons in 3, and SF110, QuixBugs, and HumanEval in 2 each. The remaining studies use a variety of domain-specific or industrial datasets, single-project case studies, or ad-hoc collections of GitHub repositories. Only about 15% of the pool uses an established benchmark, and even within this subset the chosen subset of Defects4J or SF110 varies study by study. Direct cross-study comparison of oracle effectiveness is therefore difficult, and reproducibility relies more on the release of code and prompts than on a shared benchmark.

This concentration on a small set of well-known benchmarks also raises a concern that is specific to the language-model era. Defects4J, for example, was first released in 2014 and is widely available online, which means that parts of the benchmark may have been encountered during the training of modern language models. This does not automatically invalidate results, but it weakens the interpretation of evaluations that rely only on widely available benchmark projects. In our pool, data contamination is explicitly discussed in 8 papers, and only 4 take an active mitigation step, such as building a leakage-mitigated dataset or evaluating on projects created after the likely model training cut-off [14]. We therefore treat contamination mitigation as an important reporting item for future evaluations of AI-generated oracles.

Across the 143 studies we observed that the metrics used to evaluate AI-generated oracles fall into three broad families, a grouping that also matches the categorisation used in recent work on oracle evaluation [15, 16]. The first category consists of mutation-grounded metrics, such as mutation score, which check whether the generated oracle is strong enough to detect injected faults. Hossain et al. [16, 11] argue, at least for assertion-style oracles, mutation-grounded evaluation is more defensible than textual similarity, because it directly measures whether the generated oracle can detect injected faults. Yet in our pool it is reported in 31 of 143 studies (21.7%). The second category consists of accuracy and F1-style metrics, often computed against a reference oracle written by a developer. The third category consists of NLP-style similarity metrics, such as BLEU, ROUGE, and BERTScore, which were originally imported from the natural-language processing literature. As shown by Shin et al. [15], these metrics can be misleading for oracle quality: a high BLEU score does not imply a high mutation-killing power, and the ranking of techniques can change substantially when the evaluation switches from textual similarity to mutation grounding.

The reporting of statistical tests is uneven. Only 11 of the 143 studies (7.7%) report any explicit test of statistical significance. Among those that do, the Wilcoxon or Mann-Whitney family is the most common (4 papers), followed by the t-test or ANOVA family (3), Vargha-Delaney effect size (1), Kruskal-Wallis or Friedman tests (1), correlation tests such as Spearman and Kendall (1), and Z- or binomial tests (1). Taken together, these numbers suggest that the methodological ambition of the field, in terms of multi-model, multi-prompt, multi-benchmark designs, has grown faster than its reporting standards.

Figure 4 summarises the four main dimensions of evaluation practice across the pool.

A related and often overlooked question is which behaviour the generated oracle is meant to describe in the first place. When a model is prompted with the source code of a method that already contains a defect, it may produce an oracle that is correct with respect to the current implementation, but not with respect to the intended specification. In that case, the assertion will pass on the buggy code and the fault will remain hidden. We believe this distinction is important for evaluation design, since accuracy against a developer-written reference oracle does not, on its own, tell us whether the technique under study is generating an oracle for the implementation or for the specification.

The picture that emerges from this synthesis is twofold. On the one hand, AI-based approaches

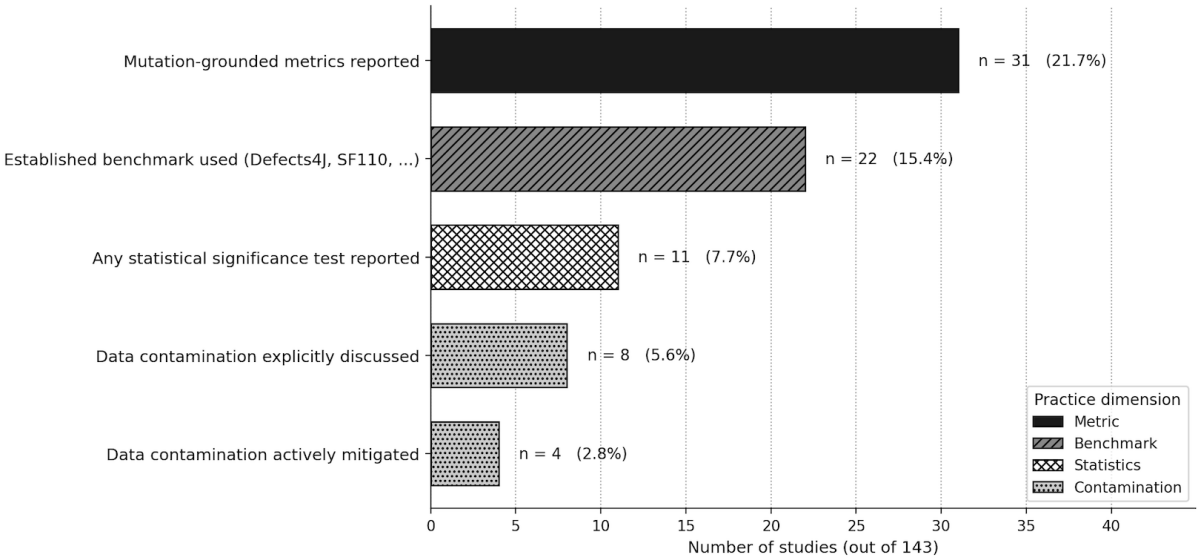


Figure 4: Reporting of evaluation practice across the 143 primary studies, summarised along four dimensions: the metric family used to score oracles, the use of established benchmarks, statistical reporting, and explicit treatment of data contamination.

have made oracle generation considerably more accessible and have produced a rich design space of prompting and validation strategies. On the other hand, the way in which these approaches are evaluated has not always followed the same pace, and several of the basic properties that one would expect from an oracle, such as oracle strength under mutation, robustness to data contamination, the separation between implementation and specification, and statistical significance of improvements, are still reported by a minority of the studies.

5. Conclusion and Future Work

This paper presented a focused systematic mapping based on an ongoing systematic literature review of AI-based test oracle generation. We synthesised a core pool of 143 primary studies published up to the retrieval cut-off in early May 2026 and organised the results around three questions: which types of oracles AI-based approaches generate, which AI/ML techniques and input artefacts they use, and how the generated oracles are evaluated.

According to the results of our review, AI-based oracle generation is now dominated by the language-model family, which accounts for roughly 39% of the full pool and for the large majority of recent publications. The shift is sharply localised in time: the language-model family represented only 7.2% of work up to and including 2022 but 68.9% of work from 2023 onward, against a 3.5× acceleration in publishing volume over the same period. This boundary coincides in time with the broader public availability of instruction-tuned LLMs, although establishing a causal link is outside the scope of this paper. However, LLMs have not introduced a fundamentally new oracle category. Metamorphic relations, assertion-style oracles, and expected-output oracles account for approximately 84% of the literature, and the main novelty of the LLM era is methodological rather than conceptual. Composite prompting designs, in particular the combination of few-shot prompting, chain-of-thought, and validate-fix loops, appear to be the most effective empirical setting in the current pool, and the more recent reasoning models tend to raise the achievable correctness further.

The synthesis also shows that evaluation practice has not always kept pace with methodological ambition. Only 21.7% of the studies report a mutation-grounded metric, fewer than 8% report a statistical significance test, and only a small minority actively mitigates data contamination. Human involvement is also limited: interactive, in-loop refinement is reported in only 3 of the 143 studies, while humans are more often involved at design time through prompt or example curation, or after generation through

validation and correction.

The synthesis presented in this paper points to several directions for future research on AI-based test oracle generation. The first direction concerns the input artefact. Only 8 of the 143 primary studies in our pool use formal specifications as the primary input to the oracle generator, and only 17 use natural-language requirements, while 70 studies rely on source code. We observed in Section 4.3 that a model prompted with the source code of a method that already contains a defect may produce an oracle that is correct with respect to the current implementation, but not with respect to the intended specification. A promising direction is therefore to investigate how natural-language requirements, API documentation, and acceptance criteria can be used as the primary input, possibly in combination with the source code, and how composite prompting designs (few-shot examples, chain-of-thought, and validate-fix loops) can deliver such intent artefacts to the model without losing coverage.

The second direction concerns how generated oracles are evaluated. A useful next step is to improve evaluation practice by reporting stronger and more comparable evidence. This includes using mutation-grounded metrics where appropriate, reporting false-positive rates and oracle strength, applying statistical tests and effect sizes for multi-model or multi-prompt comparisons, and evaluating on projects or benchmark variants that were not available at the likely time of model training. It also includes reporting the cost of generation, for example in tokens, execution time, and developer validation effort. Such reporting would make it easier to compare techniques across studies and would reduce the risk that progress is measured only through textual similarity or pass-rate improvements.

The third direction concerns the role of the human in the pipeline. We observed that interactive, in-loop refinement is reported in only 3 of the 143 studies, while the human is most often confined to design time (through the curation of prompt templates) or to post-hoc validation. A direction worth exploring is a targeted in-loop design in which the developer is asked to resolve a specific kind of ambiguity, namely whether the oracle under generation should describe the current implementation or the intended specification. The developer's answer could then serve both as additional context for the model and as ground truth for a mutation-grounded evaluation. We believe that this targeted form of human involvement is more realistic than a full code-review loop and more useful than no human involvement at all.

Beyond these three directions, this paper provides an entry point to the broader ongoing review, which investigates further research questions related to tooling and integration, author-reported threats to validity, targeted software qualities, adjacent testing tasks, and detailed LLM design choices. These questions will be addressed in the full version of the review.

Acknowledgments

The authors acknowledge financial support from the Slovenian Research and Innovation Agency (Research Core Funding No. P2-0057).

6. Declaration on Generative AI

During the preparation of this work, the authors used generative AI tools to support language editing and to generate Figures 1–4 from data provided by the authors. The authors reviewed, corrected, and validated the generated text and figures, and take full responsibility for the content of the publication. The plotted data are derived from the authors' extracted review dataset.

References

[1] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, S. Yoo, The oracle problem in software testing: A survey, IEEE Transactions on Software Engineering 41 (2015) 507–525. doi:10.1109/TSE.2014.2372785.

[2] E. J. Weyuker, On testing non-testable programs, The Computer Journal 25 (1982) 465–470. doi:10.1093/comjnl/25.4.465.
[3] M. Harman, P. McMinn, M. Shahbaz, S. Yoo, A Comprehensive Survey of Trends in Oracles for Software Testing, Technical Report CS-13-01, University of Sheffield, Department of Computer Science, 2013.
[4] M. Pezzè, C. Zhang, Automated test oracles: A survey, in: Advances in Computers, volume 95, Elsevier, 2014, pp. 1–48. doi:10.1016/B978-0-12-800160-8.00001-2.
[5] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. H. Tse, Z. Q. Zhou, Metamorphic testing: A review of challenges and opportunities, ACM Computing Surveys 51 (2018) 4:1–4:27. doi:10.1145/3143561.
[6] F. Molina, A. Gorla, M. d'Amorim, Test oracle automation in the era of LLMs, ACM Transactions on Software Engineering and Methodology 34 (2025) Article 150, 24 pages. doi:10.1145/3715107.
[7] C. Watson, M. Tufano, K. Moran, G. Bavota, D. Poshyvanyk, On learning meaningful assert statements for unit test cases, in: Proceedings of the 42nd International Conference on Software Engineering (ICSE), 2020, pp. 1398–1409. doi:10.1145/3377811.3380429.
[8] M. Tufano, D. Drain, A. Svyatkovskiy, N. Sundaresan, Generating accurate assert statements for unit test cases using pretrained transformers, in: Proceedings of the 3rd ACM/IEEE International Conference on Automation of Software Test (AST), 2022, pp. 54–64. doi:10.1145/3524481.3527220.
[9] E. Dinella, G. Ryan, T. Mytkowicz, S. K. Lahiri, TOGA: A neural method for test oracle generation, in: Proceedings of the 44th International Conference on Software Engineering (ICSE), 2022, pp. 2130–2141. doi:10.1145/3510003.3510141.
[10] M. Schäfer, S. Nadi, A. Eghbali, F. Tip, An empirical evaluation of using large language models for automated unit test generation, IEEE Transactions on Software Engineering 50 (2024) 85–105. doi:10.1109/TSE.2023.3334955.
[11] S. B. Hossain, M. B. Dwyer, Togll: Correct and strong test oracle generation with llms, 2025. URL: https://doi.org/10.1109/ICSE55347.2025.00098. doi:10.1109/ICSE55347.2025.00098.
[12] V. Garousi, N. Joy, Z. Jafarov, A. B. Keleş, S. Değirmenci, E. Özdemir, R. Zarringhalami, AI-powered software testing tools: A systematic review and empirical assessment of their features and limitations, arXiv preprint arXiv:2409.00411 (2024). URL: https://arxiv.org/abs/2409.00411. doi:10.48550/arXiv.2409.00411, multivocal Literature Review of 55 AI-powered test automation tools.
[13] S. Afshan, P. McMinn, M. Stevenson, Evolving readable string test inputs using a natural language model to reduce human oracle cost, in: 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation (ICST), 2013, pp. 352–361. doi:10.1109/ICST.2013.11.
[14] W. C. Ouédraogo, A. K. Kaboré, Y. Li, H. Tian, A. Koyuncu, J. Klein, D. Lo, T. F. Bissyandé, Prompt engineering in llms for automated unit test generation: A large-scale study, Empirical Softw. Engg. 31 (2026). URL: https://doi.org/10.1007/s10664-026-10840-4. doi:10.1007/s10664-026-10840-4.
[15] J. Shin, H. Hemmati, M. Wei, S. Wang, Assessing evaluation metrics for neural test oracle generation, IEEE Transactions on Software Engineering 50 (2024) 2337–2349. doi:10.1109/TSE.2024.3433463.
[16] S. B. Hossain, A. Filieri, M. B. Dwyer, S. Elbaum, W. Visser, Neural-based test oracle generation: A large-scale evaluation and lessons learned, in: Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), 2023, pp. 120–132. doi:10.1145/3611643.3616265.
[17] B. Kitchenham, S. Charters, Guidelines for Performing Systematic Literature Reviews in Software Engineering, Technical Report EBSE-2007-01, Keele University and Durham University Joint Report, 2007.
[18] M. J. Page, J. E. McKenzie, P. M. Bossuyt, et al., The PRISMA 2020 statement: an updated guideline for reporting systematic reviews, BMJ 372 (2021) n71. doi:10.1136/bmj.n71.
[19] M. Méndez, A. Becerra-Terón, J. M. Almendros-Jiménez, M. G. Merayo, M. Núñez, Combining metamorphic testing and machine learning to enhance openstreetmap, IEEE Transactions on Reliability 73 (2024) 1834–1848. doi:10.1109/TR.2024.3379366.

[20] C. Xu, S. Chen, J. Wu, S.-C. Cheung, V. Terragni, H. Zhu, J. Cao, Mr-adopt: Automatic deduction of input transformation function for metamorphic testing, in: Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, ASE '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 557–569. URL: https://doi.org/10.1145/3691620.3696020. doi:10.1145/3691620.3696020.

[21] P. C. Cañizares, P. Gómez-Abajo, E. Guerra, J. de Lara, Towards metamorphic testing with llm-based workflows: Metamorphic relation inference and follow-up test case generation, Information and Software Technology 196 (2026) 108150. URL: https://www.sciencedirect.com/science/article/pii/S0950584926001394. doi:10.1016/j.infsof.2026.108150.

[22] S. R. Shahamiri, W. M. N. W. Kadir, S. bin Ibrahim, An automated oracle approach to test decision-making structures, in: 2010 3rd International Conference on Computer Science and Information Technology, volume 5, 2010, pp. 30–34. doi:10.1109/ICCSIT.2010.5563989.

[23] J. Ayerdi, V. Terragni, A. Arrieta, P. Tonella, G. Sagardui, M. Arratibel, Generating metamorphic relations for cyber-physical systems with genetic programming: an industrial case study, in: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2021, Association for Computing Machinery, New York, NY, USA, 2021, p. 1264–1274. URL: https://doi.org/10.1145/3468264.3473920. doi:10.1145/3468264.3473920.

[24] A. Gartziandia, J. Ayerdi, A. Arrieta, S. Ali, T. Yue, A. Agirre, G. Sagardui, M. Arratibel, Microservices for continuous deployment, monitoring and validation in cyber-physical systems: an industrial case study for elevators systems, in: 2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C), 2021, pp. 46–53. doi:10.1109/ICSA-C52384.2021.00014.