REPOEQUIV: BENCHMARKING LARGE LANGUAGE MODELS' UNDERSTANDING FOR REPOSITORY-LEVEL SEMANTIC EQUIVALENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) for code and agentic coding systems are rapidly improving at complex repository-level programming tasks. However, training and evaluating these systems largely rely on execution-based signals from unit tests, which are costly to obtain and difficult to scale across web-scale repositories and heterogeneous execution environments. We advocate a complementary approach: learning an LLM-based verifier for semantic equivalence that determines whether two pieces of code produce identical outputs for all valid inputs, thereby reducing the need for explicit execution. We introduce **RepoEquiv**, a benchmark and dataset suite for repository-level semantic equivalence constructed from repository-level code infilling datasets spanning both general-purpose and scientific programming domains. Unlike prior benchmarks focused on standalone functions with synthetic perturbations, RepoEquiv features repository-level context and dependencies, as well as realistic candidate code snippets generated by diverse coding LLMs. Evaluating 5 representative LLMs on RepoEquiv shows that semantic equivalence verification in repository contexts remains challenging for off-the-shelf models. Fine-tuning on RepoEquiv substantially improves verifier performance on general coding repositories, demonstrating the feasibility of learned semantic verifiers for scalable, execution-free evaluation, and yields uniform gains across all four scientific coding benchmarks, indicating promising out-of-domain generalization for learned verifiers. We further conduct calibration analysis and find that fine-tuning reduces calibration error, and we measure the resulting cost-accuracy trade-off directly: routing low-confidence cases to execution reaches 95% accuracy on the scientific benchmarks while executing 58% of cases, against 80% under confidence-agnostic routing. RepoEquiv establishes a foundation for developing learned semantic verifiers that can significantly reduce the cost of large-scale evaluation and learning for repository-level code generation.

1 INTRODUCTION

Large language models (LLMs) for code are rapidly advancing toward complex, repository-level software engineering tasks (SWE), with agentic systems operating over full codebases (Liu et al., 2026; Yang et al., 2024; Wang et al., 2025). A key driver of this progress is execution-based supervision, where candidate solutions are validated via execution and unit testing (Pan et al., 2025; Jain et al., 2025). While effective, this paradigm introduces a major scalability bottleneck: setting up and maintaining execution environments is costly and brittle, especially at web scale and in domains such as scientific computing with complex dependencies (Xiang et al., 2025; Kim et al., 2026). For example, scaling SWE tasks often requires sophisticated engineering efforts and relies on preexisting unit tests (Badertdinov et al., 2025; Pan et al., 2025), and scientific repository tasks are limited in scale due to the need of hand-crafting test cases and often take hours to evaluate on just hundreds of problems (Hua et al., 2025; Yan et al., 2025).

This paper explores a complementary direction, shown in Figure 1: learning LLM-based verifiers for semantic equivalence. Instead of executing programs to determine correctness, a semantic verifier predicts whether two pieces of code are functionally equivalent, that is, whether they produce identical outputs for all valid inputs. Such verifiers could serve as scalable proxies for execution-based

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

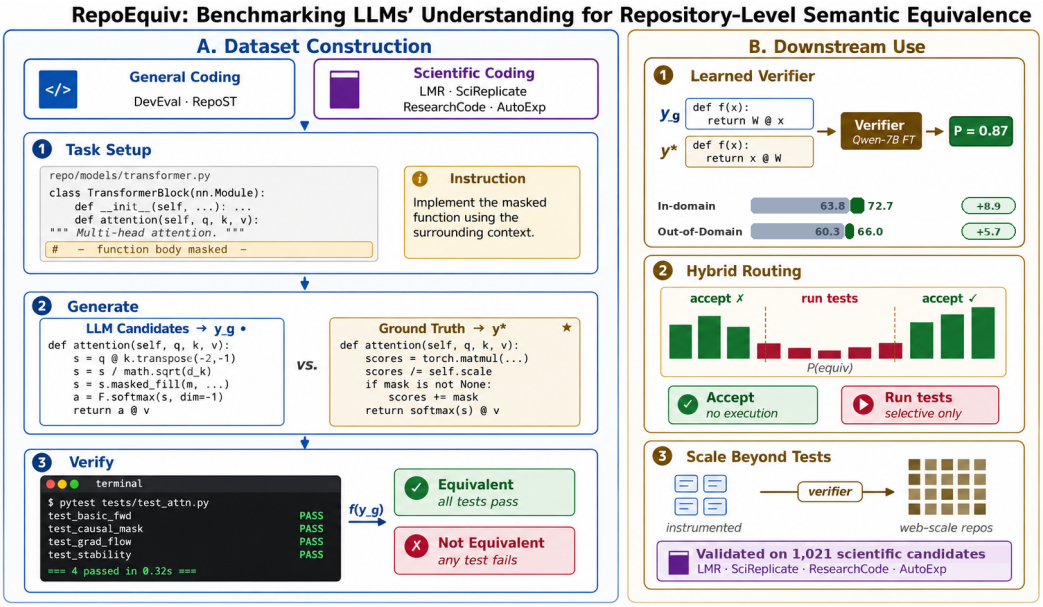


Figure 1: Overview of RepoEquiv, a suite of datasets and evaluation benchmarks for repository-level semantic equivalence. **Panel A** covers dataset construction: masking a target function and its instruction (A1), generating candidate infillings against the reference (A2), and labelling each candidate by unit test execution (A3). **Panel B** covers downstream use: the learned verifier and its in- and out-of-domain gains (B1), hybrid routing that accepts confident predictions and executes only the rest (B2), and scaling verification beyond the in-domain general coding repositories that the verifiers are trained on, to distant scientific repository coding domain (B3).

evaluation. In practice, execution-based supervision could be collected for a relatively small set of repositories to train a semantic verifier, which can then be applied to a much larger set without requiring explicit execution. This paradigm has the potential to significantly reduce the cost of large-scale evaluation and enable new learning pipelines that rely on learned correctness signals rather than expensive environment setup and program execution (Badertdinov et al., 2025; Pan et al., 2025).

Despite this promise, existing benchmarks for semantic equivalence do not reflect the realities of repository-level coding tasks. Prior datasets such as EquiBench (Wei et al., 2025) and SeqCoBench (Wei et al., 2025) evaluate models on equivalence checking for standalone functions or short code snippets generated through synthetic perturbations. Although useful, these settings omit key aspects of real-world programming. Determining whether two implementations are equivalent often requires reasoning about the repository-level context, including dependencies, helper modules, configuration files, and interactions across multiple files. Furthermore, the candidate solutions encountered in practice are rarely synthetic perturbations of the original code; they are typically generated by diverse coding LLMs and can differ substantially in structure and implementation strategy.

To address this gap, we introduce **RepoEquiv**, a benchmark and dataset suite for repository-level semantic equivalence verification, constructed as in Figure 1A. RepoEquiv is constructed from repository-level code infilling datasets spanning both general-purpose and scientific programming domains. Unlike prior benchmarks, RepoEquiv preserves repository context and dependencies, enabling equivalence judgments within realistic project structures. Moreover, the candidate code snippets in RepoEquiv are generated by a diverse set of coding LLMs, producing realistic variations that reflect the outputs encountered in modern coding workflows. This design allows RepoEquiv to capture the practical challenges of verifying LLM-generated code in complex repositories. Because every source benchmark ships executable tests, RepoEquiv also lets us quantify what those tests cost: verifying a single candidate by execution takes a median of 2.8 s on DevEval and 185 s on AutoExperiment, against roughly 0.2 s for one verifier call, a speed-up of 6× to 796× on five of the six benchmarks.

Using RepoEquiv, we perform a systematic evaluation of 5 representative LLMs as semantic equivalence verifiers. Our results show that determining the semantic equivalence of LLM-generated code in repository contexts remains a challenge for off-the-shelf LLMs, and that fine-tuning on RepoEquiv substantially improves verifier performance on general coding repositories, with the improvements transferrable to out-of-domain scientific coding tasks. We further perform a calibration analysis and find that fine-tuning reduces calibration errors, indicating that model confidence becomes more reliable after training. This enables the practical hybrid strategy of Figure 1B2, routing high-confidence cases to the learned verifier and deferring the rest to execution. We further measure the trade-off (Figure 3): confidence-based routing dominates confidence-agnostic routing baselines on all six benchmarks, and finetuning further widens the margin, improving the overall accuracy of the hybrid strategy. The gains extend to distant domains: in scientific codebases the fine-tuned verifier improves over its base on all four benchmarks, a uniform direction of transfer that points to promising out-of-domain generalization for learned verifiers.

Our contributions are as follows:

- We advocate semantic equivalence verification as a scalable alternative to execution-based evaluation for repository-level code generation.
- We introduce RepoEquiv, a benchmark and dataset suite for repository-level semantic equivalence that incorporates repository context and realistic candidate code snippets generated by diverse coding LLMs.
- We provide the first systematic evaluation of LLM-based semantic verifiers in repository settings, showing that fine-tuned verifiers can substantially improve over off-the-shelf models in terms of both performance and calibration. We further measure the cost-accuracy trade-off of the resulting hybrid strategy, which combines model-based verification with selective execution based on confidence.

2 Related Work

2.1 Repository-Level Code Generation

As LLMs have grown more capable at coding (Anthropic, 2026; DeepMind, 2025; OpenAI, 2026; Hui et al., 2024; Cao et al., 2026), benchmarks built on standalone functions (Austin et al., 2021; Chen et al., 2021) have become less reflective of realistic use. Effort has shifted to repository-level tasks: resolving GitHub issues (Jimenez et al., 2024), reproducing the computational experiments of research papers (Starace et al., 2025; Kim et al., 2026; Yan et al., 2025; Xiang et al., 2025; Hua et al., 2025), and infilling missing functions (Li et al., 2024; Xie et al., 2025). Because such benchmarks are small and serve only as testbeds, recent work scales executable repository collection so that unit-test feedback can act as a verifiable reward for finetuning (FAIR CodeGen team et al., 2025; Pan et al., 2025; Jain et al., 2025). That scaling is costly: it demands sophisticated environment construction and test synthesis per repository (Badertdinov et al., 2025; Xie et al., 2025), which is the bottleneck this work addresses.

2.2 LLM for Semantic Equivalence

Semantic (functional)[1] equivalence concerns whether two pieces of code produce the same output for any given input. With the increasing capability of LLMs for code, several equivalence-checking benchmarks have been proposed to evaluate and analyze LLMs' understanding of code semantics through the lens of functional equivalence (Maveli et al., 2025; Wei et al., 2025). In contrast to existing semantic equivalence benchmarks that are created for standalone functions and constructed from synthetic perturbations, RepoEquiv features 1) repository-level context, where determining equivalence requires the understanding of context dependency; 2) realistic equivalence checking pairs generated by a diverse set of LLMs. Moreover, while prior work on LLM for semantic equivalence checking merely as a way to evaluate LLMs' understanding of code semantics, RepoEquiv aims at learning strong equivalence verifiers that can serve as reward signals for scalable repository-level learning.

[1]We use semantic and functional equivalence interchangeably in this work

3 REPOEQUIV

3.1 REPURPOSE REPOSITORY-LEVEL CODE GENERATION FOR EQUIVALENCE VERIFICATION

We denote a general process of repository-level code generation as

$$y_g \sim \pi_g(I, C) \tag{1}$$

where π_g is the generation policy (e.g., LLM, agentic workflow) that takes as input the instruction I and repository-level context C and outputs the target code snippet y_g. Based on Equation 1, we formulate repository-level semantic equivalence verification as

$$y_v \sim \pi_v(I, C, y_g, y_g^*) \tag{2}$$

where given the task specification in I and C, the generation result y_g and the corresponding ground truth y_g^*, a verifier policy π_v produce a binary equivalence judgment y_v. In our setup, we constrain y_g^* to be a target function body that is masked out in C, and y_g is the candidate infilling for the function body.

Based on this formulation, we construct RepoEquiv by repurposing generation data into semantic equivalence verification data. In particular, given a generation data point (I, C, y_g^*, f), where $f : Y_g \to \{0, 1\}$ represents the unit testing, we collect one or more equivalence verification data points $(I', C', y_g, y_g^*, y_v^*)$ where each of the components is obtained as follows:

- **Instruction** I' and **context** C': we rewrite the generation instruction into a verification instruction, and combine *in-file* context, the source file with the target function masked out, with the *cross-file* dependency context supplied by the generation benchmark where available.
- **Candidate** y_g: following the original benchmark's setup (pure prompting or agentic generation), we produce repository-level infilling candidates via $y_g \sim \pi_g(I, C)$.
- **Label** y_v^*: obtained from unit test execution $f(y_g)$, equivalent if the snippet passes all provided tests and non-equivalent otherwise.

With the collected data, we instantiate the verification policy π_v through LLM prompting for both training and evaluation. For each data point, we concatenate (I', C', y_g, y_g^*) into a single prompt, and instruct the LLM to directly output "Yes" or "No" as the equivalence judgment.

3.2 SOURCE DATA

We source the generation data for RepoEquiv from the following benchmarks in both general and scientific coding domains:

- **General coding. DevEval** (Li et al., 2024) contributes 1,874 developer-annotated, test-evaluated samples from 117 real-world repositories. **RepoST** (Xie et al., 2025) builds sandboxed execution environments and supplies RepoST-Train (7,415 functions, 824 repositories) and RepoST-Eval (296 functions, 99 repositories).
- **Scientific coding. LMR-Bench** (Yan et al., 2025) covers 28 reproduction tasks from 23 NLP/LLM papers; **SciReplicate-Bench** (Xiang et al., 2025) 100 tasks from 36 NLP publications, each with a dedicated environment and verification suite; **ResearchCodeBench** (Hua et al., 2025) with 12 tasks from 20 recent ML papers; and **AutoExperiment** (Kim et al., 2026) of 85 essential functions masked from four reproducibility-challenge codebases.

3.3 DATA COLLECTION WITH DIVERSE MODEL GENERATIONS

Following the formulation in subsection 3.1, we repurpose repository-level code generation data into semantic equivalence verification data, and apply this process to all source benchmarks described in subsection 3.2. This procedure is used to construct both the training and evaluation splits of RepoEquiv.

For training, we focus on general coding benchmarks (DevEval and RepoST), where data is more abundant. We construct disjoint train/validation/test splits at the repository level, using an 80/10/10

Source	Cand.	Repos	Inst.	%Eq.
General Coding: training				
DevEval-train	2,528	73	1,138	50.0
RepoST-train	3,636	673	2,473	50.0
General Coding: evaluation (in-domain)				
DevEval-test	1,112	24	287	38.2
RepoST-test	717	83	246	36.5
Scientific Coding: evaluation (out-of-domain)				
ResearchCodeBench	590	18	173	41.7
SciReplicate-Bench	231	28	66	19.0
AutoExp	134	2	42	41.8
LMR-Bench	66	13	13	39.4

Table 1: Data statistics of RepoEquiv. **Cand.** is the number of candidate snippets in each split, **Repos** and **Inst.** are the distinct repositories and masked functions the candidate snippets are sourced from, respectively. **%Eq.** is the proportion of candidates that are labeled as equivalent. Every instance across in domain and out-of-domain tasks are sampled from the five generator models, with further filtering based on diversity and difficulty

split for DevEval and a 90/5/5 split for RepoST-Train, with the resulting test split combined with RepoST-Eval to form the final evaluation set. We use `GPT-4o` (OpenAI, 2024) and `o3-mini` (OpenAI, 2025) as the primary generator models for constructing training data. Because scientific-domain data is comparatively smaller and more heterogeneous, benchmarks in the scientific domain are used primarily for evaluation and domain-shift analysis, rather than for training.

For evaluation, we construct a comprehensive test suite consisting of (i) all scientific coding benchmarks (LMR-Bench, SciReplicate-Bench, ResearchCodeBench, and AutoExp), and (ii) the test splits of DevEval and RepoST. To better reflect realistic deployment scenarios and evaluate verifiers against diverse model outputs, every evaluation instance is attempted by five heterogeneous code generation models: `GPT-4o` (OpenAI, 2024), `GPT-5.3-Codex` (OpenAI, 2026), `Gemini-3-Flash` (DeepMind, 2025), `Claude-Sonnet-4.6` (Anthropic, 2026), and `Qwen3-Coder-Next` (Cao et al., 2026). All five are represented in every evaluation split.

Selecting a diverse and challenging candidate set To improve the quality of the collected data, we further filter the raw generations to prune redundant and overly easy candidates, in three steps. First, we collapse exact-duplicate candidate bodies and apply diversity-based selection within each instance, computing cosine similarity over code embeddings and keeping the candidates that are most dissimilar from one another. Second, we label each surviving candidate by unit test execution, using three replicates and requiring unanimity, and we drop a candidate if the reference solution for its instance fails the tests. Third, we cross the execution label with a whitespace-normalised `difflib` ratio at threshold 0.5, giving *easy positives* (similar and equivalent), *hard positives* (dissimilar yet equivalent), *hard negatives* (similar yet not equivalent) and *easy negatives*, and retain at most three candidates per instance per cell.

As the result of our data processing pipeline, we present the overall statistics of **RepoEquiv** in Table 1.

4 Experiments

4.1 Models

We evaluate a set of both closed-source and open-source language models on RepoEquiv. Closed-source models include `Claude-Sonnet-4.6`, `Gemini-3-Flash`, `GPT-5.3-Codex` and `GPT-4o`. For open-source models, we evaluate `Qwen2.5-Coder-7B-Instruct` (Hui et al., 2024).

To further learn semantic equivalence verifiers, we additionally perform supervised fine-tuning on Qwen2.5-Coder-7B-Instruct. The model is fine-tuned on the combined RepoST-train and DevEval-train splits of RepoEquiv. We apply LoRA fine-tuning (Hu et al., 2022) with rank = 64 and train for one epoch on the training data.

Model	General Coding		Scientific Coding			
	RepoST	DevEval	LMR	SciRep	AutoExp	ResearchCode
Closed-Source Models						
GPT-4o	76.2	62.0	53.5	79.5	59.0	74.5
GPT-5.3-Codex	65.1	54.8	39.7	82.1	57.7	72.9
Gemini-3-Flash	80.5	64.4	63.5	78.6	68.2	79.3
Claude-Sonnet-4.6	80.0	59.5	47.6	83.7	62.7	79.1
Open-Source Models						
Qwen2.5-Coder-7B	67.7	59.8	46.7	71.0	52.6	70.9
Qwen2.5-Coder-7B-SFT	**78.7**	**66.6**	**52.9**	**75.7**	**63.0**	**72.3**

Table 2: LLMs as verifiers for semantic equivalence checking (Macro-F1 %). Finetuning attains large gains on the general coding benchmarks, reaching scores better or comparable to much larger proprietary models, and gains of the finetuned model is also transferred to on all four scientific benchmarks. Underline marks the best closed-source and **bold** the best open-source model per column; Under a two-sided paired bootstrap against the block runner-up (20,000 resamples), the finetuned model's lead over its base is significant on RepoST and DevEval (both $p < 0.001$) and AutoExperiment (p=0.018), and mildly significant on SciReplicate-Bench (0.152), LMR-Bench (0.220) and ResearchCodeBench (0.422); among closed-source models only Gemini-3-Flash on DevEval (0.081) and AutoExperiment (0.099) separate from their runner-up.

For evaluation, we measure the model performance using macro-F1, which balances precision and recall between equivalent and non-equivalent predictions. We test all comparisons with a two-sided paired bootstrap over evaluation examples (Koehn, 2004) using 10,000 resamples, which controls for example difficulty.

4.2 Main Results

As the main results shown in Table 2, fine-tuning on the collected training data from RepoST and DevEval leads to substantial improvements in semantic equivalence predictions on the corresponding in-domain evaluation sets. The gains are large and unambiguous: $+11.0$ macro-F1 on RepoST and $+6.7$ on DevEval, both significant at $p < 0.001$ under a paired bootstrap. The fine-tuned `Qwen2.5-Coder-7B-Instruct` model achieves performance competitive with or superior to that of much larger proprietary models. Moreover, the benefit of fine-tuning also transfers out of domain, giving uniform improvements across all four scientific coding benchmarks. These results provide strong empirical evidence supporting the feasibility of learning LLM-based semantic equivalence verifiers from execution-labeled data.

The out-of-domain gains are uniform in direction but smaller in size. The fine-tuned model improves on every scientific benchmark: $+10.4$ macro-F1 on AutoExperiment, which is significant at p=0.019, followed by $+6.2$ on LMR-Bench, $+4.7$ on SciReplicate-Bench and $+1.4$ on ResearchCodeBench, with mild significance. The decrease in statistical significance in the scientific domain is largely due to the limited size of these evaluation sets, which are between one and two orders of magnitude smaller than the in-domain ones.

Across all models and datasets, overall macro-F1 scores typically fall in the 50–80 range, indicating that accurately determining semantic equivalence in repository-level contexts remains a difficult task for current LLMs. Supervised fine-tuning on RepoEquiv makes clear progress, while leaving substantial room before LLMs can reliably verify semantic equivalence at repository scale.

4.3 Ablation: Effect of Repository Context

Recent work has suggested that for repository-level code generation tasks such as SWE-Bench, providing repository context does not always improve model performance (Gloaguen et al., 2026). We therefore investigate the role of repository context in semantic equivalence verification through a series of ablation experiments. Specifically, we exclude the context from the prompt and ask the model to judge equivalence from the generated and ground-truth snippets alone, reducing the task to the standalone equivalence detection studied in prior benchmarks.

Model	General Coding		Scientific Coding			
	RepoST	DevEval	LMR	SciRep	AutoExp	ResearchCode
Closed-Source Models						
GPT-4o	$+1.5$	-4.9^{***}	-1.2	-3.0	-0.7	-9.1^{***}
GPT-5.3-Codex	$+6.8^{***}$	-2.1^{*}	$+0.8$	-1.0	-1.1	-4.6^{***}
Gemini-3-Flash	-3.0^{*}	-4.7^{***}	-1.8	$+7.8^{**}$	-5.4	-3.6^{*}
Claude-Sonnet-4.6	-2.1	-3.6^{**}	$+9.0^{*}$	-3.4	-3.9	-3.5^{**}
Open-Source Models						
Qwen2.5-Coder-7B	$+2.6$	-2.6^{*}	-1.2	-0.0	$+3.7$	$+0.0$
Qwen2.5-Coder-7B-SFT	-0.8	-4.6^{**}	-2.1	-4.9	-12.9^{**}	-0.3

Table 3: Change in Macro-F1 (%) when repository context is removed, $\Delta = \mathrm{F1}(\pi_v(I', y_g, y_g^*)) - \mathrm{F1}(\pi_v(I', C', y_g, y_g^*))$, so negative values mean context *helps*. Markers give a paired bootstrap against $\Delta{=}0$ ($^{*}p{<}0.05$, $^{**}p{<}0.01$, $^{***}p{<}0.001$); unmarked cells are indistinguishable from zero. Only 14 of 36 deltas are significant, and they are concentrated in DevEval (6/6, all negative) and ResearchCodeBench (4/6, all negative), versus LMR-Bench (1/6), SciReplicate-Bench (1/6) and AutoExperiment (1/6).

Ablation Results The results of the context ablation experiments are shown in Table 3. Overall, the effect of context on repository-level semantic equivalence checking is mixed: removing context decreases the performance of 25/36 model-benchmark combinations, but only 14 of the 36 deltas are statistically significant. The significant effects are, however, sharply concentrated by benchmark.

On general coding benchmarks, every model shows a significant drop on DevEval (6/6 significant, all negative), whereas RepoST is significant for only 3 of 6 and with mixed sign, suggesting that repository context is genuinely more critical for DevEval tasks than for RepoST. Among the scientific benchmarks, only ResearchCodeBench shows a consistent context effect (4/6 significant, all negative); LMR-Bench, SciReplicate-Bench and AutoExperiment each contribute a single significant cell, and two of those three are positive, i.e. context *hurt*.

At the model level, the finetuned `Qwen2.5-Coder-7B-Instruct` is the only verifier that degrades on every benchmark when context is removed, significantly so on DevEval (-4.6, $p < 0.01$) and AutoExperiment (-12.9, $p < 0.01$). Since the SFT data contain context, this is consistent with finetuning amplifying reliance on repository dependencies for equivalence checking. In the opposite direction, the two largest single effects are positive: removing context *improves* Gemini-3-Flash on SciReplicate-Bench by 7.8 points ($p < 0.01$) and Claude-Sonnet-4.6 on LMR-Bench by 9.0 ($p < 0.05$).

These results indicate that the usefulness of repository context is task- and domain-dependent: it supplies essential dependency information in some settings and irrelevant signal in others, and the changes are small relative to absolute accuracy. This suggests that off-the-shelf LLMs do not fully exploit context for equivalence checking, and that teaching them to retrieve and use the relevant part of it remains an important direction.

4.4 Calibration Analysis

Beyond raw accuracy, we study whether semantic equivalence verifiers produce well-calibrated confidence estimations, which are critical for hybrid strategies that combine model-based verification with selective execution, as discussed in section 1. We evaluate calibration with calibration curves and expected calibration error (ECE) for the base and fine-tuned `Qwen2.5-Coder-7B-Instruct` across DevEval, RepoST, and all scientific coding benchmarks. To approximate the binary classification probability of the positive class, we extract the token probabilities $P(\text{Yes})$ and $P(\text{No})$ from the verifier model output, and calculate $P(\text{positive}) = \frac{P(\text{Yes})}{P(\text{Yes})+P(\text{No})}$.

Results Figure 2 shows the resulting reliability diagrams. Fine-tuning consistently improves calibration in both general and scientific coding domains. On RepoST, ECE decreases substantially, from 0.217 to 0.077, and the fitted calibration slope rises from 0.46 to 0.92: the base model is systematically overconfident, whereas the fine-tuned model is close to perfect calibration. A similar trend holds on DevEval, where ECE falls from 0.277 to 0.093 and the slope improves from 0.28

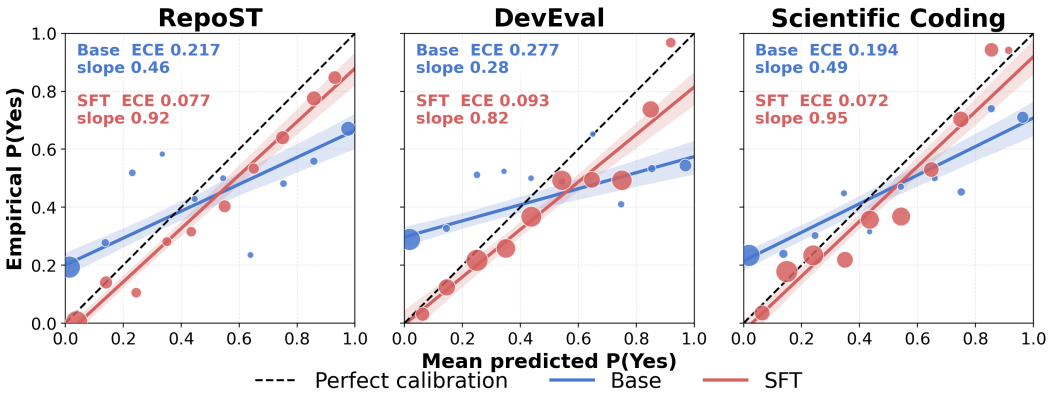


Figure 2: Calibration of base (*Base*) and finetuned (*SFT*) `Qwen2.5-Coder-7B-Instruct`. Points are the ten $P(\text{Yes})$ bins, sized by bin count; lines are count-weighted least-squares fits with 95% bootstrap bands (1,000 resamples), for which perfect calibration is slope 1.

to 0.82. Fine-tuning therefore improves not only accuracy, but also the reliability of confidence estimates. The same pattern holds out of domain, where pooled scientific ECE falls from 0.194 to 0.072,[2] and the slope rises from 0.49 to 0.95: on pooled scientific data the fine-tuned model is close to perfect calibration. Overall, these results indicate that learned semantic verifiers can be calibrated effectively both in-domain and, in aggregate, under domain shift.

From calibration to a measured cost-accuracy trade-off Good calibration is necessary for the hybrid strategy of section 1 but not sufficient: a verifier can be well calibrated in aggregate and still rank its own errors poorly, which is what a deferral policy requires. We therefore measure the trade-off directly. Writing p for the predicted probability of the verifier that a candidate is equivalent, its confidence is $\max(p, 1-p)$. We sort the examples by confidence, defer the least-confident fraction f to execution, which resolves them correctly, answer the rest with the verifier, and sweep f from 0 (verifier only) to 1 (execution only), tracing the curves of Figure 3. Two reference policies bound the result on identical data: *random deferral*, which defers a randomly chosen fraction, and *oracle deferral*, which defers exactly the misclassified cases and is the best any policy could do. We summarize each curve by the area beneath it (AURC), where higher means higher accuracy for the same execution budget.

Confidence-based routing beats random deferral on all six benchmarks and both base and finetuned models, by $+2.1$ to $+6.8$ AURC, and finetuning widens the gap in domain ($+3.82 \rightarrow +4.35$) and out of domain ($+2.77 \rightarrow +3.31$): finetuning improves the *ranking* of the model's own errors, not only the accuracy of its predictions. The saving is practically meaningful. Reaching 95% accuracy on the pooled scientific benchmarks requires executing 58% of cases, against 80% under random routing, and 60% against 84% in domain. Headroom remains, since oracle deferral reaches full accuracy at roughly 30% executed.

4.5 Verification Cost

Execution-free verification is the motivation for RepoEquiv, so we measure what it saves. Table 4 reports the median wall-clock time to verify a single candidate two ways: by running the benchmark's tests, and by querying the finetuned verifier. As of all previous experiments, the verifier is deployed and queried on a single NVIDIA A100-SXM4-80GB GPU, and the execution harnesses are deployed on the combination of CPU and GPU devices required by each benchmark. Overall, the verifier is between $6\times$ and $796\times$ faster on five of the six benchmarks, and the margin grows with how heavy the environment is: it is largest on AutoExperiment and SciReplicate-Bench, whose tasks run full experimental pipelines, and smallest on ResearchCodeBench, whose snippets are short. Among all the benchmarks, RepoST is the one case where execution costs less running time than verifier

[2] Computed over $P(\text{positive})$ as defined above. The alternative confidence-vs-accuracy ECE (confidence $= \max(p, 1-p)$, target $=$ correctness of the arg-max verdict) agrees in direction and is larger, $0.191 \rightarrow 0.012$ here and $0.216 \rightarrow 0.022$ and $0.274 \rightarrow 0.042$ on RepoST and DevEval.

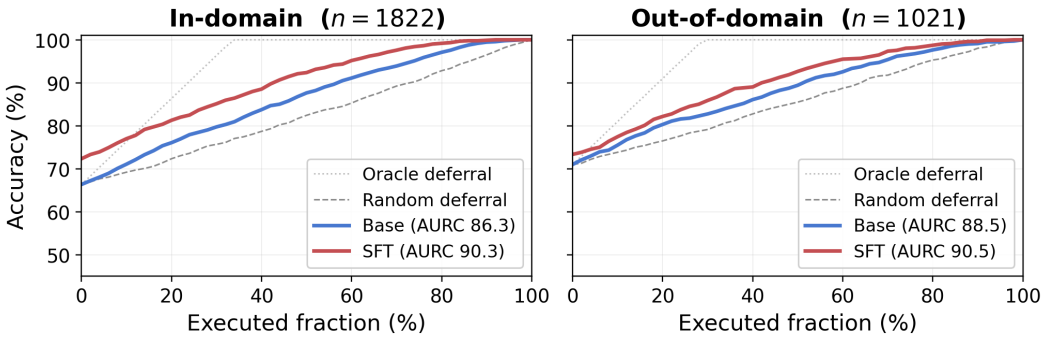


Figure 3: Cost-accuracy trade-off for the hybrid verify-or-execute strategy, pooling DevEval and RepoST (in-domain) and the four scientific benchmarks (out-of-domain). Examples are sorted by verifier confidence $\max(p, 1-p)$; the least-confident fraction is deferred to execution and resolved correctly, the rest answered by `Qwen2.5-Coder-7B-Instruct` before (*Base*) and after finetuning (*SFT*). *Random* deferral is the curve obtained when confidence carries no routing information, *oracle* deferral defers exactly the misclassified cases. Both arms lie above random on both panels and finetuning widens the margin, so finetuning improves the ranking of the model's own errors as well as its accuracy.

Benchmark	**Exec.**	**Verifier**	×	**Benchmark**	**Exec.**	**Verifier**	×
RepoST	0.04	0.07	0.6	LMR-Bench	8.8	0.16	55
DevEval	2.8	0.23	12	SciReplicate-Bench	64.5†	0.26	252
ResearchCodeBench	3.4	0.55	6	AutoExperiment	184.9‡	0.23	796

Table 4: Median wall-clock seconds to verify one candidate by execution and by the finetuned verifier (serial, one candidate at a time), and the resulting speed-up. The model-based verifiers attain 6x to more than 700x run-time speedup compared to execution-based verification, on 5 out of the 6 benchmarks, revealing the substantial cost-saving of the proposed paradigm. It is worth noting that RepoST is the only exception, which is due to that RepoST is designed to be maximally sandboxed, with minimal execution needed.

inference, with the reason that its harness is purpose-built to run in a sandbox and completes in tens of milliseconds.

These measurements cover only the per-candidate cost of verification, not the time and effort required to setup a working environment in the first place. That cost is typically substantial: for example, building the 22 containers of LMR-Bench requires 10.2 CPU-hours and 164 GB of disk, and the scientific benchmarks each needed per-repository dependency resolution before any candidate could be run. On the other hand, obtaining a robust learned verifier carries its own lifecycle costs, including collecting execution-labeled training data, initial training, and potential retraining as models and repositories drift. How to properly compare environment setup against model training and deployment across a full learning and deployment lifecycle is a broad research question that is beyond the scope of this paper.

5 Conclusion

We introduce RepoEquiv, the first benchmark and dataset suite for LLM on repository-level semantic equivalence verification, constructed from six repository-level code generation benchmarks spanning both general software and scientific research domains. Our experiments show that while off-the-shelf LLMs struggle to reliably judge semantic equivalence in complex repository contexts, fine-tuning on RepoEquiv significantly improves verifier performance on both in-domain general coding and out-of-domain scientific coding repositories, demonstrating the feasibility of learning execution-free correctness signals from execution-labeled data. The overall performance indicates that there is substantial headroom for learning a robust verifer with equivalence reasoning with complex context dependency. We hope RepoEquiv supports future work on scalable learned verifiers for repository-level code generation tasks that reduce reliance on costly environment setup and execution.

AI USE STATEMENT

In this work, we used generative AI tools to assist with implementing and debugging the experiments, to provide feedback on research methodology and experimental design, to help interpret results, to help edit the manuscript, and to render an initial version of Figure 1. We have not used generative AI tools for the remaining tasks that require disclosure. We have reviewed all AI-assisted work. In particular, the authors manually checked all code edits made by the AI assistant and verified their correctness with respect to all results reported in the paper, and independently assessed every methodological suggestion and interpretation of results before adopting it. The authors also fully reviewed the paper content drafted or edited by the AI assistant. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Anthropic. Introducing Claude Sonnet 4.6. https://www.anthropic.com/news/claude-sonnet-4-6, 2026. Accessed: 2026-04-01.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models. *ArXiv*, abs/2108.07732, 2021. URL https://arxiv.org/abs/2108.07732.

Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. SWE-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. In *Advances in Neural Information Processing Systems 38 (NeurIPS 2025) Track on Datasets and Benchmarks*, 2025. URL https://arxiv.org/abs/2505.20411.

Ruisheng Cao, Mouxiang Chen, Jiawei Chen, Zeyu Cui, Yunlong Feng, Binyuan Hui, Yuheng Jing, Kaixin Li, Mingze Li, Junyang Lin, Zeyao Ma, Kashun Shum, Xuwu Wang, Jinxi Wei, Jiaxi Yang, Jiajun Zhang, Lei Zhang, Zongmeng Zhang, Wenting Zhao, and Fan Zhou. Qwen3-Coder-Next technical report. *ArXiv*, abs/2603.00729, 2026. URL https://arxiv.org/abs/2603.00729.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *ArXiv*, abs/2107.03374, 2021. URL https://arxiv.org/abs/2107.03374.

Google DeepMind. Gemini 3 Flash: Frontier intelligence built for speed. https://blog.google/products/gemini/gemini-3-flash/, 2025. Accessed: 2026-04-01.

FAIR CodeGen team, Jade Copet, Quentin Carbonneaux, Gal Cohen, Jonas Gehring, Jacob Kahn, Jannik Kossen, Felix Kreuk, Emily McMilin, Michel Meyer, Yuxiang Wei, David Zhang, Kunhao Zheng, Jordi Armengol-Estapé, Pedram Bashiri, Maximilian Beck, Pierre Chambon, Abhishek Charnalia, Chris Cummins, Juliette Decugis, Zacharias V. Fisches, François Fleuret, Fabian Gloeckle, Alex Gu, Michael Hassid, Daniel Haziza, Badr Youbi Idrissi, Christian Keller, Rahul Kindi, Hugh Leather, Gallil Maimon, Aram Markosyan, Francisco Massa, Pierre-Emmanuel Mazaré, Vegard Mella, Naila Murray, Keyur Muzumdar, Peter O'Hearn, Matteo Pagliardini, Dmitrii Pedchenko, Tal Remez, Volker Seeker, Marco Selvi, Oren Sultan, Sida Wang, Luca Wehrstedt, Ori Yoran, Lingming Zhang, Taco Cohen, Yossi Adi, and Gabriel Synnaeve. CWM: An open-weights LLM for research on code generation with world models. *ArXiv*, abs/2510.02387, 2025. URL https://arxiv.org/abs/2510.02387.

Thibaud Gloaguen, Niels Mündler, Mark Niklas Müller, Veselin Raychev, and Martin T. Vechev. Evaluating AGENTS.md: Are repository-level context files helpful for coding agents? In *ICLR 2026 Workshop on Memory for LLM-Based Agentic Systems*, 2026. URL https://arxiv.org/abs/2602.11988.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Tianyu Hua, Harper Hua, Violet Xiang, Benjamin Klieger, Sang T. Truong, Weixin Liang, Fan-Yun Sun, and Nick Haber. ResearchCodeBench: Benchmarking LLMs on implementing novel machine learning research code. In *Advances in Neural Information Processing Systems 38 (NeurIPS 2025) Track on Datasets and Benchmarks*, 2025. URL https://openreview.net/forum?id=3k70Vt0YFS.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-Coder technical report. *ArXiv*, abs/2409.12186, 2024. URL https://arxiv.org/abs/2409.12186.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2E-Gym: Procedural environment generation and hybrid verifiers for scaling open-weights SWE agents. In *Second Conference on Language Modeling (COLM)*, 2025. URL https://openreview.net/forum?id=7evvwwdo3z.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Gyeongwon James Kim, Alex Wilf, Louis-Philippe Morency, and Daniel Fried. From reproduction to replication: Evaluating research agents with progressive code masking. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=qBcHWGBnIb.

Philipp Koehn. Statistical significance tests for machine translation evaluation. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 388–395, 2004. URL https://aclanthology.org/W04-3250/.

Jia Li, Ge Li, Yunfei Zhao, Yongmin Li, Huanyu Liu, Hao Zhu, Lecheng Wang, Kaibo Liu, Zheng Fang, Lanshen Wang, Jiazheng Ding, Xuanming Zhang, Yuqi Zhu, Yihong Dong, Zhi Jin, Binhua Li, Fei Huang, Yongbin Li, Bin Gu, and Mengfei Yang. DevEval: A manually-annotated code generation benchmark aligned with real-world code repositories. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3603–3614, 2024. doi: 10.18653/v1/2024.findings-acl.214. URL https://aclanthology.org/2024.findings-acl.214/.

Junwei Liu, Kaixin Wang, Yixuan Chen, Xin Peng, Zhenpeng Chen, Lingming Zhang, and Yiling Lou. Large language model-based agents for software engineering: A survey. *ACM Transactions on Software Engineering and Methodology*, 2026. doi: 10.1145/3796507. URL https://doi.org/10.1145/3796507.

Nickil Maveli, Antonio Vergari, and Shay B. Cohen. What can large language models capture about code functional equivalence? In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 6880–6918, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.382. URL https://aclanthology.org/2025.findings-naacl.382/.

OpenAI. Hello GPT-4o. https://openai.com/index/hello-gpt-4o/, 2024. Accessed: 2026-04-01.

OpenAI. OpenAI o3-mini. https://openai.com/index/openai-o3-mini/, 2025. Accessed: 2026-04-01.

OpenAI. Introducing GPT-5.3-Codex. https://openai.com/index/introducing-gpt-5-3-codex/, 2026. Accessed: 2026-04-01.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with SWE-Gym. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*, pp. 47717–47737. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/pan25g.html.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. PaperBench: Evaluating AI's ability to replicate AI research. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*, pp. 56843–56873. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/starace25a.html.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=OJd3ayDDoF.

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. EquiBench: Benchmarking large language models' reasoning about program semantics via equivalence checking. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 33868–33881, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1718. URL https://aclanthology.org/2025.emnlp-main.1718/.

Yanzheng Xiang, Hanqi Yan, Shuyin Ouyang, Lin Gui, and Yulan He. SciReplicate-Bench: Benchmarking LLMs in agent-driven algorithmic reproduction from research papers. In *Second Conference on Language Modeling (COLM)*, 2025. URL https://openreview.net/forum?id=8LoPjpvWde.

Yiqing Xie, Alex Xie, Divyanshu Sheth, Pengfei Liu, Daniel Fried, and Carolyn Rosé. RepoST: Scalable repository-level coding environment construction with sandbox testing. In *Second Conference on Language Modeling (COLM)*, 2025. URL https://arxiv.org/abs/2503.07358.

Shuo Yan, Ruochen Li, Ziming Luo, Zimu Wang, Daoyang Li, Liqiang Jing, Kaiyu He, Peilin Wu, Juntong Ni, George Michalopoulos, Yue Zhang, Ziyang Zhang, Mian Zhang, Zhiyu Chen, and Xinya Du. LMR-BENCH: Evaluating LLM agent's ability on reproducing language modeling research. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 6164–6186, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.314. URL https://aclanthology.org/2025.emnlp-main.314/.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652. Curran Associates, Inc., 2024. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/5a7c947568c1b1328ccc5230172e1e7c-Paper-Conference.pdf.

A LIMITATIONS

Our equivalence labels follow the standard practice of the source benchmarks and are derived from unit test outcomes, so they certify agreement on the provided tests rather than equivalence over all valid inputs; we do not additionally quantify this residual label noise. All data is repository-level function-body infilling in Python, so equivalence checking over larger edits such as cross-file refactors, and over other languages, is left to future work. Finally, the scientific evaluation sets are an order of magnitude smaller than the general coding ones, which is why the out-of-domain gains in Table 2, though uniform in direction, are measured with correspondingly wider error.