EVALUATION IS AN INTERVENTION: PROTOCOL CHOICE AS A CONSEQUENTIAL EXPERIMENTAL FACTOR IN LLM PROGRAM REPAIR

Anonymous authors
Paper under double-blind review

ABSTRACT

Benchmark scores are often used to measure a large language model's capability. We argue that the resulting number can be affected by the evaluation procedure as it determines the opportunities and information available to the model. In this paper, we investigate how evaluation protocol choice changes measured program-repair performance. The effects of repeated sampling without feedback, refinement with generic feedback and refinement with structured feedback are compared under matched maximum attempt budgets. Experiments on the synthetic BenchCraft benchmark show substantial protocol effects across three models. The estimated improvement in visible-test success with structured feedback over repeated sampling without feedback is about 11 percentage points for Claude Haiku, 23 for GPT-5.6-terra and 31 for Claude Sonnet 5. Each protocol allowed up to three attempts per task. The results support a substantial improvement, although uncertainty about its size exceeds the pre-specified precision target. In the real-world BugsInPy benchmark experiment, refinement with structured feedback shows no clear advantage over repeated sampling without feedback at a comparable attempt budget. In this experiment, additional attempts produced successful repairs even without feedback. Success rates also varied substantially across defects. The structured-feedback advantage observed on BenchCraft therefore does not clearly extend to BugsInPy under the execution conditions studied. This leads us to conclude that protocol effects depend on the tasks and execution conditions under which they are measured. Evaluation protocol should therefore be treated as a consequential experimental factor. An LLM benchmark score is under-specified without the protocol and execution regime that produced it.

1 INTRODUCTION

A reported benchmark number appears to answer "how well does this model do on this task?" But in reality it answers "how well does this model do on this task under this evaluation procedure." A benchmark score is not solely a property of a model: it is the outcome of a model evaluated under a particular protocol and execution regime. The procedure determines how many generations are drawn, whether the model sees test feedback between attempts, what counts as success, and how a candidate patch is applied. These choices are rarely varied within a study and rarely reported alongside the score. This makes it difficult to compare a particular model's results reported in different publications.

We treat the evaluation protocol as an experimental factor and vary it while keeping the model, repair instances, and maximum number of generation calls fixed. We ask how much a reported program-repair score changes with the protocol. We examine this under controlled conditions and on real-world defects. Protocol choice substantially changes scores in the controlled setting, but the same advantage from structured feedback does not reproduce on real-world defects.

Contributions.

- **Controlled measurement of protocol sensitivity.** A pre-registered, experimental contrast of three protocols (P1, P2, and P3) at matched generation budget over frozen

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.

model and instance sets, analyzed with a two-part hierarchical Bayesian model against thresholds fixed before execution (Section 3–4).

- **Large but strongly model-dependent protocol effects.** On the synthetic benchmark, structured feedback (P3) improved visible-test success over repeated sampling without feedback (P1) by 31.22 percentage points for the model and settings chosen in advance for the primary comparison. The largest estimated improvement across models was roughly three times the smallest. The improvement met the pre-registered requirements for practical importance and confidence in its direction. However, its uncertainty interval was wider than required, so H1 was not fully supported (Section 6).
- **A real-world benchmark that limits generalization.** We compared P1, P2, and P3 on 20 real-world defects from BugsInPy. Structured feedback showed no clear advantage over repeated sampling without feedback, and outcomes varied mainly by defect. The advantage observed on the synthetic benchmark therefore did not clearly carry over to these real-world defects (Section 7).

2 Related work

Sensitivity of LLM evaluation.
An LLM's measured performance depends on how it is evaluated. Previous studies have examined several aspects of this sensitivity. Lu et al. (2022) investigate the ordering of examples in prompts, while Perez, Kiela, and Cho (2021) examine the selection of few-shot examples. Beyond prompt design, Holtzman et al. (2020) study how decoding settings affect generated text. Shao et al. (2025) further show that sensitivity to inference settings can exceed differences between models. Building on this work, we examine how the information available between repair attempts affects measured performance. We also distinguish success on visible tests from success on held-out tests.

Test-time scaling and self-refinement.
Models can use additional computation at inference time to generate multiple candidates or revise earlier responses. Repeated sampling (Brown et al. 2024) and self-consistency (Wang et al. 2023) explore multiple candidate solutions. Self-refinement instead uses feedback to revise a response over successive attempts (Madaan et al. 2023). Li et al. (2025) combine parallel generation and sequential refinement for code. These approaches motivate our comparison of evaluation protocols. We examine how protocol choice changes reported repair performance when the model, instances, and maximum number of generation calls are held fixed.

Program repair and overfitting.
A repair can pass the tests used to guide its development yet fail on other inputs. This problem, known as test-suite overfitting, is well established in automated program repair (APR) (Just, Jalali, and Ernst 2014; Smith et al. 2015). It is particularly relevant when test results guide successive repair attempts. Conversational APR feeds validation results back to the model (Xia and Zhang 2023), while Ahmed et al. (2026) examine test-guided refinement and overfitting on SWE-bench. We build on this work by using APR to study protocol sensitivity. Specifically, we measure whether changing the protocol changes the gap between visible-test success and held-out-test success.

Research gap.

Together, these studies show that evaluation settings and the use of additional inference computation can affect performance. They also show that test-guided repair can overfit the tests used for feedback. Our study connects these findings by measuring how much protocol choice changes reported repair scores while holding the model, repair instances, and maximum number of generation calls fixed. We quantify uncertainty in these effects and examine whether findings from a synthetic benchmark carry over to real-world defects.

3 Experimental design

Unit of analysis.
A *trial* is one complete repair run for a particular combination of model, repair instance, protocol, attempt budget, and replicate. During the run, the model generates patches, which are applied and

evaluated using visible tests. Where held-out tests are available, they provide an additional check of repair success.

Multiple trials use the same repair instance, so their outcomes may be related. We account for this dependence by including instance-level random effects in the statistical model.

Protocols.
All three protocols use the same stopping and acceptance rules. Each trial allows at most K generations and stops when a candidate first passes the visible tests or the budget is exhausted. The first candidate to pass is accepted and evaluated on held-out tests, where available. If no candidate passes the visible tests, the trial is recorded as unsuccessful.

The protocols differ in the information provided after a failed attempt:

- **P1 (independent sampling):** The model generates another patch from the original prompt. It receives no previous patch, failure notice, or test result. Visible tests determine only whether the trial stops.
- **P2 (generic refinement):** The model receives its previous candidate and the message, "The previous solution was incorrect. Try again."
- **P3 (structured test-guided refinement):** The model receives its previous candidate and details of the visible failure, such as failing test names, assertion mismatches, or compiler and runtime errors.

At K=1, the protocols coincide because there is no subsequent attempt to receive feedback. We denote this shared condition as **K1_SHARED** and report protocol contrasts at K=3.

These protocols also differ in prompt length and conversational context. Structured feedback adds information that may be truncated when limits are reached. We therefore interpret each contrast as the effect of the protocol as a whole, rather than the isolated effect of feedback content.

Outcomes.
Visible-test success (VTS) means that the trial's accepted patch passes the visible test suite. When a trial achieves VTS, its accepted patch is run once against a held-out test suite that is never shown to the model. *Validated repair success* (VRS) means that the accepted patch passes both suites: VRS = P(VTS) × P(pass held-out | VTS). The *generalization gap*, Gap = VTS − VRS, is the proportion of trials whose accepted patch passes the visible tests but fails the held-out tests, that is, a repair that overfits the visible suite. A protocol contrast in Gap (ΔGap) is therefore the protocol's VTS gain minus its VRS gain. A positive ΔGap means the protocol raises visible-test success by more than it raises validated repair success. BugsInPy has no held-out suite, so VRS and Gap are defined only for BenchCraft.

Models.

We evaluated three proprietary API models using fixed model identifiers: **claude-sonnet-5**, **gpt-5.6-terra**, and **claude-haiku-4-5-20251001**.

Table 1: Main experiments. Experiment A measures protocol effects on synthetic BenchCraft tasks. Experiment B evaluates the protocols on real-world BugsInPy defects. The experiments are analyzed separately.

Experiment	Benchmark	Tasks / defects	Trials	Purpose
A: Synthetic repair tasks	BenchCraft	30	720	Measure protocol effects on synthetic tasks
B: Real-world defects	BugsInPy	20	480	Measure protocol effects on real-world defects

Benchmarks and experiments.
Table 1 summarizes the experiments that were executed. We used two benchmarks:

- **BugsInPy** (Widyasari et al. 2020) contains real-world Python bugs with developer-written visible tests. It has no hidden test suite in our evaluation, so we cannot measure VRS or the generalisation gap, or assess H3.
- **BenchCraft** (Ebrahim 2026) generates synthetic Python repair tasks. Each task has visible tests and a separate hidden test suite derived from the implementation before the bug was introduced.

We analyze each experiment separately and do not pool results across experiments.

4 STATISTICAL ANALYSIS

We use a two-part hierarchical Bayesian model to estimate the probability that a patch passes the visible tests and, conditional on that success, passes held-out validation. The model accounts for differences between instances and allows the two outcomes to be related. Overall validated repair success is the probability of passing both test suites. We estimate protocol differences at a matched budget of three attempts and report posterior means and 95% credible intervals. Primary inference uses NUTS, a departure from the planned MAP+Laplace method.

Pre-registered thresholds and hypotheses.
The same effect-size and interval-width thresholds apply to both experiments; directional support applies to Experiment A's Bayesian analysis: a minimum effect of 0.05 for VTS, VRS and Gap, a credible-interval width of at most 0.10, and directional support of at least 0.95.

The written plan defines how each hypothesis is assessed:

- **H1:** Protocol contrasts in VTS and VRS must meet all three criteria: effect size, precision and directional support.
- **H2:** A model pair's ranking requires posterior support of at least 0.95.
- **H3:** We assess whether the P3−P1 difference in Gap meets the 0.05 effect threshold with directional support of at least 0.95. The plan requires reporting this evidence without a pass/fail verdict.

The implementation also calculates whether the study meets its design-adequacy and stopping criteria.

5 REGISTERED SCOPE AND WHAT IS POST-REGISTRATION

For Experiment A, we assess the registered hypotheses using the reference condition and thresholds fixed before execution. We corrected the VRS calculation and a missing parameter-transformation term in the original implementation. We also replaced the planned MAP+Laplace inference method with NUTS. These changes use the existing observations and affect estimated effects and uncertainty, but preserve the positive direction of the protocol effect. Convergence diagnostics were satisfactory for the monitored parameters.

Experiment A's protocol contrasts and model-ranking analysis address the registered questions using the corrected analysis. Additional analysis includes per-model and across-model summaries, difficulty stratification, structural comparisons and failure decomposition. Their results are distinguished from the registered hypothesis assessments.

Experiment B examines protocol effects on real-world BugsInPy defects. It was added after the original registration and is analyzed separately from Experiment A.

For Experiment B, we estimate uncertainty by resampling defects using a clustered bootstrap. Its 480 trials comprise 20 defects with 24 trials each. Trials involving the same defect are therefore treated as a cluster rather than as independent observations.

6 RESULTS: BENCHCRAFT

Experiment A is the paper's controlled evidence. Structured feedback beats independent sampling on both visible-test and validated-repair success, for every model and averaged across models, with no posterior uncertainty about the direction (Table 2). The adjacent steps do not all resolve. P3−P2 resolves everywhere (across-model +0.1554 VTS, P=1.000; every model P≥0.998). P2−P1 resolves across-model (+0.0611) and at the reference cell (+0.1359) but is unresolved for gpt-5.6-terra (+0.0267, P=0.838) and claude-haiku-4-5 (+0.0207, P=0.732). Being told specifically what failed helps everywhere; merely receiving the previous attempt does not resolve for two of three models individually.

Table 2: Protocol effects in Experiment A. Estimated differences between structured feedback (P3) and independent sampling (P1), with up to three attempts per task. Results show posterior means, 95% credible intervals, interval widths and the posterior probability of a positive difference. The pre-specified precision target is an interval width of at most 0.10.

Contrast	Scope	Mean	95% CrI	Width	$P(\text{dir})$
ΔVTS	**sonnet-5** *(reg.)*	+0.3122	[0.220, 0.411]	0.192	1.000
	gpt-5.6-terra	+0.2278	[0.152, 0.318]	0.167	1.000
	haiku-4-5	+0.1093	[0.048, 0.177]	0.130	1.000
	across-model	+0.2164	[0.157, 0.281]	0.124	1.000
ΔVRS	**sonnet-5** *(reg.)*	+0.2961	[0.203, 0.394]	0.190	1.000
	gpt-5.6-terra	+0.2282	[0.152, 0.316]	0.165	1.000
	haiku-4-5	+0.1099	[0.046, 0.177]	0.130	1.000
	across-model	+0.2114	[0.152, 0.273]	0.121	1.000
ΔGap	**sonnet-5** *(reg.)*	+0.0161	[−0.021, 0.065]	0.086	0.795
	across-model	+0.0051	[−0.014, 0.028]	0.042	0.708

The effect is strongly model-dependent.
The effect for `sonnet-5` is almost three times that for `haiku-4-5` (Table 2), and the Model×Protocol difference-in-differences is −0.2030 [−0.317, −0.097], comparable to the across-model effect itself. Reporting the reference-cell value as a study-wide result would overstate the across-model effect by roughly 40%. The equal weighting is a post-hoc choice, averaged *within* each posterior draw so that uncertainty and the interaction propagate; it targets a population of exactly these three models.

H1 (written criterion).
Two of three conjuncts are met at the reference cell (Table 2, sonnet-5 rows): ΔVTS and ΔVRS both exceed the 0.05 practical-effect criterion roughly sixfold with full directional support, but both intervals are nearly twice as wide as the ≤0.10 precision criterion allows. H1 is therefore not fully supported: we claim the direction and approximate magnitude of the protocol effect, not its precise size. No exploratory ΔVTS or ΔVRS row meets precision either; only the ΔGap rows do.

H2, ranking robustness.
At $K = 3$ under P3, ranking support is 1.000 for `sonnet-5` vs `haiku-4-5` (+0.3234) and 1.000 for `gpt-5.6-terra` vs `haiku-4-5` (+0.3062); `sonnet-5` vs `gpt-5.6-terra` does not resolve (+0.0173, support 0.881). H2 is met for two of three pairs. This shows which pairs resolve under P3, not that ordering is stable across protocols.

H3, generalisation gap (reported, not adjudicated).
At the reference cell (Table 2), the ΔGap estimate is below the 0.05 minimum-effect threshold and its directional support below 0.95, while its interval width is inside the 0.10 target—the only registered quantity that is. We stop there: failing a criterion is not evidence of absence, and meeting a precision target is not an equivalence test, as the plan fixes no equivalence bounds. The two non-reference models show gaps near zero ($P(\text{dir}) < 0.55$; exploratory).

7 RESULTS: BUGSINPY

Setup and outcome.
Experiment B executed a frozen 480-trial matrix over 20 BugsInPy defects with reasoning explicitly disabled: zero truncation (0/1,104 output-cap events), zero empty responses, and no provider or evaluator errors. Repair was substantial: 83 successes, with 12 of 20 defects repaired at least once.

Applying Experiment A's thresholds to Experiment B. The same 0.05 minimum-effect and 0.10 maximum-width criteria used for H1 apply equally here: in Table 3, the P3−P1 trial-success contrast meets neither criterion, and the P3−P2 contrast meets the effect-size criterion but not precision. As in Experiment A, precision — not the absence of a same-direction effect — is the binding constraint: point estimates favour structured feedback over independent resampling in both experiments, and in both, the interval is wider than the pre-registered target allows.

Structured feedback does not outperform resampling.
The comparison this experiment turns on is P3 (structured feedback) against P1 (independent resampling) at a comparable attempt budget. P3's advantage over P1 is small and its interval covers zero (Table 3). The same holds conditional on attempt 1 failing — the only point at which feedback can act — where the sign of the difference flips if a single defect is dropped. This experiment provides no evidence that structured feedback improves repair success beyond independent resampling at a comparable attempt budget. That is not an equivalence claim: no equivalence bounds were fixed and the defect-clustered intervals remain wide. P2 is directionally worse than P1 on both measures; one possible explanation is anchoring on the failed candidate, but this experiment does not test that mechanism. Only P3−P2 has an interval that excludes zero, and it does not survive Bonferroni correction ([−0.8, +20.8] pp).

Table 3: Protocol effects in Experiment B. Results on 20 BugsInPy defects across three models and 480 trials. Uncertainty intervals are estimated by resampling defects, keeping trials from the same defect together. Protocol differences use the same resampled defects to preserve pairing.

Protocol / comparison	Trial success rate	95% CI	Rescue after attempt 1	95% CI
P1	24/120 = 20.0%	[8.3%, 33.3%]	14/110 = 12.7%	[4.3%, 23.6%]
P2	18/120 = 15.0%	[6.7%, 25.0%]	4/106 = 3.8%	[0.9%, 7.5%]
P3	29/120 = 24.2%	[12.5%, 36.7%]	16/107 = 15.0%	[6.8%, 25.5%]
P2 − P1	−5.0 pp	[−12.5, +0.8] pp	−9.0 pp	[−19.5, −0.9] pp
P3 − P1	+4.2 pp	[−3.3, +12.5] pp	+2.4 pp	[−4.7, +10.2] pp
P3 − P2	+9.2 pp	[+0.8, +18.3] pp	+11.3 pp	[+3.6, +20.5] pp

Note: Rescue rate measures success on later attempts among trials whose first attempt failed. The shared single-attempt baseline provides a reference success rate and does not compare feedback strategies. This table reports only the 360 trials in the three K=3 protocol arms (71 successes); the remaining 120 trials and 12 successes in the 480-trial matrix belong to the K1_SHARED baseline (Section 3), bringing the matrix-wide total to 83 successes.

Additional attempts provide substantial repair opportunity.
First-attempt prompts are identical across P1, P2 and P3 by frozen gate, and attempt-1 success is indistinguishable (10, 14 and 13 of 120), so the arms start from the same place and the protocol signal lies entirely after attempt 1. There, 47.9% of successful multi-attempt trials required iteration beyond the first attempt, and P1 rescues initial failures with no feedback of any kind (Table 3). Additional attempts account for a substantial share of the repair gains, while no additional P3-over-P1 feedback benefit is supported here.

Why the signal is weak.
Of 693 terminal episodes, 75.9% end at patch application, 12.1% at visible tests and 12.0% in success. Many trials therefore fail before an executable patch exists, so structured test feedback cannot operate in those cases — a plausible structural explanation for the weak signal, not a demonstrated causal mechanism. Defect identity dominates the rest: per-defect success ranges from 0% to 62.5%, 8 of 20 defects are never repaired, and the top five supply 74.7% of all successes.

That concentration is why every interval here is a defect-clustered bootstrap: the 480 trials are 20 clusters of 24, not independent draws.

Scope, not contradiction.
The Experiment A effect does not reproduce clearly on real BugsInPy defects under the Experiment B execution regime. The results therefore restrict the scope of the controlled effect rather than invalidate it. The two experiments differ in benchmark realism, defect distribution, inference regime (provider-default versus explicitly disabled reasoning) and failure profile, and any of these may carry the difference. That is this paper's own thesis applied to its headline number: protocol effects are themselves task- and execution-regime dependent.

8 DIFFICULTY STRATIFICATION AND LABEL PROVENANCE

What the labels are.
BenchCraft includes a structural difficulty checker. It measures four properties of the faulty code: cognitive complexity, cyclomatic complexity, nesting depth at the fault line, and branch count. It uses tier-specific thresholds to assign a difficulty level. We reimplemented the checker from the generator's source and verified that it reproduces the original implementation's behavior and the measurements recorded during generation.

However, the difficulty gate was disabled when this corpus was generated. None of the exported records shows that a measured tier replaced the generating model's declared label. The easy, medium and hard labels therefore reflect the model's assessment, not an enforced structural classification.

These labels were assigned before the repair trials, so they could not have been influenced by the outcomes. We can use them to compare protocol effects across declared difficulty groups. Those comparisons do not establish that the groups represent distinct levels of structural difficulty or that difficulty causes differences in protocol sensitivity.

Do declared labels track structure?
Partly. Medians order easy ≤ medium ≤ hard on all four metrics. The rank-based probability that a random easy instance scores below a random hard one (0.5 = no separation) is 0.91 for cognitive complexity, 0.91 for branch count and 0.85 for cyclomatic complexity. But nesting depth barely separates (0.56 easy-vs-hard; 0.43 medium-vs-hard, slightly reversed), and medium-vs-hard separates weakly on every metric (0.43–0.74)—the same pair the protocol analysis leaves unresolved.

Table 4: Protocol effects in Experiment A by generator-declared difficulty. Each difficulty group is analyzed separately using NUTS. These exploratory estimates describe effects within each group and are not directly interchangeable with the overall estimates in Table 2.

Stratum	n	P1@ $K = 3$	P3@ $K = 3$	ΔVTS[P3-P1] [95% CrI]
easy	7	0.879	0.987	+0.1080 [0.024, 0.256]
medium	12	0.663	0.966	+0.3020 [0.167, 0.448]
hard	11	0.495	0.949	+0.4546 [0.306, 0.600]

What the tier contrasts establish.
Little. The strata are refit under NUTS (Table 4; $\hat{R} \leq 1.0042$, min ESS 728, zero divergences), replacing unstable 7-node MAP+Laplace fits. The ordering easy < medium < hard holds in point estimates, and hard−easy separates on both outcomes: ΔVTS +0.3465 [0.148, 0.525], P(>0)=1.000; ΔVRS +0.3432 [0.144, 0.519], P=0.999. Intermediate steps are mixed (medium−easy: P(>0)=0.974 VTS, P(>0)=0.911 VRS; hard−medium: P(>0)=0.925 VTS, P(>0)=0.975 VRS), and every interval is three to four times the precision target, so we report a between-extreme ordering and no resolved claim about adjacent strata. A pre-committed frequentist trend test on tier rank finds no significant interaction (+0.1143, 95% CI [−0.2419, +0.4704], p=0.53). The easy stratum's high P1 level leaves little headroom, nearly all of which P3 recovers (Table 4) — compatible with a ceiling constraint, though a genuinely smaller effect on simpler instances would give the same pattern. We claim nothing about true task difficulty.

9 LIMITATIONS

Precision is not met for H1, so we claim direction and approximate magnitude only. Experiment B applies the same matched-protocol design and the same effect-size and precision thresholds as Experiment A to real-world defects; as in Experiment A, precision is the binding constraint on both experiments' headline contrasts (Section 7). BugsInPy has no hidden oracle, so VRS, Gap and H3 are not evaluable there. Difficulty labels are generator declarations, not certified properties. Compute is matched on the maximum number of attempts a trial is allowed, not realized tokens; no instance × protocol random slopes or posterior predictive checks were run; scope is three proprietary models and two Python benchmarks.

10 CONCLUSION

Evaluation protocol can move measured program-repair performance substantially: on our synthetic benchmark, replacing independent sampling with structured verifier feedback changes visible-test success by +0.31 at the registered reference condition (+0.22 averaged over three models), with model and budget fixed. On real BugsInPy defects that advantage does not reproduce: additional attempts provide substantial repair opportunity, while structured feedback does not show a clear advantage over independent resampling, and defect identity dominates what remains. The magnitude and practical benefit of a protocol effect are therefore conditional on the task and the execution regime. A benchmark score reported without its evaluation protocol, and the regime that produced it, is under-specified.

AI USE STATEMENT

Large language models appear in two distinct roles. **As objects of study**, the three evaluated models (Section 3) are the subject of the experiments. **As authoring and engineering assistance**, an LLM-based coding assistant was used to implement analysis code, draft and revise manuscript text, and run verification passes over numerical claims against the frozen artifacts. All reported numbers derive from committed artifacts rather than model output, every analysis script is released, and the authors take full responsibility for the content, including all claims and limitations.

REFERENCES

Ahmed, Toufique, Jatin Ganhotra, Avraham Shinnar, and Martin Hirzel. 2026. "Investigating Test Overfitting on SWE-Bench." In *FSE (Ideas, Visions and Reflections)*.

Brown, Bradley et al. 2024. "Large Language Monkeys: Scaling Inference Compute with Repeated Sampling." *arXiv:2407.21787*.

Ebrahim, Yasser. 2026. "Specification-Driven Benchmarking for Automated Program Repair." *arXiv:2609.28896*.

Holtzman, Ari, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. "The Curious Case of Neural Text Degeneration." In *ICLR*.

Just, René, Darioush Jalali, and Michael D. Ernst. 2014. "Defects4J: A Database of Existing Faults to Enable Controlled Testing Studies for Java Programs." In *ISSTA*.

Li, Dacheng et al. 2025. "S*: Test Time Scaling for Code Generation." In *Findings of EMNLP*.

Lu, Yao, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2022. "Fantastically Ordered Prompts and Where to Find Them." In *ACL*.

Madaan, Aman et al. 2023. "Self-Refine: Iterative Refinement with Self-Feedback." *arXiv:2303.17651*.

Perez, Ethan, Douwe Kiela, and Kyunghyun Cho. 2021. "True Few-Shot Learning with Language Models." In *NeurIPS*.

Shao, Minghao et al. 2025. "Configuration over Selection: Hyperparameter Sensitivity in LLM Evaluation." *arXiv*.

Smith, Edward K., Earl T. Barr, Claire Le Goues, and Yuriy Brun. 2015. "Is the Cure Worse Than the Disease? Overfitting in Automated Program Repair." In *ESEC/FSE*.

Wang, Xuezhi et al. 2023. “Self-Consistency Improves Chain of Thought Reasoning in Language Models.” In *ICLR*.

Widyasari, Ratnadira et al. 2020. “BugsInPy: A Database of Existing Bugs in Python Programs.” In *ESEC/FSE*.

Xia, Chunqiu Steven, and Lingming Zhang. 2023. “Keep the Conversation Going: Fixing 162 Out of 337 Bugs for $0.42 Each Using ChatGPT.” In *ISSTA*.