Re-evaluating Detection of Equivalent Mutants using LLMs: We Should Properly Measure How Far We Are

ARJUN TANDON*, Indraprastha Institute of Information Technology Delhi, India
MEHMET FIRAT DÜNDAR*, Sabancı University, Turkiye
MILKIYAS GEBREMICHAEL GEBRU*, Addis Ababa University, Ethiopia
DARKO MARINOV, University of Illinois Urbana-Champaign, USA
YILING LOU, University of Illinois Urbana-Champaign, USA
WENXI WANG, University of Virginia, USA

Mutation testing is a widely used approach for measuring test-suite quality. A critical problem in mutation testing is equivalent mutant detection (EMD), i.e., determining if a mutant semantically behaves the same as the original code despite some syntactic differences. A recent study has shown that LLM-based EMD techniques hold great promise, reporting substantial improvements over traditional compiler- and machine-learning–based approaches. In this work, we revisit those recent results and evaluate the generalization capabilities of the proposed LLM-based EMD techniques across two additional datasets that differ from the prior dataset in mutation operators, programming languages, or source projects. Contrary to prior findings, the proposed LLM-based EMD techniques suffer substantial performance degradation on the two additional datasets. Through an extensive analysis, we identify a key factor underlying the differences as *original-method–level data leakage* (i.e., the same original method appearing in both training and test sets), indicating that prior results under *within-method evaluation* do not generalize to *cross-method evaluation*. We find that the studied LLMs tend to rely on a method-wise majority-voting shortcut rather than reasoning about the semantic effects of mutations. Based on these findings, we call for the adoption of realistic cross-method evaluation and the development of mutation-centric semantic reasoning in future LLM-based EMD research.

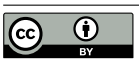
CCS Concepts: • **Software and its engineering** → **Software testing and debugging**; • **General and reference** → *Empirical studies*.

Additional Key Words and Phrases: Equivalent Mutant Detection, Mutation Testing, Large Language Models, Software Testing

ACM Reference Format:
Arjun Tandon, Mehmet Fırat Dündar, Milkiyas Gebremichael Gebru, Darko Marinov, Yiling Lou, and Wenxi Wang. 2026. Re-evaluating Detection of Equivalent Mutants using LLMs: We Should Properly Measure How Far We Are. *Proc. ACM Softw. Eng.* 3, ISSTA, Article ISSTA159 (October 2026), 23 pages. https://doi.org/10.1145/3832250

*These authors contributed equally to this work.

Authors' Contact Information: Arjun Tandon, Indraprastha Institute of Information Technology Delhi, New Delhi, India, arjun22095@iiitd.ac.in; Mehmet Fırat Dündar, Sabancı University, Istanbul, Turkiye, firat.dundar@sabanciuniv.edu; Milkiyas Gebremichael Gebru, Addis Ababa University, Addis Ababa, Ethiopia, milkiyasgebru@gmail.com; Darko Marinov, University of Illinois Urbana-Champaign, Urbana, USA, marinov@illinois.edu; Yiling Lou, University of Illinois Urbana-Champaign, Urbana, USA, yilingl@illinois.edu; Wenxi Wang, University of Virginia, Charlottesville, USA, wenxiw@virginia.edu.



ACM 2994-970X/2026/10-ARTISSTA159
https://doi.org/10.1145/3832250

1 Introduction

Mutation testing [19] is a powerful and widely studied technique for assessing the effectiveness of test suites. It works by systematically injecting small syntactic changes into a program to create *mutants* and measuring whether existing tests can distinguish the mutants from the original program. If a test fails, the mutant is considered *killed*; if the tests still pass, the mutant has *survived*. The mutation score is calculated as the proportion of killed mutants among all mutants, and a test suite with a higher mutation score is generally considered more effective at detecting real faults. Due to its fault-revealing capability and close connection to real programming errors, mutation testing has been adopted in both academic research and industrial practice [3, 15, 35, 36], and has served as a foundational technique for evaluating test adequacy [5] and studying software quality.

Despite its effectiveness, mutation testing suffers from a long-standing challenge: *equivalent mutants*. An equivalent mutant is syntactically different from the original program but semantically indistinguishable for all possible test inputs. Since no test can kill an equivalent mutant, its presence artificially lowers mutation scores, introduces noise into evaluation results, and wastes substantial computational resources. More importantly, the inability to distinguish equivalent mutants from killable ones restricts the reliability of mutation testing as a test adequacy metric. As a result, detecting and filtering equivalent mutants has been recognized as a critical problem for making mutation testing practical, scalable, and trustworthy [15, 35].

To date, a large number of automated equivalent mutant detection (EMD) techniques have been proposed. Given a pair of original code (e.g., a method) and its corresponding mutant as input, EMD techniques determine whether the mutant is equivalent to the original code. Earlier techniques leverage compiler optimizations [4, 33] or constraint solving [31] to identify equivalent mutants, but suffer from limited effectiveness [29, 34]. Later work [6, 34] proposes training machine-learning or neural-network models for EMD.

More recently, large language models (LLMs) have emerged as a promising solution for EMD. Owing to their strong code reasoning capabilities, LLMs show great potential for directly determining whether a mutant preserves the behavior of its original program. In particular, Tian et al. [41] (referred to as *Tian-24* throughout this paper) present a comprehensive empirical study of LLM-based EMD techniques. Their results suggest that LLM-based approaches substantially outperform traditional EMD techniques, to quote from their RQ1 result: "LLM-based techniques significantly surpass all ten EMD baselines in detecting equivalent mutants, with average F1 score improvements of 75.18% for Compiler-based techniques, 19.14% for ML-based techniques, and 12.75% for Tree-based NN techniques."

While Tian-24 demonstrates promising results for applying LLMs to EMD, its evaluation is confined to a single dataset and programming language, leaving the generalizability of LLM-based EMD techniques across diverse codebases and mutant distributions largely unexplored. To fill this gap, our work revisits Tian-24 to investigate whether their reported effectiveness of LLM-based EMD truly extends beyond its original evaluation settings. Specifically, we systematically evaluate their LLM-based approaches under both *reproduction* and *replication* settings. Following the ACM definitions [2], *reproduction* refers to obtaining the same results using the authors' original artifacts under the same experimental setup, while *replication* refers to obtaining the same results using independently developed artifacts. Accordingly, we conduct a reproduction study on their dataset and a replication study on two additional datasets to assess the reproducibility and generalizability of their experimental setup. Our study is organized as follows.

Reproduction Study (RQ1): Can we reproduce the high performance of LLM-based EMD techniques reported in prior work on the Tian-24 dataset? We first conduct a reproduction study to verify the main findings (i.e., the high performance of LLM-based EMD techniques) reported

by Tian-24 on their Java dataset, which we call *the Tian-24 dataset*. Starting from the authors' released implementation and dataset, we successfully reproduce the reported results, observing consistently high effectiveness comparable to that reported in the Tian-24 paper. This experiment confirms that our scripts and experimental settings align with those of the Tian-24 work and establishes a reliable reference point for our subsequent replication experiments.

Replication Study (RQ2): How well do existing LLM-based EMD techniques proposed by Tian-24 generalize to additional datasets? We next evaluate the generalizability of LLM-based EMD techniques proposed by Tian-24 (which we call *Tian-24 LLM-based EMD techniques)* on two additional EMD datasets with different characteristics, i.e., from "Equivalent Mutants in the Wild" (for short *EMIW*) [23] and LLMorpheus [43]. Both are independent, human-labeled datasets constructed in prior EMD studies. Unlike the Tian-24 dataset, EMIW and LLMorpheus are derived from different mutation frameworks and different real-world projects; moreover, LLMorpheus targets JavaScript programs. *Across both datasets, all proposed Tian-24 LLM-based EMD techniques exhibit substantial performance degradation compared to the Tian-24 dataset.* For example, CodeT5 with the pre-trained strategy achieves an F1 score of 80.52% on the Tian-24 dataset, but its performance drops to 47.18% on EMIW and 46.75% on LLMorpheus. *Our replication results reveal significant generalization limitations in Tian-24 LLM-based EMD techniques.*

Factor Analysis (RQ3): What factors contribute to the generalization gap observed between datasets? We conduct a factor analysis to investigate the causes of the observed generalization gap. Our analysis focuses on Tian-24 and EMIW, as both target the same programming language (Java). Specifically, we examine whether the performance drop from the Tian-24 dataset to EMIW can be attributed to differences in (1) data duplication between training and test sets, (2) input length, and (3) mutation operator distributions. However, even after aligning the two datasets along these dimensions, we still observe a substantial performance gap, indicating that these factors are not the primary drivers of the generalization issue in Tian-24 LLM-based EMD techniques.

We further analyze the distribution of original methods (i.e., the code before mutation) in the Tian-24 dataset. We found that mutants derived from the same original method frequently appear in both the training and test sets: 98.11% of original methods are represented in both. We call this overlap *original-method–level data leakage*.

We then examine whether the original-method–level data leakage contributes to the high performance of Tian-24 LLM-based EMD techniques on the Tian-24 dataset. We call the Tian-24 evaluation setting *within-method evaluation* (i.e., mutants from the same original method may appear in both training and test sets). We introduce a stricter *cross-method evaluation*, which enforces a disjoint separation of original methods across data splits and potentially better reflects realistic mutation testing scenarios. We re-split the Tian-24 dataset accordingly and re-train and evaluate Tian-24 LLM-based EMD techniques, eliminating original-method–level data leakage.

Notably, we find that all Tian-24 LLM-based EMD techniques suffer substantial performance degradation in cross-method evaluation, and their earlier advantages over traditional EMD approaches largely vanish. *This finding indicates the prior gains of Tian-24 LLM-based EMD techniques reported in Tian-24 are largely driven by within-method evaluation*, where models tend to memorize method-specific semantics rather than learn a generalizable capability for equivalent mutant detection. Moreover, we find most original methods exhibit strongly polarized mutant equivalence rates in the Tian-24 dataset, which encourages models to adopt a *method-wise majority-voting shortcut* (i.e., the behavior in which a model appears to bypass mutation-relevant semantic reasoning and instead predicts each test mutant using the dominant training label associated with its original method), rather than reason about the semantic impact of individual mutations. This shortcut explains both the high performance observed under within-method evaluation and the sharp performance degradation under cross-method evaluation.

While our findings expose the limitations of Tian-24 LLM-based EMD techniques under cross-method evaluation, they also offer important guidance for future research, highlighting both the necessity of adopting realistic cross-method evaluation and the importance of developing mutation-centric semantic reasoning rather than relying on method-specific memorization.

To check whether these findings extend beyond the models studied in Tian-24, we additionally evaluate two recent LLMs, DeepSeek-V3.2 and GPT-5.4 mini, under both reasoning and non-reasoning modes. In within-method evaluation, these models still underperform the fine-tuned techniques from Tian-24. In the cross-method evaluation, however, their reasoning variants substantially outperform all techniques proposed by Tian-24.

In summary, this paper makes the following contributions:

(1) We conduct a reproduction study of the prior work Tian-24 and successfully validate the consistently high performance of their LLM-based EMD techniques on the Tian-24 dataset.
(2) We conduct a replication study to evaluate the generalization of Tian-24 LLM-based EMD techniques on two additional EMD datasets. We find that the performance drops substantially when evaluated beyond the Tian-24 dataset.
(3) We conduct a systematic analysis to investigate the factors contributing to this performance degradation. Our results show that the original-method–level data leakage in the Tian-24 dataset substantially affects the performance of the Tian-24 LLM-based EMD techniques.

2 Background and Related Work

2.1 Mutation Testing

Mutation testing [19] is a widely studied and effective technique for evaluating test-suite quality. It operates by (1) systematically introducing small syntactic modifications into a program to create mutants and (2) checking whether the existing tests can distinguish the mutated program from the original. A mutant is killed if at least one test fails, and survives if all tests continue to pass. The test-suite effectiveness is commonly measured using the mutation score, defined as the proportion of killed mutants among all generated mutants. Higher mutation scores are generally associated with stronger fault-detection capability. Owing to its strong correlation with real programming faults and its fault-revealing power, mutation testing has become a foundational technique for assessing test adequacy and software quality.

Mutation Strength. Mutation testing evaluates mutants using weak, firm, or strong mutation. Weak mutation considers a mutant killed when its execution state diverges from that of the original program immediately after the mutated statement [17], while firm mutation checks for divergence at a later intermediate point [50]. Strong mutation requires the mutation's effect to propagate to an observable program output [19]. We focus on strong mutation because all benchmarks evaluated in this work—Tian-24, EMIW, and LLMorpheus—adopt strong mutation, so our choice preserves their label semantics and enables direct comparison with prior work.

2.2 Equivalent Mutants

A fundamental problem in mutation testing is the existence of equivalent mutants, which are syntactically different from the original program but semantically identical [19]. As a result, equivalent mutants cannot be killed by any test. Equivalent mutants adversely affect a wide range of mutation-based software engineering tasks. For example, Meta employs an LLM agent to filter likely equivalent mutants before mutation-based test generation [15], while MBFL-FLIM suppresses the suspiciousness scores of equivalent mutants to improve mutation-based fault localization [25]. Recent work has also explored generating mutants that are known to be semantically equivalent or non-equivalent by construction for evaluating analysis tools [30, 48]. These studies demonstrate the promise of LLMs for handling equivalent mutants under specific experimental settings. However,

the extent to which their reported performance depends on dataset construction, data splitting strategies, and train–test overlap remains largely unexplored.

2.3 Equivalent Mutant Detection (EMD)

EMD is a long-standing problem that has been studied since the beginning of mutation testing itself [10], including recent work from the last few years [9, 23, 41]. For example, the MUPPAAL framework removes useless (equivalent and duplicate) mutants and has techniques for avoiding, reducing, and removing equivalent mutants [9].

Earlier work by Baldwin and Sayward [4] proposed compiler optimization techniques for EMD. Similarly, Papadakis et al. [33] proposed the Trivial Compiler Equivalence technique for EMD, classifying as equivalent any mutant that compiles to the same machine code as its original code. Offutt and Pan [31] proposed EMD using constraint-based testing, which uses constraints to represent the test conditions under which a mutant is killed. Therefore, if a test case kills a mutant, the constraint system is true, and for equivalent mutants, the constraint system is infeasible.

With the advancement of machine learning, Junior et al. [6] proposed to apply seven machine learning algorithms for EMD. Further, Peacock et al. [34] introduced a tree-based neural-network EMD technique using an Abstract Syntax Tree Neural Network model. Though they improve on traditional compiler- or constraint-based techniques, existing ML-based techniques exhibit suboptimal effectiveness across diverse code structures [41].

More recently, large language models (LLMs) have been explored as an alternative for EMD given their powerful code reasoning capabilities. To analyze ChatGPT's ability to understand dynamic code behaviors, Ma et al. [28] conducted an experiment with 200 mutants, asking ChatGPT to determine the equivalent mutants. Tian et al. [41] conducted a more comprehensive empirical study evaluating the effectiveness of different LLM models and learning strategies on a dataset containing 3,302 Java mutants.

Our study focuses on revisiting the evaluation of Tian et al. [41] by first reproducing their experiments on the same dataset and then replicating their experimental pipelines on two additional datasets. Our replication results reveal the limited generalization of their proposed LLM-based EMD techniques across datasets, and our factor analysis shows that the high performance of LLMs on the Tian-24 dataset stems from original-method–level data leakage.

3 Reproduction Study

We first conduct a reproduction study to verify whether we can repeat the results (i.e., the high performance of LLM-based EMD techniques) reported by Tian-24 on their dataset. By confirming the reproducibility, we ensure consistency with the implementation and experimental settings of the original work, thereby establishing a reliable reference point for subsequent comparisons.

3.1 Reproduction Setup

Baselines. We focus on re-assessing the top LLM-based EMD techniques introduced in Tian-24, so we do not include older baselines (e.g., compiler-based, ML-based, or TNN-based techniques). For LLM-based EMD baselines, we study all eight open-source models included in Tian-24. We exclude the two commercial OpenAI models (i.e., GPT-3.5 Turbo [7] and OpenAI text embedding models [32]) from Tian-24, because opaque model updates can change their behavior and thereby limit reproducibility. Instead, we include several recent API-based models in our replication study (Section 6). The following lists the models we studied:

(1) *Encoder-Only:* CodeBERT (110M) [12], GraphCodeBERT (110M) [14].
(2) *Encoder-Decoder:* PLBART (210M) [1], CodeT5 (210M) [47], UniXCoder (110M) [13], CodeT5+ (6B) [46], StarCoder (7B) [24].
(3) *Decoder-Only:* Code Llama (7B) [38].

Each model can be applied for EMD via different learning or prompting strategies. We include all five strategies from Tian-24:

(1) *Pre-trained Code Embedding* [37]: This strategy freezes the LLM parameters and trains only a multilayer perceptron (MLP) classifier.
(2) *Fine-tuned Code Embedding* [11, 16]: This strategy simultaneously updates the parameters of both the LLM encoder and the MLP-based classifier.
(3) *Zero-shot Prompting* [27]: This strategy uses a prompt without examples.
(4) *Few-shot Prompting* [27]: This strategy prompts the model with a set of examples.
(5) *Fine-tuning with Instruction* [49]: This strategy trains the model on instruction-filled mutant pairs to acquire specific task knowledge.

Dataset. Tian-24 constructs its dataset based on MutantBench [44, 45]. Specifically, the Tian-24 dataset contains 3,302 Java method-level data items from MutantBench. Each item includes the original method, its mutant, and a ground-truth label indicating whether the mutant is equivalent to the original method. The Tian-24 dataset is split into training and test sets using a 50/50 ratio, with the test set also used as the validation set. Although we recognize that using the test set for validation is suboptimal, we follow the original Tian-24 experimental setup to ensure faithful reproduction. Accordingly, our reproduction study reuses the Tian-24 dataset with its original training–(validation–)test split.

Metrics. In this study, we focus on the effectiveness of EMD techniques. Tian-24 reports macro-averaged precision, recall, and F1 score; for space reasons, we show only macro-averaged F1 score. F1 score is the harmonic mean of precision (the fraction of mutants predicted as equivalent that are actually equivalent) and recall (the fraction of truly equivalent mutants that are correctly identified). The macro-averaged F1 score is computed by first calculating F1 independently for each class (i.e., equivalent and non-equivalent) and then taking the unweighted average across both classes.

Implementation. We directly adopt the implementation of all baselines and metric calculation released in the Tian-24 artifacts [41, 42]. For fine-tuning, we followed Tian-24 exactly: (1) for encoder-based models (e.g., UniXCoder, GraphCodeBERT, StarCoder), we used an encoder and MLP classifier framework and fine-tuned both the encoder and classifier end-to-end. Each original-mutant method pair was tokenized, encoded into embeddings, and classified as equivalent or not. The training used AdamW with linear warmup and binary cross-entropy loss. The best checkpoint was selected based on validation F1; (2) for the decoder-only Code Llama model, we used supervised instruction fine-tuning with LoRA [18]. Each example was formatted as an instruction-response pair, and the model predicted "yes" or "no" for semantic equivalence. LoRA was applied to the attention projection layers (q_proj, k_proj, v_proj, and o_proj) with rank 16 and alpha 16, using TRL's SFTTrainer with fp16 precision and gradient checkpointing.

When starting our reproduction, we encountered several challenges that required manual intervention. First, the underspecified dependencies in their `requirements.txt` file led to compatibility issues between their provided code and the most recent library versions. A notable example occurred with GraphCodeBERT, where a version-related conflict in the underlying model architecture prevented the processing of code sequences exceeding a standard length limit. We implemented a targeted modification in the model's forward pass logic to correctly handle token-type identification, allowing the model to function across varying input sizes. Second, we resolved several environment issues, such as outdated resource paths and specific parser compatibility requirements.

Hardware Configuration. Tian-24 reports its experimental configuration as a server with 512 GB system RAM, Ubuntu 20.04.6, and two NVIDIA A800 GPUs. In our reproduction and subsequent experiments, we use a server equipped with two NVIDIA RTX PRO 6000 Blackwell Server Edition GPUs (96 GB GDDR7 VRAM each), which is comparable in scale to the Tian-24 setup.

Table 1. Reproduction and replication results (F1 scores) of Tian-24 LLM-based EMD techniques for different detection strategies and models from Tian-24.

Strategy	Model	Tian-24 [41]	Reproduction	Replication	
				EMIW	LLMorpheus
Pre-trained	CodeBERT (110M)	69.20	69.20	47.18	46.75
	GraphCodeBERT (110M)	80.52	77.43	47.18	46.75
	PLBART (210M)	80.52	80.52	47.18	46.75
	CodeT5 (210M)	80.52	80.52	47.18	46.75
	UniXCoder (110M)	81.88	81.88	47.18	46.75
	CodeT5+ (6B)	81.88	81.88	47.18	49.21
	StarCoder (7B)	81.69	80.52	47.18	46.75
Fine-tuned	CodeBERT (110M)	83.87	86.30	61.81	63.75
	GraphCodeBERT (110M)	85.18	86.74	58.44	63.37
	PLBART (210M)	85.42	85.31	59.01	65.23
	CodeT5 (210M)	84.37	85.99	58.18	61.48
	UniXCoder (110M)	86.58	84.53	57.36	59.73
	CodeT5+ (6B)	85.59	81.88	64.08	68.89
	StarCoder (7B)	81.88	78.07	47.18	46.75
Zero-shot prompting	Code Llama (7B)	48.04	48.04	52.36	48.43
Few-shot prompting	Code Llama (7B)	47.76	46.85	51.10	52.58
Fine-tuned with instruction	Code Llama (7B)	56.79	66.26	48.97	46.33

3.2 Reproduction Results

Columns "Tian-24 [41]" and "Reproduction" in Table 1 report the results from Tian-24 and our reproduction study. *Overall, our reproduction results closely match those reported in Tian-24.* Specifically: (1) For the pre-trained code embedding strategy, we observe identical F1 scores for most models (e.g., CodeBERT, PLBART, CodeT5, UniXCoder, and CodeT5+). The only exceptions are GraphCodeBERT and StarCoder, which exhibit slightly lower F1 scores in our reproduction (by 3.09 and 1.17 points, respectively). (2) For the fine-tuned code embedding strategy, our reproduction achieves comparable or slightly better performance across most models, with F1 differences relative to Tian-24 ranging from −3.81 (StarCoder) to +2.43 (CodeBERT). (3) For both zero-shot and few-shot prompting strategies, our reproduction yields identical or slightly lower performance compared to Tian-24. (4) For the instruction fine-tuning strategy, Code Llama performs better in our reproduction than reported in Tian-24, with an F1 improvement of 9.47 points. One possible reason is a training configuration issue in the Tian-24 artifact. The provided epoch argument is not actually used by the training script, so the model effectively trains for only one epoch, yielding an F1 score of 47.76%. After fixing the script to use the correct epoch parameter and training for 10 epochs as intended, Code Llama's performance increased to an F1 score of 66.26%.

> **Finding 1:** Our reproduction study successfully validates the main findings of Tian-24. Across all evaluated strategies, we observe performance largely consistent with the Tian-24 results, with only minor variations for a few models. These results confirm the reproducibility of Tian-24 and establish a reliable baseline for our replication and generalization analyses.

4 Replication

After successfully reproducing the Tian-24 results, we further evaluate the generalizability of these techniques through independent replications on two additional datasets.

4.1 Replication Datasets

We include two EMD datasets, i.e., EMIW [23] and LLMorpheus [43].

4.1.1 Dataset Construction. We describe how we process the two datasets for our replication study. **EMIW** [23] is an EMD dataset of mutants generated using the *Major* mutation framework [20] on seven Java projects from Defects4J [21]. The authors of EMIW manually labeled all mutants as equivalent or non-equivalent. Following Tian-24, our study focuses on method-level EMD, but the original EMIW dataset provides only line-level code information and does not include complete original or mutated methods. To enable method-level evaluation, we extend EMIW by collecting full method-level code for both original and mutated versions. Specifically, we configure the *Major* mutation framework to log the complete mutated Java files during mutant generation, rather than logging only line-level metadata. For both the original program and the mutant, we extract the enclosing method in which the mutation occurs; if the mutation is not in a method, we try to extract the enclosing class instead. If there is no enclosing class, we take the entire source file instead. The extracted original program is unmodified, whereas the extracted mutant contains the applied mutation. The resulting dataset contains 1,988 unique data items (1,965 methods, 22 classes, 1 file), each consisting of an original-mutant method pair along with a binary equivalence label. (From the 1,992 mutants in EMIW, we removed 3 duplicates and our script could not extract 1 mutant.)
LLMorpheus [43] is an EMD dataset for JavaScript, representing a different programming language from the Tian-24 dataset. LLMorpheus includes both JavaScript and TypeScript mutants generated using (1) the StrykerJS mutation framework [22, 39] and (2) an LLM-based mutant generation approach. The authors' artifact contains information on 954 manually labeled mutants. From the 954 mutants, we focus exclusively on the 687 JavaScript mutants, as the remaining 267 TypeScript mutants are not supported by several EMD baselines studied in Tian-24 (e.g., UniXCoder and GraphCodeBERT). From the artifact, we reconstruct the mutants and extract the source program and mutant methods as for the EMIW dataset, except that we look for the enclosing function rather than a method. The resulting processed dataset contains 671 unique data items (610 functions, 61 files), each consisting of an original–mutant pair and a binary equivalence label. (From the 687 mutants in LLMorpheus, 10 are duplicates, 3 do not exist, and our script could not extract the remaining 3. We reported two problems to the LLMorpheus authors who confirmed both.)

Following the Tian-24 methodology, we split both the EMIW and LLMorpheus into 50/50 training and test sets, and ensured that each subset contained 50% of the equivalent mutants.

4.1.2 Dataset Statistics. Table 2 compares key statistics of the three datasets: the Tian-24 dataset [41] (used for reproduction), EMIW [23], and LLMorpheus [43] (both for replication). For brevity, we will use "methods" as a general term for code units. Notably, the Tian-24 dataset contains a substantially higher number of mutants per original method (62.30 on average) than EMIW (1.67) and LLMorpheus (3.81). The high mutant density per method in the Tian-24 dataset induces stronger within-method correlations during training, which may lead models to overfit to method-specific patterns rather than learn generalizable mutation semantics.

Table 3 shows the source projects and the distribution of equivalent mutants across projects. The three datasets have no project overlap. Notably, eight projects in the Tian-24 dataset (e.g., Defroster) consist entirely of equivalent mutants (100%), while EMIW has no such extreme case, and its proportion of equivalent mutants ranges from 4.66% (Chart) to 19.71% (Gson). In summary, the two additional EMD datasets, EMIW and LLMorpheus, provide complementary perspectives to the Tian-24 dataset by exhibiting more diverse distributions of equivalent mutants across original methods and projects.

4.2 Replication Pipeline

We adopt the same experimental pipeline as in our reproduction study (Section 3.1), including baselines, metrics, implementation, and hardware configuration, with the only difference being

Table 2. Dataset statistics summary. "Original Methods": number of distinct original methods mutated; "Total Mutants": all mutant instances before deduplication; "Unique Mutants": distinct mutants after deduplication; "Avg/Max Mut./Meth.": average and maximum mutants generated per original method.

Dataset	Split	Original Methods	Total Mutants	Unique Mutants	Avg Mut./Meth.	Max Mut./Meth.	Equiv. (%)
Tian-24	Train	52	1,652	1,584	31.77	337	250 (15.13)
	Test	53	1,650	1,574	31.13	366	249 (15.09)
	Combined	53	3,302	3,059	62.30	703	499 (15.11)
EMIW	Train	694	994	994	1.43	42	106 (10.66)
	Test	696	994	994	1.43	45	106 (10.66)
	Combined	1,188	1,988	1,988	1.67	87	212 (10.66)
LLMorpheus	Train	124	335	335	2.70	26	40 (11.94)
	Test	128	336	336	2.63	19	41 (12.20)
	Combined	176	671	671	3.81	45	81 (12.07)

Table 3. Projects in the Tian-24 dataset, EMIW, and LLMorpheus.

Tian-24 [41]					EMIW [23]				
Project	**Total**	**Equiv.**	**Non-Eq.**	**Equiv. %**	**Project**	**Total**	**Equiv.**	**Non-Eq.**	**Equiv. %**
StringTokenizer	755	51	704	6.75	Chart-1f	429	20	409	4.66
Triangle	703	46	657	6.54	Collections-28f	360	20	340	5.56
ArrayUtils	487	12	475	2.46	Gson-18	340	67	273	19.71
QuickSort	319	12	307	3.76	Csv-16f	292	31	261	10.62
Bisect	193	31	162	16.06	Lang-1f	251	44	207	17.53
Defroster	143	143	0	100.00	JxPath-22f	242	21	221	8.68
MathUtils	127	21	106	16.54	Math-1f	74	9	65	12.16
Vector3D	120	7	113	5.83					
WordUtils	106	13	93	12.26	**LLMorpheus [43]**				
BubbleSort	103	20	83	19.42	**Project**	**Total**	**Equiv.**	**Non-Eq.**	**Equiv. %**
XmlFriendlyN…	69	23	46	33.33	Complex.js	100	7	93	7.00
Simulator	68	11	57	16.18	q	100	0	100	0.00
Profit	46	46	0	100.00	node-dirty	98	13	85	13.27
Insert	19	19	0	100.00	pull-stream	98	1	97	1.02
Day	14	14	0	100.00	crawler-url-parser	94	16	78	17.02
Bubble	9	9	0	100.00	plural	87	12	75	13.79
Min	9	9	0	100.00	countries-and-t…	35	18	17	51.43
Prime_num	7	7	0	100.00	node-jsonfile	34	10	24	29.41
Mid	5	5	0	100.00	image-downloader	25	4	21	16.00

that we replace the Tian-24 dataset with the EMIW and LLMorpheus datasets. In addition, for the few-shot prompting strategy, we maintain the same Java examples from the Tian-24 dataset when evaluating on EMIW; for LLMorpheus, we construct new JavaScript examples while preserving the equivalence ratio (i.e., three examples including two equivalent examples and one non-equivalent example) to maintain consistency across languages.

4.3 Replication Results

Columns "EMIW" and "LLMorpheus" in Table 1 present our replication results on the two additional datasets. *Overall, we observe that all Tian-24 LLM-based EMD techniques suffer substantial performance degradation on EMIW and LLMorpheus compared to their performance on the Tian-24 dataset.*

While the performance of the proposed techniques is impressive on the Tian-24 dataset (with F1 scores mostly above 80.00%), it drops dramatically when evaluated on the additional datasets.

Specifically, (1) for the pre-trained code embedding strategy, all the models exhibit a substantial performance drop on the two additional datasets, indicating poor generalization across datasets. For example, UniXCoder achieves 81.88% F1 on the Tian-24 dataset but drops dramatically to 47.18% on EMIW and 46.75% on LLMorpheus. Also, all seven models with the pre-trained strategy show the same F1 scores on EMIW: after our manual inspection, we find that they predict all mutants as non-equivalent in the test set. In fact, some models also show the same F1 scores in the original Tian-24 results. (2) For the fine-tuned code embedding strategy, all the models similarly show substantial degradation on the two additional datasets, e.g., UniXCoder drops to 57.36% on EMIW and 59.73% on LLMorpheus. (3) For the zero-shot and few-shot prompting strategies, conversely, we observe similar effectiveness across different datasets. Notably, Code Llama's prompting strategies' performance on EMIW and LLMorpheus slightly outperforms the performance on the Tian-24 dataset. On the Tian-24 dataset, these prompting strategies perform substantially worse than pre-trained or fine-tuned code embedding strategies; however, this performance gap diminishes on the two additional datasets, with prompting strategies even outperforming the latter in some cases. Overall, prompting strategies demonstrate significantly better generalization capabilities than pre-trained and fine-tuned strategies. (4) For the fine-tuning with instruction strategy, the performance of Code Llama also decreases markedly, dropping from 66.26% on the Tian-24 dataset to 48.97% on EMIW and 46.33% on LLMorpheus.

> **Finding 2:** Our replication study reveals that the effectiveness of Tian-24 LLM-based EMD techniques on the Tian-24 dataset does not generalize to new datasets. Across both EMIW and LLMorpheus, all the training-required strategies (i.e., pre-trained code embedding, fine-tuned code embedding, and fine-tuning with instruction) experience substantial performance degradation, indicating limited generalization across datasets. In contrast, the training-free strategies (i.e., zero-shot and few-shot prompting) exhibit consistent effectiveness across datasets, in some cases outperforming embedding-based approaches on the additional datasets. Overall, these results highlight substantial generalization issues for Tian-24 LLM-based EMD techniques.

5 Factor Analysis

We further perform a series of factor analyses to investigate why Tian-24 LLM-based EMD techniques suffer from generalization issues across different datasets. In particular, our analysis focuses on a direct comparison between Tian-24 and EMIW, as both datasets target the same programming language (Java). This comparison allows us to conduct more controlled experiments by eliminating confounding effects introduced by differences in programming languages. We focus on the pre-trained and fine-tuned code embedding approaches, as these represent the top Tian-24 LLM-based EMD techniques identified in Tian-24 yet exhibit severe generalization issues in our replication study. For each strategy, we evaluate all seven models, consistent with the settings used in our reproduction and replication studies.

5.1 Data Deduplication and Analysis

By analyzing the Tian-24 dataset, we identify a fraction of duplicate mutants (7.36%) between the training and test sets. In contrast, we followed best practice by constructing EMIW with deduplication, suggesting that such duplication in the Tian-24 dataset may inflate the performance of Tian-24 LLM-based EMD techniques relative to EMIW. To assess the effect of duplication, we conduct a comprehensive deduplication analysis by removing all duplicate data from the Tian-24 dataset and re-evaluating Tian-24 LLM-based EMD techniques on the deduplicated dataset.

5.1.1 Setup. We adopt the following deduplication process to identify and remove duplicate mutants in the Tian-24 dataset. (1) *Internal Deduplication:* We first identify and remove duplicate mutants within the training or test set, eliminating 68 duplicates from the training set and 76 from the test set. (2) *Cross-Set Deduplication:* We then identify 99 mutants that appear in both training and test sets, although Tian-24 [41, page 4] explicitly states "we confirmed that there is no data leakage between our training and test datasets through manual inspection." To maintain balanced equivalent mutation rates across both sets, we employ a greedy algorithm to selectively remove a duplicate from one or the other set. This process removes 56 additional mutants from the training set and 43 from the test set. Our deduplication strategy prioritizes maintaining similar equivalent mutation rates across both sets to ensure neither becomes biased toward equivalent or non-equivalent mutants. After deduplication, the Tian-24 dataset is reduced from 3,302 to 3,059 mutants, with 243 duplicate mutants removed. The equivalent mutation rate remains essentially unchanged after deduplication, at 15.71% for the training set and 15.74% for the test set.

5.1.2 Results. Tian-24 LLM-based EMD techniques are re-trained and re-evaluated on a deduplicated version of the dataset, Tian-24-dedup. Table 4 compares the performance on the original Tian-24 dataset (with duplicates) and Tian-24-dedup. *The results reveal minimal performance differences for all Tian-24 LLM-based EMD techniques between the original and deduplicated Tian-24 datasets.* Across all studied models, the average F1 score difference is less than 2%, with the maximum difference being 4.13% (fine-tuned StarCoder). For some models such as pre-trained CodeBERT and UniXCoder, performance even slightly increases on Tian-24-dedup, with F1 score increases of 0.47% and 0.71%, respectively. Despite removing 7.36% of the data, including 99 duplicates between training and test sets, models still maintain their high performance on the deduplicated dataset.

Finding 3: Removing duplicate mutants from the Tian-24 dataset has a negligible impact on the performance of Tian-24 LLM-based EMD techniques. Across all models, performance differences between the original and deduplicated datasets remain minimal, and high F1 scores are largely preserved. These results indicate that mutant duplication is not the primary factor driving the performance gap observed between the Tian-24 dataset and EMIW.

Table 4. Performance comparison: the Tian-24 dataset vs. the deduplicated Tian-24 dataset.

Strategy	Model	Tian-24 dataset	Tian-24-dedup
Pre-trained	CodeBERT (110M)	69.20	69.67
	GraphCodeBERT (110M)	77.43	78.71
	PLBART (210M)	80.52	81.20
	CodeT5 (210M)	80.52	81.20
	UniXCoder (110M)	81.88	82.59
	CodeT5+ (6B)	81.88	82.59
	StarCoder (7B)	80.52	81.20
Fine-tuned	CodeBERT (110M)	86.30	88.46
	GraphCodeBERT (110M)	86.74	87.87
	PLBART (210M)	85.31	87.17
	CodeT5 (210M)	85.99	82.59
	UniXCoder (110M)	84.53	84.56
	CodeT5+ (6B)	81.88	82.59
	StarCoder (7B)	78.07	82.20

5.2 Input Length Analysis

The input of each data item in the Tian-24 dataset and EMIW includes both the original method and a mutant generated from that method. After comparing the Tian-24 dataset and EMIW, we find that EMIW contains substantially longer inputs than the Tian-24 dataset. Specifically, EMIW has

an average input length of 3,774.97 characters (ranging from 105 to 136,727), whereas the Tian-24 dataset has a much shorter average input length of 1,756.99 characters (ranging from 119 to 7,616). LLMs may exhibit degraded performance on longer inputs [26]. To investigate whether input length contributes to the observed performance gap between the Tian-24 dataset and EMIW, we conduct an input-length analysis by constructing comparison groups with short and long inputs, and then re-evaluating Tian-24 LLM-based EMD techniques on these groups.

5.2.1 Setup. We partition the EMIW dataset into two equal-sized subsets based on input length using the following procedure. First, we sort all EMIW data items in ascending order of input length. We then split the dataset at the median into two equal halves: (1) *EMIW Short* contains the 994 shortest data items, with an average input length of 795.25 characters (min: 105, max: 1,598). This subset contains 9.66% equivalent mutants (96/994). (2) *EMIW Long* contains the 994 longest data items, with an average input length of 6,754.68 characters (min: 1,611, max: 136,727). This subset contains 11.67% equivalent mutants (116/994). Both EMIW Short and EMIW Long are further split into training and test sets using a 50/50 ratio. For both EMIW Short and EMIW Long, we re-train models on their training dataset and test them on their corresponding testing dataset.

5.2.2 Results. Table 5 compares effectiveness across different input lengths. *Overall, we observe no significant effectiveness differences in Tian-24 LLM-based EMD techniques between short and long inputs.* For the pre-trained code embedding strategy, all models perform similarly on EMIW Short and EMIW Long, with F1 score differences being only 0.56%. For the fine-tuned code embedding strategy, all models except CodeT5+ achieve better performance on EMIW Long than on EMIW Short. Note that the effectiveness of both pre-trained and fine-tuned strategies on EMIW Short and EMIW Long remains similar to their performance on the full EMIW dataset, and is substantially lower than their performance on the Tian-24 dataset.

Finding 4: Input length has little impact on the performance of Tian-24 LLM-based EMD techniques. Models exhibit comparable performance on short and long inputs under both pre-trained and fine-tuned code embedding strategies, and their performance on these subsets is still substantially lower than on the Tian-24 dataset. These findings indicate that input length does not explain the observed generalization limitations of Tian-24 LLM-based EMD techniques.

Table 5. Performance comparison across input lengths on EMIW.

Strategy	Model	EMIW Short	EMIW Long
Pre-trained	CodeBERT (110M)	47.46	46.90
	GraphCodeBERT (110M)	47.46	46.90
	PLBART (210M)	47.46	46.90
	CodeT5 (210M)	47.46	46.90
	UniXCoder (110M)	47.46	46.90
	CodeT5+ (6B)	47.46	46.90
	StarCoder (7B)	47.46	46.90
Fine-tuned	CodeBERT (110M)	57.25	57.80
	GraphCodeBERT (110M)	54.22	58.49
	PLBART (210M)	54.48	58.66
	CodeT5 (210M)	50.56	56.97
	UniXCoder (110M)	53.19	56.72
	CodeT5+ (6B)	63.26	56.58
	StarCoder (7B)	47.46	59.43

5.3 Mutation Operator Analysis

Given that the Tian-24 dataset and EMIW are generated with different mutation frameworks, they contain mutants produced by different sets of mutation operators. To investigate whether differences in mutation operators contribute to the performance gap between the Tian-24 dataset and EMIW, we conduct a mutation operator analysis by filtering EMIW to retain only mutants produced by mutation operators shared with the Tian-24 dataset, and then re-evaluating Tian-24 LLM-based EMD techniques on the filtered EMIW dataset.

Table 6. Mapping relation of mutation operators between the Tian-24 dataset and EMIW. "Coverage Details" explains which specific mutation operators are retained. Operators are split into three groups: Fully Covered, where all Major mutations have a mutation framework of the Tian-24 dataset equivalents (100% retention); Partially Covered, where only specific sub-operations are retained while others are filtered out; and Not Covered that have no comparable in the mutations from the Tian-24 dataset.

EMIW	Tian-24	Coverage Details
Fully Covered (100% - All mutants retained)		
AOR	AORB	All binary arithmetic operators: `+, -, *, /, %`
SOR	SOR	All shift operators: `<<, >>, >>>`
LOR	LOR	All bitwise logical operators: `&, |, ^`
ROR	ROR, ROD	All relational operators: `>=, >, <=, <, !=, ==, TRUE, FALSE`
Partially Covered (Specific sub-operations retained)		
COR	SEOR, SEOD	**Retained:** `&&` ↔ `||` replacements, LHS/RHS operand selection **Filtered out:** TRUE, FALSE, ==, != replacements
STD	ASDL, FSDL, AODS	**Retained:** Method call (`<CALL>`), assignment (`<ASSIGN>`), increment (`<INC>`), decrement (`<DEC>`) deletions **Filtered out:** Return (`<RETURN>`), break (`<BREAK>`), continue (`<CONTINUE>`) statement deletions
LVR	CR, CDL	**Retained:** Generic constant replacements (all literal types)
Not Covered (0% - All mutants filtered out)		
ORU	None	Unary operator replacement (+, -, ~) not available in mutation framework of Tian-24 dataset
EVR	None	Expression-to-default-value replacement not available in mutation framework of Tian-24 dataset

5.3.1 Dataset Construction. We first analyze and compare the mutation operators in the Tian-24 and EMIW datasets. The Tian-24 dataset involves 26 mutation operators, and EMIW involves 9 mutation operators. While their mutation operators are defined in different scopes, we analyze and summarize their mapping relation in Table 6. For space limits, the detailed definition of each mutation operator is in [40].

We then construct EMIW-EquivOp, a subset of EMIW dataset by filtering out those mutants which are generated by the mutation operators that do not overlap with those in the Tian-24 dataset. In particular, we apply the following filtering strategy to construct EMIW-EquivOp:

- **Fully Covered (100.00%):** We retain all mutants in EMIW that are generated from mutation operators (AOR, SOR, LOR, ROR).
- **Partially Covered:** We retain only mutants matching the specific sub-operations covered by the Tian-24 dataset based on their transformations:

  - **COR:** We keep mutants with transformations like <&&>:<||>, <LHS>, <RHS>, and remove mutants with <TRUE>, <FALSE>, <==>, <!=>
  - **STD:** We keep mutants with transformations <CALL>, <ASSIGN>, <INC>, <DEC>, and remove mutants with <RETURN>, <BREAK>, <CONTINUE>
  - **LVR:** We retain all mutants that cover constant replacement concepts, despite different implementation patterns.
- **Not Covered (0.00%):** We remove all EMIWmutants generated from these operators (ORU, EVR) given they are not mapped to any mutation operator in the Tian-24 dataset.

This filtering yields EMIW-EquivOp, which contains 1,596 mutants generated by seven mutation operators that overlap with those in the Tian-24 dataset. We further split EMIW-EquivOp into training and test sets using a 50/50 ratio, similar to the Tian-24 dataset.

Table 7. Performance on the EMIW-EquivOp test set after training on the training sets of Tian-24-dedup and EMIW-EquivOp.

Strategy	Model	Training set	
		Tian-24-dedup	EMIW-EquivOp
Pre-trained	CodeBERT (110M)	47.30	47.30
	GraphCodeBERT (110M)	49.43	47.30
	PLBART (210M)	50.31	47.30
	CodeT5 (210M)	48.10	47.30
	UniXCoder (110M)	47.06	47.30
	CodeT5+ (6B)	48.91	47.30
	StarCoder (7B)	47.23	47.30
Fine-tuned	CodeBERT (110M)	47.30	56.92
	GraphCodeBERT (110M)	49.55	59.16
	PLBART (210M)	59.78	58.33
	CodeT5 (210M)	47.23	57.49
	UniXCoder (110M)	47.26	55.85
	CodeT5+ (6B)	47.30	60.79
	StarCoder (7B)	46.84	47.30

5.3.2 Results. We evaluate Tian-24 LLM-based EMD techniques on the test set of EMIW-EquivOp under two training settings: (1) training on the training set of Tian-24-dedup and (2) training on the training set of EMIW-EquivOp. Table 7 reports the results for both settings. For the pre-trained code embedding strategy, F1 scores are relatively similar across both sets. For the fine-tuned strategy, all models except PLBART achieve higher F1 scores when trained on the EMIW-EquivOp set. In both settings, F1 scores remain significantly lower than those reported in Tian-24. Overall, these results indicate that differences in mutation operator sets do not explain the observed generalization gap across datasets. In both settings, *all studied Tian-24 LLM-based EMD techniques perform substantially worse on EMIW-EquivOp (with F1 scores ranging from approximately 47% to 60%) than on the Tian-24 dataset*, even when evaluation is restricted to mutants generated using the same operators as those in the Tian-24 dataset.

Finding 5: Aligning mutation operator sets does not resolve the generalization gap of Tian-24 LLM-based EMD techniques across different datasets. Their performance on EMIW-EquivOp remains substantially lower than on the Tian-24 dataset, even when evaluated on mutants generated by the same operators. These results indicate that differences in mutation operator sets are not a primary driver of the observed performance degradation between the Tian-24 dataset and EMIW.

5.4 Original-Method-Level Data Leakage

After ruling out dataset duplication, input length, and mutation operators as factors, we further analyze the distribution of original methods in the Tian-24 dataset and EMIW. We observe that 98.11% of original methods (52 out of 53) in the Tian-24 dataset appear in both the training and test sets, whereas this ratio is only 17.00% (202 out of 1,188) in EMIW. Moreover, 99.97% of mutants in the Tian-24 dataset belong to original methods that appear in both training and test sets, compared to only 42.96% in EMIW. The differences are driven by the substantially higher number of mutants per original method in the Tian-24 dataset than in EMIW (on average, 62.30 vs. 1.67, as shown in Table 2). As a result, the Tian-24 dataset has many more mutants derived from the same original methods in both the training and test sets, leading to more *original-method–level data leakage*. For example, in the Tian-24 dataset, a single original method has 170 mutants, with 89 assigned to the training set and 81 to the testing set. This overlap exposes models to the same original code structure during training and evaluation, thereby potentially inflating model performance.

We further investigate whether original-method–level data leakage contributes to the strong performance of Tian-24 LLM-based EMD techniques on the Tian-24 dataset. Specifically, we refer to the original evaluation setting used in Tian-24 as *within-method evaluation*, in which mutants derived from the same original method may appear in both the training and test sets. We then introduce a stricter evaluation protocol, termed *cross-method evaluation*, where mutants from the same original method are strictly separated and never appear in both train/test splits.

The within-method and cross-method evaluation settings are analogous to the within-project and cross-project evaluation settings commonly used in prior work on predictive mutation testing [51]. In practice, cross-method evaluation is both more realistic and more important for real-world mutation testing, as it is rarely feasible to obtain human-labeled equivalent and non-equivalent mutants from the same methods used for training.

5.4.1 Setup. To eliminate original-method–level data leakage in the Tian-24 dataset, we re-split it for cross-method evaluation by creating new train/test splits, called *cross-method*, with *no overlap of the original methods in the training and test sets*. We start from the deduplicated Tian-24 dataset by keeping its 3,059 unique mutants (from 3,302 in the Tian-24 dataset). In addition, to further analyze the impact of training and testing data ratio, we create multiple cross-method splits based on the number of mutants, including 50/50 (50% training, 50% testing) as in the Tian-24 setup, 70/30 (70% training, 30% testing), and 80/20 (80% training, 20% testing). We then train and evaluate all Tian-24 LLM-based EMD techniques on these splits. Table 8 summarizes the detailed statistics of these cross-method splits.

Table 8. Dataset specifics for cross-method splits.

Dataset	Split	Mutants	Percentage	Equivalent Mutants
Cross-method 50/50 Tian-24	Training	1,507	49.26%	235 (15.59%)
	Testing	1,552	50.74%	247 (15.91%)
Cross-method 70/30 Tian-24	Training	2,102	68.72%	356 (16.94%)
	Testing	957	31.28%	126 (13.17%)
Cross-method 80/20 Tian-24	Training	2,450	80.09%	388 (15.84%)
	Testing	609	19.91%	94 (15.44%)

5.4.2 Results. Table 9 presents the cross-method evaluation results of all Tian-24 LLM-based EMD techniques across all cross-method training-testing splits (50/50, 70/30, and 80/20).

Performance Decrease in Cross-Method Evaluation. The most prominent finding is the sharp performance decline observed when Tian-24 LLM-based EMD techniques are evaluated on unseen

Table 9. Cross-method evaluation on the Tian-24 dataset. "Within-Method" shows the original Tian-24 results where training and testing mutants share the same original methods.

Strategy	Model	Within-Method	Cross-Method		
		50/50	50/50	70/30	80/20
Pre-trained	CodeBERT (110M)	69.20	45.68	46.48	45.82
	GraphCodeBERT (110M)	77.43	45.68	54.15	51.44
	PLBART (210M)	80.52	45.68	64.40	67.28
	CodeT5 (210M)	80.52	45.68	64.82	67.28
	UniXCoder (110M)	81.88	48.55	61.81	63.77
	CodeT5+ (6B)	81.88	48.14	67.26	71.27
	StarCoder (7B)	80.52	62.12	68.15	71.27
Fine-tuned	CodeBERT (110M)	86.30	59.82	68.39	66.06
	GraphCodeBERT (110M)	86.74	65.60	72.34	70.94
	PLBART (210M)	85.31	58.98	70.20	66.42
	CodeT5 (210M)	85.99	61.88	76.63	78.21
	UniXCoder (110M)	84.53	66.86	72.79	71.27
	CodeT5+ (6B)	81.88	45.68	66.50	71.27
	StarCoder (7B)	78.07	62.12	56.84	64.95

original methods under cross-method evaluation. *This pronounced drop indicates that the previously observed generalization gap between the Tian-24 dataset and EMIW is largely driven by original-method–level data leakage.* Key observations include:

(1) *Consistent performance degradation.* All evaluated models exhibit a clear and consistent drop in performance under cross-method evaluation compared to within-method evaluation. For example, GraphCodeBERT, the top-performing fine-tuned strategy model, drops from an F1 score of 86.74% under the within-method setting to 65.60% under the 50/50 cross-method evaluation.

(2) *Breakdown of model rankings.* The relative performance rankings reported in Tian-24 no longer hold under cross-method evaluation. Under the within-method setting with the pre-trained strategy, UniXCoder and CodeT5+ are the top-performing models (81.88% F1). However, both suffer much steeper declines under 50/50 cross-method evaluation (dropping to 48–49% F1), while StarCoder emerges as the new relative top performer (62.12%) in this setting.

(3) *Potential loss of advantage over traditional baselines.* Even the key claim of Tian-24—that their LLM-based EMD techniques outperform traditional compiler-, ML-, and tree-based approaches—may not persist under cross-method evaluation. In Tian-24, traditional baselines achieve F1 scores of 39.31% and 50.80% (compiler-based), 61.61–72.15% (ML-based), and 70.00% (tree-based). Under 50/50 cross-method evaluation, pre-trained LLM-based models achieve only 45.68–62.12%, and fine-tuned models achieve 45.68–66.86% F1. Overall, the previously reported advantage of Tian-24 open-source-LLM-based EMD techniques may disappear when original methods are disjoint between the training and test sets (and duplicate mutants are removed from each set).

(4) *Limited benefit from more training data.* Although increasing the training data size improves performance under cross-method evaluation, it does not recover the high results reported in Tian-24 under within-method evaluation. For instance, even with an expanded 80/20 cross-method split, the best-performing model (fine-tuned CodeT5) reaches only 78.21% F1, which is still much lower than the Tian-24 result of 85.99% for the same model and strategy.

Finding 6: The prior gains of Tian-24 LLM-based EMD techniques reported in Tian-24 are largely driven by original-method–level data leakage, with models tending to memorize method-specific semantics rather than learn a generalizable capability for EMD. Under cross-method evaluation, where original methods are disjoint between the training and test sets, all Tian-24 LLM-based EMD techniques suffer substantial performance degradation, and the earlier

advantages over traditional EMD approaches potentially vanish. Moreover, increasing the amount of training data yields improvements but does not recover the high performance levels.

Method-Wise Majority-Voting in Tian-24. Given the large performance gap between the within-method and cross-method evaluation settings, we further investigate why the original-method–level data leakage substantially inflates model performance on the Tian-24 dataset. Specifically, we analyze the distribution of equivalent mutants and the prediction distributions across original methods. For an original method and a data subset, we define the *equivalence rate* as the percentage of the mutants in the subset that are labeled or predicted equivalent.

We analyze Tian-24-dedup under within-method evaluation. We use the fine-tuned strategy. Figures 1 and 2 plot each method's training equivalence rate on the x-axis against its test equivalence rate on the y-axis. Panels (a), (b), and (c) use ground-truth labels, CodeT5 predictions, and GraphCodeBERT predictions, respectively. Other models exhibit a similar trend (detailed results are available in our artifact [40]). The figures share coordinates but differ in bubble area: blue bubbles in Figure 1 count original methods, whereas orange bubbles in Figure 2 count their test mutants. Bubble areas are normalized within each figure (maxima: 22 methods and 250 mutants) and are not comparable across figures. Darker colors indicate higher training equivalence rates, the dashed diagonal marks equal training and testing rates ($y = x$), and the axes omit the observation-free interval $[40\%, 100\%)$.

Polarized Distribution of Equivalent Mutants. The training equivalence rates are highly polarized: of the 52 shared methods, 22 contain no equivalent mutants, 19 fall in the 0–20% range, 3 in the 20–40% range, and 8 contain only equivalent mutants. Thus, most methods in Tian-24 dataset have highly imbalanced label distributions.

Test data largely preserves this polarization: 51 of 52 methods (1,528 of 1,530 test mutants) remain on their training majority-label side. In panel (a), most bubbles cluster near the $y = x$ line, confirming closely matching training and testing distributions.

Biased Prediction Distribution. This polarization greatly affects the model predictions as shown in panels (b) and (c). CodeT5 predicts for every mutant (across all 52 methods) the majority label that the mutant's method has in the training set; this complete majority-label prediction covers all 1,530 mutants in the test set. GraphCodeBERT does so completely for 40 methods (76.9%) and 623 mutants (40.7%); the remaining 12 methods and 907 mutants receive mixed predictions but most from the majority label (94.9% vs. 5.1%).

These observations indicate that the models tend to follow a method-wise majority-vote pattern, where the majority label of an original method learned during training is applied to all of its mutants during test. When the same methods appear in both splits, method identity becomes a strong predictive feature. Models can therefore base their predictions on each method's training-label distribution rather than on the semantic differences between individual mutants. This explains both the inflated performance on the Tian-24 split and the sharp decline under cross-method evaluation, where no prior information is available for the methods in the test set.

Finding 7: The original-method–level data leakage inflates performance of Tian-24 LLM-based EMD techniques on the Tian-24 dataset because of the highly imbalanced distributions of method-wise equivalence rate. Most original methods exhibit strongly polarized mutant equivalence rates, which encourages models to adopt a method-level majority-voting label shortcut rather than reason about the semantic impact of individual mutations. As a result, models tend to predict the majority label associated with each method when method identity is shared across training and test sets. This shortcut explains both the artificially high performance

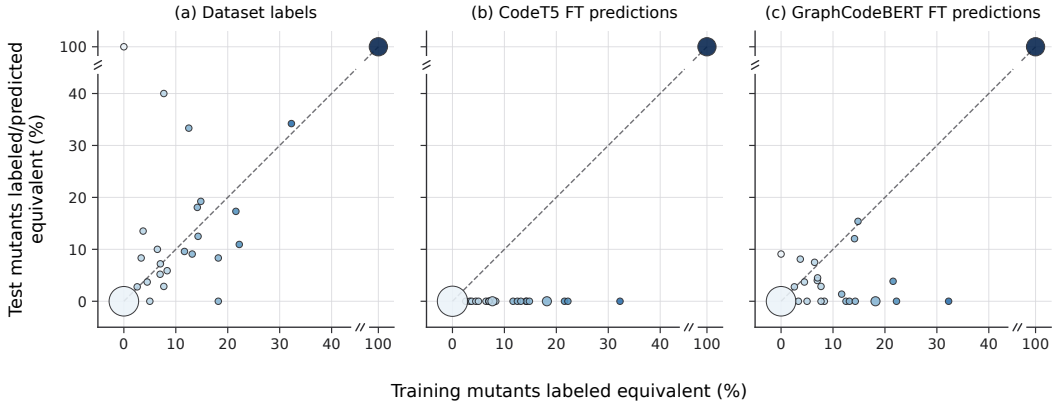


Fig. 1. Training equivalence rates versus test labels and model predictions on Tian-24-dedup. Bubble area represents the number of *original methods* at each coordinate.

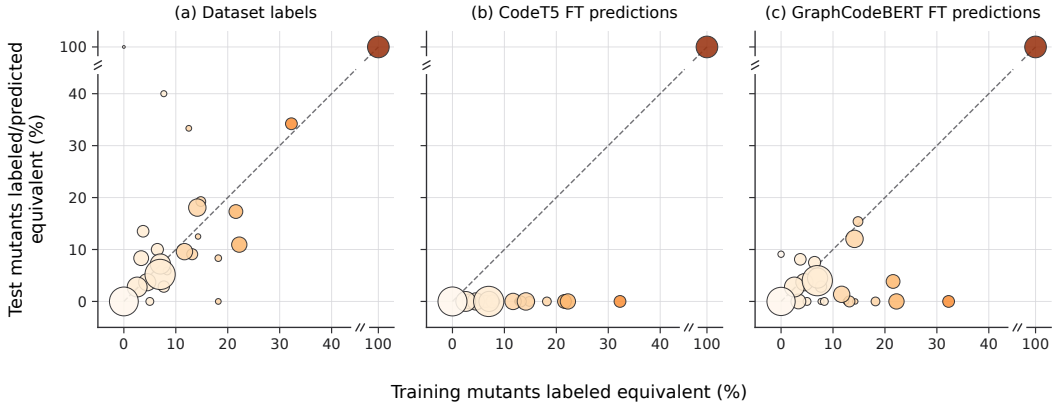


Fig. 2. Training equivalence rates versus test labels and model predictions on Tian-24-dedup. Bubble area represents the corresponding number of *test mutants*.

observed under within-method evaluation and the sharp performance degradation under cross-method evaluation, where such method-level cues are no longer available.

6 Discussion

Practical Implications. By reproducing prior results and extending the evaluation across additional datasets and stricter evaluation protocols, we show that the strong performance previously reported under within-method evaluation does not translate to settings that better reflect real-world use. Our analysis identifies original-method–level data leakage as the key factor inflating earlier results. Importantly, these findings should not be interpreted as a negative verdict on LLMs for equivalent mutant detection, but rather as guidance on how future research can more effectively harness their potential. The observed failures stem less from inherent model limitations than from evaluation regimes that allow shortcut learning, enabling models to rely on method-level regularities instead of reasoning about the semantic consequences of individual mutations. When these shortcuts are removed under cross-method evaluation, current approaches expose a clear gap between pattern recognition and mutation-level semantic reasoning, which should be the target for future LLM-based EMD research.

From a practical perspective, our results clarify when and how Tian-24 LLM-based EMD techniques can be expected to work. In settings where mutants are densely generated from a small number of methods, LLMs may still provide help. However, for realistic mutation testing workflows involving unseen methods and evolving codebases, robustness requires models, training objectives, and datasets that explicitly promote mutation-centric semantic reasoning rather than method-specific memorization. Promising directions include cross-method evaluation, mutation-aware representations, and contrastive learning across semantically equivalent and non-equivalent transformations. More broadly, this study underscores the importance of aligning evaluation methodology with intended deployment. Just as cross-project evaluation has become standard in predictive mutation testing, cross-method evaluation should become a baseline requirement for assessing LLM-based EMD. Adopting such protocols will enable the community to make more reliable progress and to distinguish genuine advances in semantic reasoning from artifacts of dataset construction.

Data Leakage in LLM-based EMD. While our results demonstrate that within-method evaluation suffers from *method-level data leakage*, another potential concern is *pre-training data leakage*, where LLMs may have been trained on some exact evaluation instances. However, we believe this risk is limited for most of the evaluated benchmarks. Specifically, for the EMIW and LLMorpheus benchmarks, although the original source projects are publicly available, the corresponding mutants are not directly released. In EMIW, mutants must be regenerated by applying the Major mutation framework [20] to the original code. In LLMorpheus, some multi-line mutants in the released file[1] are even incorrect, making it difficult to reconstruct the exact benchmark instances. Consequently, it is unlikely that the precise mutated code appears in LLM pre-training data unless the pre-training pipeline explicitly generates mutants using mutation frameworks. Furthermore, the equivalence labels are not available alongside the original projects, making it even less likely that LLMs have encountered the exact EMD tasks during pre-training.

The situation differs for the Tian-24 dataset, which is constructed from MutantBench. In this dataset, both the original and mutated code, together with the equivalence labels, are publicly available, making it possible that some LLMs have been exposed to these instances during training. Nevertheless, our experiments show that all evaluated older LLMs still perform poorly under the cross-method split (e.g., F1 scores below 50% with a 50/50 split), suggesting that—even with potential prior exposure—they possess only limited capability to reason about mutant equivalence.

Generality on Recent LLMs. In our main experiments, we follow the model selection criteria adopted in Tian-24. Given the rapid advancement of LLMs, it is also important to examine whether our findings generalize to more recent models. To this end, we additionally evaluate two recent LLMs, DeepSeek-V3.2 and GPT-5.4 mini, each under both reasoning and non-reasoning modes. Table 10 reports their effectiveness on the three datasets under the within-method and cross-method evaluation settings, respectively. Comparing these results with those in Table 9, we observe that these recent LLMs, using prompting-based inference, still underperform earlier fine-tuned models under the within-method setting. In contrast, under the cross-method setting, the reasoning variants of these recent LLMs substantially outperform all previously evaluated models and learning strategies, demonstrating significantly stronger cross-method generalization capabilities.

Randomness Analysis. Since both model training and inference involve inherent randomness, we further investigate whether stochasticity affects the validity of our findings by repeating the main fine-tuning experiments on the 50/50 cross-method dataset four additional times. The results exhibit low variability, with standard deviations ranging from 0.00 to 3.60, consistently confirming

[1]The released file containing the LLMorpheus mutants is available here: https://github.com/neu-se/mutation-testing-data/blob/main/manual-inspection/all-coded-mutants-final.csv

Table 10. Performance (F1 scores) of recent models on EMD for both *within-method evaluation* and *cross-method evaluation*.

Strategy	Model	Within-method	Cross-method		
		50/50	50/50	70/30	80/20
Recent LLMs	DeepSeek-V3.2 (No Reasoning)	46.62	48.48	48.74	46.79
	GPT-5.4 mini (No Reasoning)	51.37	52.99	57.09	55.30
	DeepSeek-V3.2 (Reasoning)	77.78	86.07	85.47	84.66
	GPT-5.4 mini (Low Reasoning)	75.17	84.02	84.43	82.63

our main finding that the evaluated models perform substantially worse under the cross-method setting than under the within-method setting.

7 Threats to Validity

External Validity. Our evaluation spans two Java datasets and one JavaScript dataset, which may limit generalization to other languages and codebases. Mutants are generated using mutation frameworks such as Major and StrykerJS, thus reflecting only their supported mutation types.
Construct Validity. We focus on open-source models due to cost constraints. Including proprietary API models may change absolute performance but is unlikely to affect the key cross-method generalization trend we observe. We also restrict our evaluation to the models studied in Tian-24 to ensure comparability of results, though newer models have since been introduced.
A further threat concerns the correctness of the ground-truth labels from MutantBench and the other benchmarks, which may be wrong to the point of invalidating some of our high-level findings. Following prior work, our experiments use the labels as they are. However, our inspection has already found cases where the mutant labels are definitely wrong; for example, we identified 9 mutants that have *both* equivalent and non-equivalent labels. Chung and Yoo [8] also raised potential bugs in the MutantBench dataset, and there does not appear to be a "MutantBench-Verified" version; we leave cleaning this dataset as future work for the community.

Internal Validity. We applied minor engineering modifications to the original artifact (e.g., dependency, logging, and tensor-parallel updates), which may introduce small implementation differences while preserving the overall experimental setup and training configuration.

8 Conclusion

This paper re-evaluates Tian-24 LLM-based EMD techniques for equivalent mutant detection through reproduction, replication, and factor analysis. While prior results can be reproduced under within-method evaluation, they do not hold under cross-method evaluation, which better reflects real mutation testing scenarios. We show that original-method–level data leakage is the main cause of inflated performance, whereas factors such as input length and mutation operators have little effect. These findings suggest that progress in LLM-based EMD requires moving beyond method-level shortcuts toward reasoning about the semantic impact of individual mutations. Achieving this will require cross-method evaluation, mutation-aware training objectives, and hybrid approaches that integrate LLMs with static or symbolic analysis, enabling practical EMD in real-world settings.

Acknowledgments

We would like to thank students from the UIUC++ SRSE (Summer Research in Software Engineering) 2024 and 2025 for their initial work on equivalent mutant detection.

Data Availability

All our data files and analysis scripts are publicly available [40].

References

[1] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified Pre-training for Program Understanding and Generation. In *NAACL*. doi:10.18653/v1/2021.naacl-main.211

[2] Association for Computing Machinery. 2026. Artifact Review and Badging – Definitions. Accessed: January 13, 2026. https://www.acm.org/publications/badging-terms

[3] Richard Baker and Ibrahim Habli. 2013. An Empirical Evaluation of Mutation Testing for Improving the Test Quality of Safety-Critical Software. *TSE* (2013). doi:10.1109/TSE.2012.56

[4] Douglas Baldwin and Frederick Sayward. 1979. *Heuristics for Determining Equivalence of Program Mutations.* Technical Report. Department of Computer Science, Yale University.

[5] Moritz Beller, Chu-Pan Wong, Johannes Bader, Andrew Scott, Mateusz Machalica, Satish Chandra, and Erik Meijer. 2021. What It Would Take to Use Mutation Testing in Industry—A Study at Facebook. In *ICSE-SEIP*. doi:10.1109/ICSE-SEIP52600.2021.00036

[6] Claudinei Brito, Vinicius H. S. Durelli, Rafael S. Durelli, Simone R. S. de Souza, Auri M. R. Vincenzi, and Marcio Eduardo Delamaro. 2020. A Preliminary Investigation into Using Machine Learning Algorithms to Identify Minimal and Equivalent Mutants. In *Mutation Workshop*. doi:10.1109/ICSTW50294.2020.00056

[7] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *NeurIPS*.

[8] Seungjoon Chung and Shin Yoo. 2022. Augmenting Equivalent Mutant Dataset Using Symbolic Execution. In *Mutation Workshop*. doi:10.1109/ICSTW55395.2022.00038

[9] Jaime Cuartas, David Cortés, Joan Betancourt, Jesús Aranda, Maxime Cordy, James Ortiz, Gilles Perrouin, and Pierre-Yves Schobbens. 2025. MUPPAAL: Efficient Elimination and Reduction of Useless Mutants in Real-Time Model-Based Systems. *STVR* (2025). doi:10.1002/stvr.1907

[10] R.A. DeMillo, R.J. Lipton, and F.G. Sayward. 1978. Hints on Test Data Selection: Help for the Practicing Programmer. *Computer* (1978). doi:10.1109/C-M.1978.218136

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL*. doi:10.18653/v1/N19-1423

[12] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *EMNLP*. doi:10.18653/v1/2020.findings-emnlp.139

[13] Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. UniXcoder: Unified Cross-Modal Pre-training for Code Representation. In *ACL*. doi:10.18653/v1/2022.acl-long.499

[14] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2021. GraphCodeBERT: Pre-training Code Representations with Data Flow. In *ICLR*. https://openreview.net/forum?id=jLoC4ez43PZ

[15] Mark Harman, Jillian Ritchey, Inna Harper, Shubho Sengupta, Ke Mao, Abhishek Gulati, Christopher Foster, and Hervé Robert. 2025. Mutation-Guided LLM-based Test Generation at Meta. In *FSE*. doi:10.1145/3696630.3728544

[16] Jeremy Howard and Sebastian Ruder. 2018. Universal Language Model Fine-tuning for Text Classification. In *ACL*. doi:10.18653/v1/P18-1031

[17] W.E. Howden. 1982. Weak Mutation Testing and Completeness of Test Sets. *TSE* (1982). doi:10.1109/TSE.1982.235571

[18] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *ICLR*. https://openreview.net/forum?id=nZeVKeeFYf9

[19] Yue Jia and Mark Harman. 2011. An Analysis and Survey of the Development of Mutation Testing. *TSE* (2011). https://doi.org/10.1109/TSE.2010.62

[20] René Just. 2014. The Major Mutation Framework: Efficient and Scalable Mutation Analysis for Java. In *ISSTA*. doi:10.1145/2610384.2628053

[21] René Just, Darioush Jalali, and Michael D. Ernst. 2014. Defects4J: A Database of Existing Faults to Enable Controlled Testing Studies for Java Programs. In *ISSTA*. doi:10.1145/2610384.2628055

[22] Jan-Jelle Kester. 2024. How mutation testing got practical. https://archive.fosdem.org/2024/schedule/event/fosdem-2024-2486-how-mutation-testing-got-practical/

[23] Benjamin Kushigian, Samuel J. Kaufman, Ryan Featherman, Hannah Potter, Ardi Madadi, and René Just. 2024. Equivalent Mutants in the Wild: Identifying and Efficiently Suppressing Equivalent Mutants for Java Programs. In *ISSTA*. doi:10.1145/3650212.3680310

[24] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Joel Lamy-Poirier, Joao Monteiro, Nicolas Gontier, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Ben Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason T Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Urvashi Bhattacharyya, Wenhao Yu, Sasha Luccioni, Paulo Villegas, Fedor Zhdanov, Tony Lee, Nadav Timor, Jennifer Ding, Claire S Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro Von Werra, and Harm de Vries. 2023. StarCoder: may the source be with you! *TMLR* (2023). https://openreview.net/forum?id=KoFOg41haE

[25] Hengyuan Liu, Zheng Li, Donghua Wang, Yankai Wu, Xiang Chen, and Yong Liu. 2025. FLIMs: Fault Localization Interference Mutants, Definition, Recognition and Mitigation. doi:10.48550/arXiv.2511.23302

[26] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the Middle: How Language Models Use Long Contexts. doi:10.48550/arXiv.2307.03172

[27] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *ACM Comput. Surv.* (2023). doi:10.1145/3560815

[28] Wei Ma, Shangqing Liu, Wang Wenhan, Qiang Hu, Ye Liu, Cen Zhang, Liming Nie, and Yang Liu. 2023. The Scope of ChatGPT in Software Engineering: A Thorough Investigation. doi:10.48550/arXiv.2305.12138

[29] Muhammad Rashid Naeem, Tao Lin, Hamad Naeem, and Hailu Liu. 2020. A machine learning approach for classification of equivalent mutants. *Journal of Software: Evolution and Process* (2020). doi:10.1002/smr.2238

[30] Elijah Nnorom, Md Basim Uddin Ahmed, Jiho Shin, Hung Viet Pham, and Song Wang. 2025. StaAgent: An Agentic Framework for Testing Static Analyzers. doi:10.48550/arXiv.2507.15892

[31] A. Jefferson Offutt and Jie Pan. 1997. Automatically detecting equivalent mutants and infeasible paths. *STVR* (1997). doi:10.1002/(SICI)1099-1689(199709)7:3<165::AID-STVR143>3.0.CO;2-U

[32] OpenAI. 2023. OpenAI Embeddings. https://platform.openai.com/docs/guides/embeddings. Accessed: January, 2026.

[33] Mike Papadakis, Yue Jia, Mark Harman, and Yves Le Traon. 2015. Trivial Compiler Equivalence: A Large Scale Empirical Study of a Simple, Fast and Effective Equivalent Mutant Detection Technique. In *ICSE*. doi:10.1109/ICSE.2015.103

[34] Samuel Peacock, Lin Deng, Josh Dehlinger, and Suranjan Chakraborty. 2021. Automatic Equivalent Mutants Classification Using Abstract Syntax Tree Neural Networks. In *Mutation Workshop*. doi:10.1109/ICSTW52544.2021.00016

[35] Goran Petrović, Marko Ivanković, Gordon Fraser, and René Just. 2022. Practical Mutation Testing at Scale: A view from Google. *TSE* (2022). doi:10.1109/TSE.2021.3107634

[36] Goran Petrović, Marko Ivanković, Gordon Fraser, and René Just. 2023. Please fix this mutant: How do developers resolve mutants surfaced during code review?. In *ICSE-SEIP*. doi:10.1109/ICSE-SEIP58684.2023.00019

[37] XiPeng Qiu, TianXiang Sun, YiGe Xu, YunFan Shao, Ning Dai, and XuanJing Huang. 2020. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences* (2020). doi:10.1007/s11431-020-1647-3

[38] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code Llama: Open Foundation Models for Code. doi:10.48550/arXiv.2308.12950

[39] Stryker Mutator. 2026. Stryker: Mutation Testing Framework. Accessed: January 24, 2026. https://stryker-mutator.io/

[40] Arjun Tandon, Mehmet Fırat Dündar, Milkiyas Gebru, Darko Marinov, Yiling Lou, and Wenxi Wang. 2026. Supplementary Material for Re-evaluating Detection of Equivalent Mutants Using LLMs: We Should Properly Measure How Far We Are. doi:10.6084/m9.figshare.32835686.v3

[41] Zhao Tian, Honglin Shu, Dong Wang, Xuejie Cao, Yasutaka Kamei, and Junjie Chen. 2024. Large Language Models for Equivalent Mutant Detection: How Far Are We?. In *ISSTA*. doi:10.1145/3650212.3680395

[42] Zhao Tian, Honglin Shu, Dong Wang, Xuejie Cao, Yasutaka Kamei, and Junjie Chen. 2024. Large Language Models for Equivalent Mutant Detection: How Far Are We? Accessed: January 24, 2026. https://github.com/tianzhaotju/EMD/

[43] Frank Tip, Jonathan Bell, and Max Schäfer. 2025. LLMorpheus: Mutation Testing Using Large Language Models. *TSE* (2025). doi:10.1109/TSE.2025.3562025

[44] Lars van Hijfte. 2021. *MutantBench: an Equivalent Mutant Problem Comparison Framework*. Master's Thesis. University of Amsterdam.

[45] Lars van Hijfte and Ana Oprescu. 2021. MutantBench: an Equivalent Mutant Problem Comparison Framework. In *Mutation Workshop*. doi:10.1109/ICSTW52544.2021.00015

[46] Yue Wang, Hung Le, Akhilesh Gotmare, Nghi Bui, Junnan Li, and Steven Hoi. 2023. CodeT5+: Open Code Large Language Models for Code Understanding and Generation. In *EMNLP*. doi:10.18653/v1/2023.emnlp-main.68

[47] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *EMNLP*. doi:10.18653/v1/2021.emnlp-main.685
[48] Anjiang Wei, Jiannan Cao, Ran Li, Hongyu Chen, Yuhui Zhang, Ziheng Wang, Yuan Liu, Thiago S. F. X. Teixeira, Diyi Yang, Ke Wang, and Alex Aiken. 2025. EquiBench: Benchmarking Large Language Models' Reasoning about Program Semantics via Equivalence Checking. In *EMNLP*. doi:10.18653/v1/2025.emnlp-main.1718
[49] Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. 2022. Finetuned Language Models are Zero-Shot Learners. In *ICLR*. https://openreview.net/forum?id=gEZrGCozdqR
[50] M.R. Woodward and K. Halewood. 1988. From weak to strong, dead or alive? An analysis of some mutation testing issues. In *Proceedings of the Second Workshop on Software Testing, Verification, and Analysis*. doi:10.1109/WST.1988.5370
[51] Jie Zhang, Lingming Zhang, Mark Harman, Dan Hao, Yue Jia, and Lu Zhang. 2019. Predictive Mutation Testing. *TSE* (2019). doi:10.1109/TSE.2018.2809496

Received 2026-01-30; accepted 2026-04-16