ACTION, NOT BELIEF: WHAT MAKES HYPOTHESIS PROMPTING WORK IN DEBUGGER AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent work has introduced hypothesis-driven debugging into LLM-based program repair, prompting agents to formulate hypotheses, predict observations, and test them with debugging tools. However, these methods combine hypothesis generation with guidance for subsequent actions, making it unclear which component drives the repair gain. We disentangle these components through controlled comparisons under a fixed debugger interface. To examine whether prediction–observation agreement tracks repair performance, we introduce HypoCal, a measure of agreement between stated runtime predictions and subsequent debugger observations. Our experiments show that injecting hypotheses into the next action prompt improves repair, whereas eliciting hypotheses alone does not. The gain disappears when the injected hypothesis concerns another bug or misidentifies the fault location. HypoCal remains low across prompting conditions and does not track repair gains. An analysis of tool use shows that hypothesis injection shifts tool use from source inspection toward runtime probing. Across four models, repair effects have the same rank ordering as baseline structured-probing rates, while injection redirects the models toward different tools. Together, these findings characterize hypothesis prompting as action shaping: its repair effect varies with the model's existing tool-use behavior.

1 INTRODUCTION

Debugging requires testing explanations of failures against evidence from the running program (Zeller, 2009; Böhme et al., 2017; Li & Coblenz, 2026). LLM coding agents can gather such evidence by interacting with execution environments as they inspect and modify code (Yang et al., 2024; Wang et al., 2025; 2024b). Despite having access to a debugger (Yuan et al., 2025), the reactive baseline we study relies primarily on source inspection. This gap between available tools and selected actions motivates a closer examination of how agents are prompted to investigate failures (Xu et al., 2026; Faghih et al., 2025; Ghoshal et al., 2026).

Recent work has incorporated hypothesis-driven debugging and interactive runtime inspection into LLM-based program repair (Kang et al., 2025; Garg & Huang, 2026; Wang et al., 2026; Xiang et al., 2026; Liu et al., 2026). In hypothesis-driven debugging, an agent proposes an explanation of a failure, predicts an observation, and tests the prediction against program behavior (Zeller, 2009; Kang et al., 2025). In the hypothesis-prompting protocol we study, the generated hypothesis is also included in the prompt used to select the next action. The full protocol therefore combines two components: eliciting a hypothesis and using it to guide the next action. Repair outcomes alone cannot show which component drives the gain, or whether the gain reflects better prediction–observation agreement or different action choices.

To separate these components, we use a paired falsification ladder under a fixed debugger interface. We first separate hypothesis elicitation from injection into the next action prompt, then vary whether the agent is instructed to test or falsify the hypothesis, whether predictions are structured or free text, and whether the injected hypothesis concerns the current bug and correctly localizes the fault. We introduce HypoCal, a per-step measure of agreement between predicted runtime values and subsequent debugger observations, computed independently of repair outcomes. We evaluate the full

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.

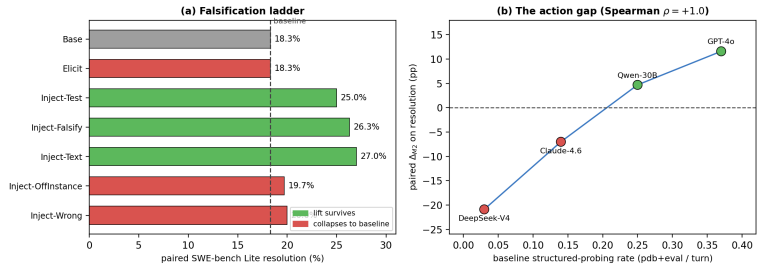

Figure 1: Overview of the analysis. Left: the falsification ladder separates components of hypothesis prompting and tests whether the injected hypothesis concerns the current bug and correctly localizes the fault. Green denotes conditions that retain the repair gain; red denotes controls that do not. Right: across four language models tested on the same 43 issues, the ranking of repair effects matches the ranking of baseline structured-probing rates (`pdb` and `eval` calls per turn; Spearman $\rho = +1.0$), but not the ranking of baseline repair rates.

ladder on SWE-bench Lite (Jimenez et al., 2024) with Qwen and compare the effects of injection across four language models on a shared subset.

In the main Qwen evaluation, eliciting a hypothesis without including it in the next action prompt yields no significant repair gain, whereas injection improves repair. The gain persists whether the prompt asks the agent to test or falsify the hypothesis and when structured predictions are replaced by free text. It disappears when the injected hypothesis concerns another bug or misidentifies the fault location. These comparisons show what makes injection effective: a correctly localized hypothesis about the current bug, supplied when the agent selects its next action.

HypoCal remains low across prompting conditions despite substantial differences in repair rates. In Qwen, injection shifts tool use from reading source code toward inspecting the running program with `pdb` and `eval`, including after turns with zero measured agreement. The clearest change is therefore in action selection rather than measured prediction–observation agreement.

Injection produces different repair effects across models. On the same 43 issues, it improves repair with GPT-4o and Qwen but reduces repair with Claude and DeepSeek. The associated changes in tool use also differ across models. Across the four models, injection effects follow the same ordering as baseline structured-probing rates (`pdb` and `eval` calls per turn), but not baseline repair rates. We term this relationship the *action gap*. Hypothesis prompting reshapes action selection, with its repair effect varying with the model's existing tool-use behavior.

Our contributions are:

1. We develop a paired falsification ladder that separates hypothesis elicitation, action-time injection, test-versus-falsify instructions, prediction format, and hypothesis content under a fixed debugger interface. These comparisons identify a correctly localized hypothesis about the current bug, supplied at action time, as the key contributor to the repair gain.
2. We introduce HypoCal, a per-step measure of agreement between predicted runtime values and subsequent debugger observations, and show that this agreement does not track repair gains.
3. We analyze how hypothesis injection changes tool use across language models and characterize the *action gap*, linking differences in repair effects to each model's baseline structured-probing behavior.

2 Setup: Hypothesis-Injection Variants and HypoCal

We evaluate repository-level bug repair. Given a natural-language issue description and a Linux sandbox containing the project source, the agent issues tool calls under a fixed interaction budget. Repair success is determined by held-out tests. The environment exposes six tools: `pdb`, `eval`, `view`, `listdir`, `edit`, and `bash`. We use a budget of $T = 25$ turns and 1,800 seconds and report

the instance-level resolution rate on all 300 instances in SWE-bench Lite (Jimenez et al., 2024). Unless stated otherwise, experiments use the self-hosted Qwen3-Coder-30B-A3B model (Qwen Team, 2025) served with vLLM; §3.4 compares Qwen with three additional language models on a shared 43-instance subset. Within each decoder seed, we compare paired binary outcomes using the exact McNemar test (McNemar, 1947); the multi-seed analysis uses five decoder seeds.

The baseline, **Base (M0)**, is the reactive `debug-gym` agent. At each turn, it receives the issue description, the interaction history, and the latest observation, then selects one tool action. The remaining modes extend the same agent, with paired comparisons isolating individual components of the hypothesis-injection protocol. Table 1 summarizes these comparisons; the aliases `M0...M2W` are retained throughout the paper.

Table 1: Hypothesis-injection modes and the component tested by each paired comparison. Each intervention varies one component relative to a matched control.

Mode (alias)	Intervention	Component tested
Base (M0)	reactive agent; free-form action each turn	reference condition
Elicit (M1)	structured hypothesis elicited and scored, not injected	hypothesis elicitation / extra LLM call
Inject-Test (M2)	elicited hypothesis injected before action selection	action-time hypothesis injection
Inject-Falsify (M3)	Inject-Test with a falsify instruction	test vs. falsify instruction
Inject-Text (M2T)	Inject-Test with a free-text prediction	structured vs. free-text prediction
Inject-OffInstance (M2TS)	Inject-Text with a hypothesis elicited for another instance	current-instance relevance
Inject-Wrong (M2W)	Inject-Text with a hypothesis that mislocalizes the fault	fault localization

HypoCal: measuring prediction–observation agreement. For each mode with structured predictions (M1–M3), we compare the predicted runtime values with the observation produced by the subsequent tool call. The fraction of predicted values matched in the subsequent observation defines HypoCal V0, a per-step measure of prediction–observation agreement. HypoCal uses neither gold patches nor an LLM judge, is not shown to the agent, and is not optimized during inference. We use it to test whether prediction–observation agreement tracks the repair gain. HypoCal V1 is a stricter offline re-scoring procedure with action gating, numeric tolerance, and flexible type matching. The prompt schema and HypoCal V0 definition are given in Appendix B, and HypoCal V1 is detailed in Appendix D.2.4.

3 Experiments

Our experiments ask whether the repair gain from hypothesis prompting is explained by prediction–observation agreement or changes in action selection. Paired comparisons isolate the effects of additional reasoning, test-versus-falsify instructions, prediction format, and injected content. We then examine how these effects relate to tool-use patterns across models.

3.1 Action-time injection reliably improves repair in Qwen

We compare Base, Elicit, Inject-Test, and Inject-Falsify on all 300 instances using exact paired McNemar tests. We repeat the Base–Inject-Test and Base–Inject-Falsify comparisons across five decoder seeds.

Injection produces a substantial lift. On the primary seed, resolution rises from 18.3% (Base) to 25.0% (Inject-Test) and 26.3% (Inject-Falsify), with exact paired McNemar $p \leq 0.0022$ for both comparisons. Across five decoder seeds, the mean lift over a 20.0% baseline is $+4.2$pp for Inject-Test and $+3.9$pp for Inject-Falsify, with positive gains in every seed (per-seed $+1.7$ to $+6.7$pp). We use

the five-seed mean gains as our main estimates; the primary-seed gain of $+6.7$pp lies at the upper end of the observed range (Figure 2).

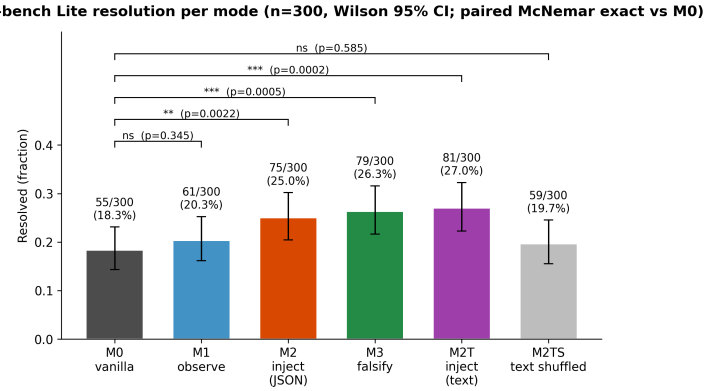

Figure 2: SWE-bench Lite resolution under the reactive baseline (Base, M0) and the injection modes. Action-time injection (Inject-Test/Falsify, M2/M3) increases resolution relative to the elicit-only control (Elicit, M1), which is statistically indistinguishable from Base.

Elicit uses the same schema, scoring procedure, and additional per-turn LLM call as the injection modes, but logs the hypothesis without injecting it. It does not differ significantly from Base on either seed ($p = 0.345/0.185$). The gain therefore depends on supplying the hypothesis at action time, not on the additional reasoning call alone.

Inject-Test uses only $+4.2\%$ more input tokens and fewer output tokens than Base; cost per resolved task decreases by 24% for Inject-Test and 23% for Inject-Falsify. On Claude, where the repair effect is negative, Inject-Test is 11% cheaper per task with similar cost per resolved bug (Appendix D.3). A gold-patch sanity sweep passes $294/300$ instances. Excluding the 6 dubious instances changes every mode rate by ≤ 0.20pp; the gains remain $+6.8$pp for Inject-Test ($p = 0.0022$) and $+8.2$pp for Inject-Falsify ($p = 0.0005$) on $n = 294$ (Appendix D.4).

3.2 The lift is not explained by better beliefs

Establishing the effect of action-time injection leaves open what makes the injected hypothesis useful. In Qwen, we examine test-versus-falsify instructions, prediction–observation agreement, and structured prediction format. We then test whether agreement tracks repair effects across models.

The test-versus-falsify instruction does not explain the gain. "Test the hypothesis" (Inject-Test) and "falsify the hypothesis" (Inject-Falsify) do not differ significantly on either seed ($p = 0.67/0.40$), and the sign of their difference flips between seeds. The gain persists under both instructions, rather than depending on the choice between them.

Repair outcomes and measured prediction–observation agreement diverge across Elicit, Inject-Test, and Inject-Falsify. If the repair gain came from more accurate runtime predictions, higher prediction–observation agreement should accompany better repair. HypoCal V0, however, changes little despite a 6pp resolution gap. Stated confidence remains at ≥ 0.91 across modes, even as measured runtime agreement stays low (Table 2).

Table 2: HypoCal remains low across Elicit, Inject-Test, and Inject-Falsify despite a 6pp resolution gap, while stated confidence remains uniformly high.

Mode (alias)	HypoCal V0 (per-step)	mean stated confidence	resolution
Elicit (M1)	0.023	0.913	20.3%
Inject-Test (M2)	0.034	0.916	25.0%
Inject-Falsify (M3)	0.034	0.915	26.3%

Of the 193 runs with at least one HypoCal score above 0.20, 27% resolve, compared with 24% across hypothesis-aware runs; the difference is not statistically significant (Figure 3). The pattern persists under a stricter matching rule. HypoCal V1, an offline re-scorer with action gating and numeric tolerance, yields low agreement across all three modes (0.008–0.014; Appendix D.2.4).

On `astropy-14995`, Elicit and Inject-Falsify reach identical diagnostic depth and identical HypoCal (≈ 0.33), yet only Inject-Falsify's action — rewriting both `None`-branches rather than one — passes the hidden test (Appendix D.2.3). The matched diagnostic evidence thus leads to different edits and different repair outcomes.

Because HypoCal is measured from structured predictions, we separately test whether the prediction format itself improves repair. Replacing the JSON / `expected_state` schema with a single free-text sentence (Inject-Text) retains a 27.0% resolution rate (+8.7pp vs. Base, $p = 0.0002$), with no detectable difference from structured Inject-Test ($p = 0.49$). The schema makes HypoCal measurement possible, but the repair gain persists without it.

The separation between measured agreement and repair effects extends beyond Qwen. HypoCal remains low across all four models and does not track their paired repair effects. Claude has the highest HypoCal but a negative repair effect, while DeepSeek has the lowest HypoCal and the most negative effect (Table 3).

Table 3: Cross-model HypoCal and repair effects. Across a 7× range in baseline resolution, HypoCal varies from 1.8% to 4.0% and does not track the paired repair effects (full magnitudes in §3.4 and Appendix D.6.2).

Model	HypoCal V0	mean stated confidence	sign of paired Δ
GPT-4o	2.79%	0.84	+ (largest)
Qwen3-Coder-30B	3.46%	0.92	+
Claude-Sonnet-4.6	**4.00%**	0.93	−
DeepSeek-V4	1.81%	0.93	− (largest)

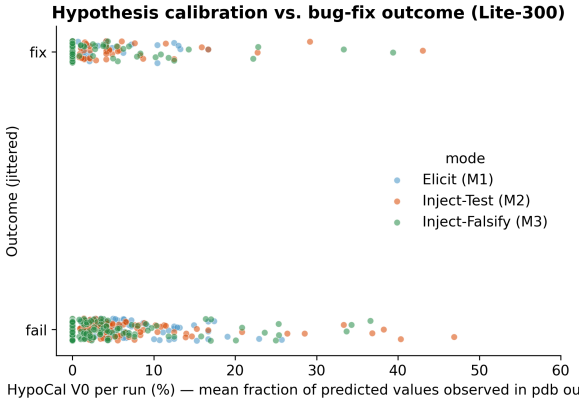

Figure 3: HypoCal and resolution across Elicit, Inject-Test, and Inject-Falsify. Per-step agreement remains near floor and does not track repair success within Qwen or paired repair effects across models.

3.3 Instance-grounded content reshapes actions

A hypothesis may help because it describes the current bug, or simply because plausible text occupies the prompt slot. We distinguish these possibilities before examining tool selection.

Inject-OffInstance and Inject-Wrong hold the template, injection slot, length, and decoder of Inject-Text fixed, varying only the injected content.

Inject-OffInstance tests whether the injected text must describe the current instance. It fills the same slot with a hypothesis elicited on a different instance. Resolution falls to 19.7%, corresponding to

+1.3pp vs. Base ($p = 0.58$) and −7.3pp vs. Inject-Text ($p = 0.0016$). The prompt slot remains occupied, but the repair gain does not; plausible text alone is insufficient if it does not concern the current bug.

Inject-Wrong uses a hypothesis about the same instance, but one that points to the wrong fault location. Across three seeds it shows no detectable difference from the off-instance control (Wilcoxon $p = 0.92$) and stays significantly below the correct self-elicited hypothesis ($p = 0.015$ vs. Inject-Text). The distinction is therefore not simply whether the text concerns the instance. Correct fault localization matters.

Relative to Elicit, Inject-Test reduces the `view` share by 10.5pp and increases the combined `pdb`/`eval` share by 12.2pp (Table 4, Figure 4). Input tokens increase by at most 12%, while the number of turns increases by less than 3%. The shift is therefore in the choice of tools, with comparatively little change in resource use.

Table 4: Per-mode action distribution (share of tool calls). Inject-Test makes 3.4× as many `pdb` calls per turn and 31% more `eval` calls per turn than Elicit, redirecting ≈ 10pp off `view` (≈ 2.0 vs. 0.6 `pdb` sessions per task). †Base values are approximate estimates from a pilot matrix and do not differ significantly from Elicit (χ^2 test, $p > 0.3$).

Mode (alias)	`view`	`eval`	`pdb`	`edit`	`bash`
Base (M0)†	53%	22%	1%	14%	9%
Elicit (M1)	53.0%	20.3%	2.5%	15.3%	8.7%
Inject-Test (M2)	**42.5**%	**26.6**%	**8.4**%	11.8%	10.7%
Inject-Falsify (M3)	46.3%	26.1%	4.3%	10.6%	12.6%

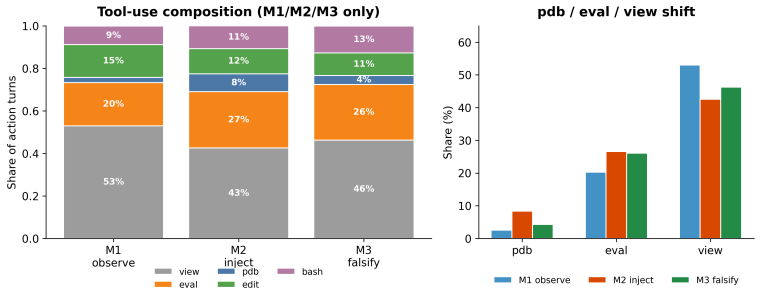

Figure 4: Action redistribution under injection. Relative to the elicit-only control, injected agents shift their tool-call distribution from `view` toward runtime probing with `pdb`/`eval`, while input tokens increase by at most 12% and turns by less than 3%.

Does the shift toward probing require agreement between the agent's prediction and the subsequent observation? The main shift persists even when HypoCal is zero. Approximately 95% of scored turns have zero measured agreement, yet the next-turn tool-use distribution after those turns already shows the same redistribution. A small upper tail (≈ 6% of turns) shows an additional 10–14pp of probing, peaking at +13.8pp when HypoCal= 1 ($p = 0.006$; Appendix D.6.3). The additional probing at high HypoCal is too infrequent to account for the overall repair gain; the main redistribution is already present without measured agreement.

The content controls identify relevance to the current bug and correct fault localization as the critical properties of the injected hypothesis. In Qwen, that content also redirects tool use toward runtime probing, including after zero-HypoCal turns.

3.4 When and where injection helps: the action gap

On the 294 instances that pass the gold-patch sanity check, repair effects vary across repositories. Inject-Test's paired effect is concentrated in `django` (+12.3pp, $p = 0.0026$, $n = 114$), which accounts for approximately 38% of Lite-300. On the $n = 180$ non-django subset, the effect is +3.3pp

($p = 0.263$); Inject-Falsify, by contrast, retains a $+7.8$pp lift outside `django` ($p = 0.0013$). The aggregate comparison between Inject-Test and Inject-Falsify therefore masks different repository-level patterns (Appendix D.1).

The cross-model comparison runs Base and Inject-Test on a shared 43-instance subset using four models spanning a $7\times$ range in baseline resolution (GPT-4o, Qwen3-Coder-30B, DeepSeek-V4, and Claude-Sonnet-4.6).

Across the four models, paired repair effects have the same rank ordering as baseline structured-probing rates (`pdb+eval` calls per turn; Spearman $\rho = +1.0$; Table 5, Figure 5). Baseline resolution does not show the same ordering. DeepSeek starts at 37.2% and loses 20.9pp, whereas Claude starts at 69.8% and loses 7.0pp. The association is with the models' existing probing behavior, not simply with how many tasks they resolve before injection.

Table 5: Baseline tool use and paired repair effects on the shared 43-instance subset. Structured probing counts `pdb` and `eval` calls per turn; repair effects are Inject-Test minus Base.

Model	baseline probing rate	baseline resolution (Base)	paired Δ (Inject-Test)	baseline `bash` share
GPT-4o	0.37	9.3%	$+11.6$pp	—
Qwen3-Coder-30B	0.25	18.6%	$+4.7$pp	—
Claude-Sonnet-4.6	0.14	69.8%	-7.0pp	38%
DeepSeek-V4	0.03	37.2%	-20.9pp	55%

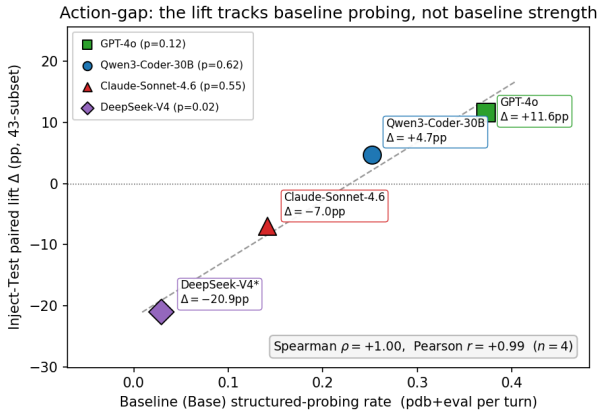

Figure 5: The action gap. Across four models, paired repair effects have the same rank ordering as baseline structured-probing rates (Spearman $\rho = +1.0$), but not baseline resolution.

The matching rank order does not imply that injection sends every model toward the same tools (Table 6). Qwen shifts from `view` toward `pdb`/`eval`; GPT-4o's largest increase is instead in `view` ($+7.8$pp), alongside a -5.2pp change in `bash`. Claude and DeepSeek increase their already-dominant `bash` share by 19.0pp and 13.4pp, respectively, alongside negative repair effects. The same injected prompt thus redirects different models toward different actions.

The consequences are visible in Claude's trajectories. In all 7 cases where Base resolves the issue but Inject-Test does not, the Inject-Test trajectory exhausts the 25-turn budget; $4/7$ continue probing past the edit point. For example, `astropy-14182` spends $23/25$ turns probing and never edits (Appendix D.6.1). In these cases, extended investigation consumes the interaction budget without producing a successful repair.

We term this cross-model relationship the *action gap*. Baseline structured-probing rates track the repair effect, while tool-share changes show that injection redirects different models toward different actions.

Table 6: Tool-share changes and repair outcomes on the same 43-instance subset. For each model, we report the largest decrease and increase among the six tool categories, alongside the paired repair-rate change. All changes are Inject-Test minus Base, in percentage points (Appendix D.6.4).

Model	Largest decrease (pp)	Largest increase (pp)	Repair Δ (pp)
GPT-4o	`bash` −5.2	`view` +7.8	+11.6
Qwen3-Coder-30B	`view` −8.7	`pdb` +5.6	+4.7
Claude-Sonnet-4.6	`other` −17.8	`bash` +19.0	−7.0
DeepSeek-V4	`edit` −7.3	`bash` +13.4	−20.9

4 Related work

Scientific debugging for LLM agents. Scientific debugging tests hypotheses about faults against observations from program execution (Zeller, 2009). AutoSD (Kang et al., 2025) implements this cycle with LLM-generated hypotheses and debugger experiments, then uses the resulting observations to guide subsequent debugging steps. Debug2Fix (Garg & Huang, 2026) delegates runtime questions to a debugging subagent, while InspectCoder (Wang et al., 2026) intervenes in intermediate runtime values using debugger feedback.

Execution-guided self-debugging and generated-test selection likewise use execution feedback to revise or select code candidates (Chen et al., 2024; 2023). Dynamic-analysis systems expand the runtime evidence and debugger interfaces available to coding agents (Liu et al., 2026; Xiang et al., 2026); `debug-gym`, SWE-Agent, and OpenHands provide environments or interfaces for repository navigation, code modification, and execution (Yuan et al., 2025; Yang et al., 2024; Wang et al., 2025). Controlled studies further examine how tool architecture affects coding-agent behavior (Xu et al., 2026). Holding each model's debugger interface fixed, we compare hypothesis elicitation with injection into the subsequent action prompt, and separately vary test-versus-falsify instructions, prediction format, and injected content. Table 7 compares the closest debugging systems by their interventions and measurements.

Table 7: Comparison with closely related debugging systems. − denotes an intervention or measurement not reported in the cited work; "repair outcomes" denotes cross-model repair evaluation without corresponding tool-use analysis. BCB-R abbreviates BigCodeBench-R; further benchmark details appear in Appendix C.1.

Comparison dimension	AutoSD ('25)	InspectCoder ('26)	Debug2Fix ('26)	This work
Prediction format	free text	–	–	JSON / free-text control
Deterministic agreement score	–	–	–	HypoCal V0/V1
Matched elicitation/injection	–	–	–	Base/Elicit/Inject
Repair benchmark	Defects4J	BCB-R	SWE-bench Live	SWE-bench Lite
Cross-model analysis	–	–	repair outcomes	tool use + repair (4)
Agreement/tool-use analysis	–	–	–	HypoCal + tool traces

Confidence, self-evaluation, and calibration. Work on verbalized confidence and model self-evaluation studies how LLMs assess their answers (Lin et al., 2022; Kadavath et al., 2022; Xiong et al., 2024). Calibration relates reported confidence to observed correctness (Guo et al., 2017), while semantic uncertainty and sampled factual consistency draw on agreement or variation across generated responses (Kuhn et al., 2023; Manakul et al., 2023). Studies of capability awareness also examine predictions of task success alongside decisions about whether to act (Barkan et al., 2026). Our predictions instead concern runtime states revealed by subsequent debugger calls. HypoCal measures agreement between each runtime prediction and the subsequent debugger observation; we compare this agreement with tool selection and repair outcomes rather than calibrating a task-success

probability. Related work on self-correction, explanation faithfulness, and intermediate reasoning is discussed in Appendix C.2.

Reasoning and action in tool-using agents. ReAct interleaves reasoning with tool actions (Yao et al., 2023); Reflexion incorporates feedback across trials (Shinn et al., 2023), and Plan-and-Solve organizes reasoning before solution generation (Wang et al., 2023a). Toolformer and CodeAct study tool invocation and executable actions (Schick et al., 2023; Wang et al., 2024b), while CodeSteer examines the choice between code execution and textual reasoning (Chen et al., 2025). SWE-Search explores repair trajectories (Antoniades et al., 2025), and scientific-discovery agents generate hypotheses, test candidates, or design experiments (Schmidgall et al., 2025; Lu et al., 2024; Wang et al., 2024a; Ma et al., 2024). We examine how supplying an explicitly elicited hypothesis to the next action prompt changes tool selection and how those changes relate to repair effects across models.

5 Discussion, Limitations, and Conclusion

On Qwen, eliciting a hypothesis without returning it to the action prompt yields no significant repair gain over Base, whereas Inject-Test and Inject-Falsify improve repair. Free-text injection retains the gain, while off-instance and mis-localized hypotheses do not. The effect thus rests on supplying a hypothesis about the current bug that correctly localizes the fault, rather than on an additional reasoning call or a particular prediction schema. In Qwen, injection also shifts tool use from `view` toward `pdb`/`eval`, including after turns with zero measured HypoCal. Prediction–observation agreement remains low across modes with different repair outcomes.

The cross-model results show why the same intervention can produce different effects. On the shared 43-instance subset, repair effects across four models have the same rank ordering as baseline structured-probing rates, but not baseline resolution. The largest tool-share increase is in `pdb` for Qwen, `view` for GPT-4o, and `bash` for Claude and DeepSeek; the latter two also lose repair performance. In Claude, unsuccessful injected trajectories can continue probing until the 25-turn budget is exhausted. The tool-use contrasts show that injection reshapes actions differently across models rather than inducing a common shift toward structured probing.

Limitations. The full falsification ladder is evaluated on Qwen; five-decoder-seed replication covers the main injection comparisons, not every control. Repair effects also vary by repository: Inject-Test's gain is concentrated in `django`, whereas Inject-Falsify retains a positive effect on the non-`django` subset. The cross-model analysis compares Base with Inject-Test on the same 43 instances across four models. The matching rank order (Spearman $\rho = +1.0$, $n = 4$) is therefore based on four model-level observations.

HypoCal measures agreement between a stated runtime prediction and the subsequent debugger observation, not the agent's internal beliefs. The stricter V1 scorer tests sensitivity to the matching rule without changing that measurement target. All experiments use `debug-gym` on SWE-bench Lite, assess resolution by hidden-test success, and impose a 25-turn budget. The Claude trajectories show that budget exhaustion accompanies the negative repair effect, motivating evaluation with longer interactions. Repeating the paired comparisons with longer budgets and on a fresh issue distribution such as SWE-bench Live would test how these action patterns transfer.

Conclusion. On Qwen, returning a hypothesis about the current bug that correctly localizes the fault to the next action prompt improves repair; elicitation alone yields no significant repair gain. Across four models, the repair effect varies with baseline probing behavior and the tool choices induced by injection. Together, these results support an action-shaping account of hypothesis prompting.

AI Use Statement

Generative AI tools were used to provide feedback on research methodology and experimental design, support literature retrieval and related-work discovery, assist with drafting and organizing selected manuscript passages, and improve language and readability. The authors reviewed all AI-assisted content and remain responsible for the paper's claims, analyses, and citations.

REFERENCES

Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Wang. SWE-Search: Enhancing software agents with Monte Carlo tree search and iterative refinement. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 64485–64515, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/a1e6783e4d739196cad3336f12d402bf-Paper-Conference.pdf.

Casey Barkan, Sidney Black, and Oliver Sourbut. Do large language models know what they are capable of? In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 15799–15822, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/1a96349bbc03432c5ec8c6c502d102e7-Paper-Conference.pdf.

Marcel Böhme, Ezekiel O. Soremekun, Sudipta Chattopadhyay, Emamurho Ugherughe, and Andreas Zeller. Where is the bug and how is it fixed? an experiment with practitioners. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pp. 117–128, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450351058. doi: 10.1145/3106237.3106255. URL https://doi.org/10.1145/3106237.3106255.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. CodeT: Code generation with generated tests. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=ktrw68Cmu9c.

Xinyun Chen, Maxwell Lin, Nathanael Schaerli, and Denny Zhou. Teaching large language models to self-debug. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 8746–8825, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/2460396f2d0d421885997dd1612ac56b-Paper-Conference.pdf.

Yongchao Chen, Harsh Jhamtani, Srinagesh Sharma, Chuchu Fan, and Chi Wang. Steering large language models between code execution and textual reasoning. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 99676–99707, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/f763f7c9a6599e14b07add5937d8189c-Paper-Conference.pdf.

Kazem Faghih, Wenxiao Wang, Yize Cheng, Siddhant Bharti, Gaurang Sriramanan, Sriram Balasubramanian, Parsa Hosseini, and Soheil Feizi. Tool preferences in agentic LLMs are unreliable. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20954–20969, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1060. URL https://aclanthology.org/2025.emnlp-main.1060/.

Spandan Garg and Yufan Huang. Debug2Fix: Supercharging coding agents with interactive debugging capabilities, February 2026. URL https://arxiv.org/abs/2602.18571.

Sandip Ghoshal, Anshul Mittal, Jyotika Singh, Miguel Ballesteros, Weiyi Sun, Fang Tu, Shailender Singh, Yassine Benajiba, Fahad Shah, Sujeeth Bharadwaj, Sujith Ravi, and Dan Roth. JTPRO: A Joint Tool–Prompt Reflective Optimization Framework for Language Agents. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 40573–40595, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.2017. URL https://aclanthology.org/2026.findings-acl.2017/.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In Doina Precup and Yee Whye Teh (eds.), *International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1321–1330. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/guo17a.html.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 32808–32824, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/8b4add8b0aa8749d80a34ca5d941c355-Paper-Conference.pdf.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/edac78c3e300629acfe6cbe9ca88fb84-Paper-Conference.pdf.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El-Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, Deep Ganguli, Danny Hernandez, Josh Jacobson, Jackson Kernion, Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom Brown, Jack Clark, Nicholas Joseph, Ben Mann, Sam McCandlish, Chris Olah, and Jared Kaplan. Language models (mostly) know what they know, 2022. URL https://arxiv.org/abs/2207.05221.

Sungmin Kang, Bei Chen, Shin Yoo, and Jian-Guang Lou. Explainable automated debugging via large language model-driven scientific debugging. *Empirical Software Engineering*, 30(2):45, 2025. doi: 10.1007/s10664-024-10594-x. URL https://link.springer.com/article/10.1007/s10664-024-10594-x.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=LJTLpEJPOU.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, pp. 611–626, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702297. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Haolin Li and Michael Coblenz. A grounded theory of debugging in professional software engineering practice. *Proc. ACM Softw. Eng.*, 3(FSE), June 2026. doi: 10.1145/3797077. URL https://doi.org/10.1145/3797077.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/aca97732e30bcf1303bc22ac3924fd16-Paper-Conference.pdf.

Stephanie Lin, Jacob Hilton, and Owain Evans. Teaching models to express their uncertainty in words. *Transactions on Machine Learning Research*, 2022. URL https://openreview.net/forum?id=8s8K2UZGTZ.

Mingwei Liu, Zihao Wang, Zhenxi Chen, Zheng Pei, Yanlin Wang, and Zibin Zheng. Dynamic analysis enhances issue resolution, 2026. URL https://arxiv.org/abs/2603.22048.

Tianyang Liu, Canwen Xu, and Julian McAuley. RepoBench: Benchmarking repository-level code auto-completion systems. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 47832–47850, 2024a. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/d191ba4c8923ed8fd8935b7c98658b5f-Paper-Conference.pdf.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 52989–53046, 2024b. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/e9df36b21ff4ee211a8b71ee8b7e9f57-Paper-Conference.pdf.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI Scientist: Towards fully automated open-ended scientific discovery, 2024. URL https://arxiv.org/abs/2408.06292.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Jim Fan, and anima anandkumar. Eureka: Human-level reward design via coding large language models. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 26516–26560, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/70c26937fbf3d4600b69a129031b66ec-Paper-Conference.pdf.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594. Curran Associates, Inc., 2023. doi: 10.52202/075280-2019. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91edff07232fb1b55a505a9e9f6c0ff3-Paper-Conference.pdf.

Potsawee Manakul, Adian Liusie, and Mark Gales. SelfCheckGPT: Zero-resource black-box hallucination detection for generative large language models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 9004–9017, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.557. URL https://aclanthology.org/2023.emnlp-main.557/.

Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947. doi: 10.1007/BF02295996. URL https://doi.org/10.1007/BF02295996.

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 *Proceedings of the 42nd International Conference on Machine Learning*, ICML'25. JMLR.org, 2025.

Ethan Perez, Sam Ringer, Kamile Lukosiute, Karina Nguyen, Edwin Chen, Scott Heiner, Craig Pettit, Catherine Olsson, Sandipan Kundu, Saurav Kadavath, Andy Jones, Anna Chen, Benjamin Mann, Brian Israel, Bryan Seethor, Cameron McKinnon, Christopher Olah, Da Yan, Daniela Amodei, Dario Amodei, Dawn Drain, Dustin Li, Eli Tran-Johnson, Guro Khundadze, Jackson Kernion, James Landis, Jamie Kerr, Jared Mueller, Jeeyoon Hyun, Joshua Landau, Kamal Ndousse, Landon Goldberg, Liane Lovitt, Martin Lucas, Michael Sellitto, Miranda Zhang, Neerav Kingsland, Nelson Elhage, Nicholas Joseph, Noemi Mercado, Nova DasSarma, Oliver Rausch, Robin Larson, Sam McCandlish, Scott Johnston, Shauna Kravec, Sheer El Showk, Tamera Lanham, Timothy Telleen-Lawton, Tom Brown, Tom Henighan, Tristan Hume, Yuntao Bai, Zac Hatfield-Dodds, Jack Clark, Samuel R. Bowman, Amanda Askell, Roger Grosse, Danny Hernandez, Deep Ganguli, Evan Hubinger, Nicholas Schiefer, and Jared Kaplan. Discovering language model behaviors with model-written evaluations. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13387–13434, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.847. URL https://aclanthology.org/2023.findings-acl.847/.

Qwen Team. Qwen3-Coder-30B-A3B-Instruct. Hugging Face Model Card, 2025. URL https://huggingface.co/Qwen/Qwen3-Coder-30B-A3B-Instruct.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 68539–68551. Curran Associates, Inc., 2023. doi: 10.52202/075280-2997. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/d842425e4bf79ba039352da0f658a906-Paper-Conference.pdf.

Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Michael Moor, Zicheng Liu, and Emad Barsoum. Agent Laboratory: Using LLM Agents as Research Assistants. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 5977–6043, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.320. URL https://aclanthology.org/2025.findings-emnlp.320/.

Mrinank Sharma, Meg Tong, Tomek Korbak, David Duvenaud, Amanda Askell, Sam Bowman, Esin DURMUS, Zac Hatfield-Dodds, Scott Johnston, Shauna Kravec, Timothy Maxwell, Sam McCandlish, Kamal Ndousse, Oliver Rausch, Nicholas Schiefer, Da Yan, Miranda Zhang, and Ethan Perez. Towards understanding sycophancy in language models. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 110–144, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/0105f7972202c1d4fb817da9f21a9663-Paper-Conference.pdf.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

Kaya Stechly, Matthew Marquez, and Subbarao Kambhampati. GPT-4 doesn't know it's wrong: An analysis of iterative prompting for reasoning problems. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023. URL https://openreview.net/forum?id=PMtZjDYB68.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 74952–74965. Curran Associates, Inc., 2023. doi: 10.52202/075280-3275. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/ed3fea9033a80fea1376299fa7863f4a-Paper-Conference.pdf.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2609–2634, Toronto, Canada, July 2023a. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.147. URL https://aclanthology.org/2023.acl-long.147/.

Ruocheng Wang, Eric Zelikman, Gabriel Poesia, Yewen Pu, Nick Haber, and Noah Goodman. Hypothesis search: Inductive reasoning with language models. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 38993–39014, 2024a. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/a934282496e7b907f5d48d49bb4c9d7d-Paper-Conference.pdf.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In Ruslan Salakhutdinov, Zico Kolter,

Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 50208–50232. PMLR, 21–27 Jul 2024b. URL https://proceedings.mlr.press/v235/wang24h.html.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Daniel 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 Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 65882–65919, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/a4b6ad6b48850c0c331d1259fc66a69c-Paper-Conference.pdf.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-Consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=1PL1NIMMrw.

Yunkun Wang, Yue Zhang, Guochang Li, Chen Zhi, Binhua Li, Fei Huang, Yongbin Li, and Shuiguang Deng. InspectCoder: Dynamic analysis-driven self repair through interactive LLM-debugger collaboration. *Proc. ACM Program. Lang.*, 10(OOPSLA1), April 2026. doi: 10.1145/3798238. URL https://doi.org/10.1145/3798238.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran Associates, Inc., 2022. doi: 10.52202/068431-1800. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying LLM-based software engineering agents. *Proc. ACM Softw. Eng.*, 2(FSE), June 2025. doi: 10.1145/3715754. URL https://doi.org/10.1145/3715754.

Jiahong Xiang, Xiaoyang Xu, Xiaopan Chu, Hongliang Tian, and Yuqun Zhang. Empowering autonomous debugging agents with efficient dynamic analysis. *Proc. ACM Softw. Eng.*, 3(FSE), June 2026. doi: 10.1145/3797126. URL https://doi.org/10.1145/3797126.

Miao Xiong, Zhiyuan Hu, Xinyang Lu, YIFEI LI, Jie Fu, Junxian He, and Bryan Hooi. Can LLMs express their uncertainty? an empirical evaluation of confidence elicitation in LLMs. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 23650–23678, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/6733cf15e10e2cd1d59af033c3bb8507-Paper-Conference.pdf.

Junjielong Xu, Qinan Zhang, Zhiqing Zhong, Shilin He, Chaoyun Zhang, Qingwei Lin, Dan Pei, Pinjia He, Dongmei Zhang, and Qi Zhang. OpenRCA: Can large language models locate the root cause of software failures? In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 84562–84590, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/d29b8d53678015079e1d245c023e49d2-Paper-Conference.pdf.

Xiangzhe Xu, Hamidreza Saghir, Qianhui Wu, Marc-Alexandre Côté, Tong Wang, Kiran Lakkaraju, Kexin Pei, and Xiangyu Zhang. The devil is in the interface: Evaluating how tool architecture shapes coding agent behavior, 2026. URL https://arxiv.org/abs/2608.11386.

John Yang, Carlos 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.

John Yang, Kilian Lieret, Carlos Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. SWE-smith: Scaling data for software engineering agents. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc., 2025. doi: 10.52202/085713-3239. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/8b86cf5ace600c48fd188efbb8dedec8-Paper-Datasets_and_Benchmarks_Track.pdf.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Xingdi Yuan, Morgane M Moss, Charbel El Feghali, Chinmay Singh, Darya Moldavskaya, Drew MacPhee, Lucas Caccia, Matheus Pereira, Minseon Kim, Alessandro Sordoni, and Marc-Alexandre Côté. debug-gym: A text-based environment for interactive debugging, 2025. URL https://arxiv.org/abs/2503.21557.

Andreas Zeller. *Why Programs Fail: A Guide to Systematic Debugging*. Morgan Kaufmann, 2 edition, 2009. ISBN 978-0-12-374515-6. doi: 10.1016/B978-0-12-374515-6.X0000-7. URL https://www.sciencedirect.com/book/monograph/9780123745156/why-programs-fail.

Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. RepoCoder: Repository-level code completion through iterative retrieval and generation. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 2471–2484, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.151. URL https://aclanthology.org/2023.emnlp-main.151/.

Linghao Zhang, Shilin He, Chaoyun Zhang, Yu Kang, Bowen Li, Chengxing Xie, Junhao Wang, Maoquan Wang, Yufan Huang, Shengyu Fu, Elsie Nallipogu, Qingwei Lin, Yingnong Dang, Saravan Rajmohan, and Dongmei Zhang. SWE-bench goes live! In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc., 2025. doi: 10.52202/085713-4923. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/d83c4a745789690f82e86d0ef752ae7c-Paper-Datasets_and_Benchmarks_Track.pdf.

A Reproducibility details

This appendix documents the implementation, data, analysis, and environment used for the experiments. The descriptions are provided to make the reported results auditable.

Code. The intervention is a $\approx$ 500-line subclass of `debug-gym`'s reactive agent that intercepts one auxiliary LLM call per turn for hypothesis elicitation and, in the Inject-$*$ modes, re-injects the result into the next action prompt; the action-decoding path is otherwise bit-identical to the baseline (Appendix B.4). The online HypoCal V0 scorer, offline HypoCal V1 scorer, configurations for all seven modes, matrix runner, and gold-patch integrity checks are described in the accompanying implementation details.

Data. The experiments comprise the Lite-300 matrix, the Inject-Text, Inject-OffInstance, and Inject-Wrong sweeps, and the four-model 43-instance cross-model runs. Each run records resolution status, per-turn elicited hypotheses, HypoCal scores, and token usage.

Analysis. The paired-McNemar, HypoCal, tool-use, cross-model action-gap, per-repository, cost, and failure-taxonomy analyses are described in the appendices.

Environment. All runs use `debug-gym` under a 25-turn / 1800 s budget. The primary Qwen evaluation uses the complete SWE-bench Lite split, while the cross-model comparison uses the shared 43-instance subset. The self-hosted Qwen3-Coder-30B-A3B model is served with vLLM 0.19.1. The gold-patch sanity sweep and its list of environment-invalid instances are used for the environment-validity analysis.

B METHOD DETAILS (EXTENDED SETUP)

These subsections expand the protocol summary given in §2 of the main body.

B.1 BASELINE: REACTIVE DEBUGGER AGENT

Our baseline (M0) is a thin wrapper around `microsoft/debug-gym`'s `FroggyAgent` (Yuan et al., 2025). At each turn t, the agent sees the bug report, the full history of prior tool calls and observations, and emits one tool call a_t via a single LLM call. The only implicit planning signal is the agent's own chain-of-thought, generated on the same LLM call as a_t. We call this purely reactive debugging: the agent can reflect about the bug, but nothing in the architecture makes reflection accountable to the debugger's verdict.

B.2 MAKING REFLECTION ACCOUNTABLE: HYPOTHESIS ELICITATION

A real human at a debugger does not just react to the next observation. They hold a theory, use a breakpoint or expression evaluation to check whether the theory predicts the state they see, and update. We build the same loop into the agent.

Before committing to an action, we make an auxiliary LLM call that asks the model to produce a structured runtime prediction:

```
Hypothesis (≤1 sentence):  why the test fails
Expected state:            {variable → predicted value}      (≤6 entries)
Next action intent:        English sentence about what to do
Confidence:                float ∈ [0, 1]
```

The JSON schema makes the prediction machine-readable, allowing the same output to be scored against the next observation and re-inserted into the subsequent action prompt.

B.3 HYPOCAL V0 — A PREDICTION–OBSERVATION AGREEMENT METRIC

The scoring of a hypothesis is done against the subsequent observation: after the action a_t runs, we parse the observation window for each predicted `(name, value)` pair and compute

$$\mathrm{HypoCal}(t) \;=\; \frac{1}{|E_t|} \sum_{(v,x)\in E_t} \mathbb{1}\big[\,\text{obs}_t \text{ contains } (v,x)\,\big]$$

using a `pdb`-aware parser that matches three observation patterns common in debug-gym output: bare assignments (`v = x`), dictionary pretty-prints (`'v': x`), and NumPy `array(...)`-wrapped values. HypoCal is local to a turn; we report $\overline{\mathrm{HypoCal}}$ averaged over turns where a hypothesis is emitted.

HypoCal is not an optimization target. It is a verifiable measurement that lets us compare prediction–observation agreement with repair success; their relationship is evaluated in §3.2.

B.4 IMPLEMENTATION

The three structured-prediction modes (M1–M3) share a single `HypothesisFroggyAgent` subclass that intercepts exactly one LLM call per step (the hypothesis call), overrides

`build_system_prompt` for the injection, and leaves the action-decoding path bit-identical to the vanilla baseline. We additionally set `tool_choice="required"` on every action call to force tool emission; without this, the LLM occasionally responds in prose to the injected hypothesis instead of calling a tool (the empty_tool_response failure mode, observed in <0.2% of turns even with the fix).

No additional tools, no additional memory, no additional fine-tuning. The total code diff against `debug-gym@v1.3.x` is roughly 500 lines. The full experimental stack runs on a single self-hosted Qwen3-Coder-30B-A3B-Instruct (Qwen Team, 2025) instance via vLLM 0.19.1 (Kwon et al., 2023) (`--max-model-len 131072`, `--enable-auto-tool-choice --tool-call-parser hermes`, $T = 0.2$, top-$p = 0.8$). The 1200-run Lite-300 matrix completes in approximately 70 h on one H200 with two parallel runners sharing the same vLLM instance, including a 15-min container reaper that clears zombie SWE-bench harness containers between tasks.

C Related work (extended)

This appendix expands the comparisons with scientific-debugging systems and research on self-assessment, hypothesis testing, and tool use.

C.1 Closest neighbours: hypothesis-driven and dynamic-analysis debuggers

AutoSD (Kang et al., 2025) implements the scientific-debugging sequence of hypothesis, prediction, experiment, observation, and conclusion (Zeller, 2009). It elicits a free-text hypothesis and prediction, executes a debugger experiment, and asks the LLM to judge the observation before continuing or generating a fix. AutoSD evaluates this sequence as an integrated protocol on Defects4J and ARHE, and uses BugsInPy in its developer study.

Inject-Text preserves the repair gain with free-text predictions; the structured schema enables deterministic scoring against subsequent debugger observations. Under the same debugger interface, Base, Elicit, Inject-Test, and Inject-Falsify separate hypothesis elicitation, action-time injection, and test-versus-falsify instructions. We study per-step agreement and action selection on SWE-bench Lite and compare repair effects across four models.

Debug2Fix (Garg & Huang, 2026) introduces a debugging subagent that answers runtime questions for a main repair agent on SWE-bench Live and GitBug-Java. Its delegation interface and tool gating change how the main agent accesses runtime evidence. Our comparisons instead hold the debugger interface fixed and vary the hypothesis supplied to the main agent's action prompt, isolating a prompt-side intervention from the architectural change.

InspectCoder (Wang et al., 2026) uses a Driver/Inspector architecture to intervene in intermediate runtime values and exploit debugger feedback. It evaluates repair on BigCodeBench-R and LiveCodeBench-R. Our content controls vary whether the injected hypothesis concerns the current bug and correctly localizes the fault; HypoCal and tool traces measure prediction–observation agreement and subsequent action changes. Table 7 summarizes the comparison.

C.2 Self-assessment, sycophancy, and explanation faithfulness

Research on model self-assessment examines whether an LLM can evaluate its answers or anticipate task success (Kadavath et al., 2022; Xiong et al., 2024; Barkan et al., 2026). Our per-step measure concerns a different prediction target: the runtime state observed after a debugger action.

Sycophancy studies examine responses to views expressed in prompts (Sharma et al., 2024; Perez et al., 2023). Inject-Test and Inject-Falsify instead vary the instruction accompanying the agent's own hypothesis, while the content controls vary whether that hypothesis concerns the current bug and correctly localizes the fault. These comparisons separate the framing of a hypothesis from the information carried by its content.

Studies of self-verification examine iterative critique and the limits of self-correction without external feedback (Stechly et al., 2023; Huang et al., 2024). Process supervision evaluates intermediate reasoning, and self-consistency aggregates reasoning paths (Lightman et al., 2024; Wang et al.,

2023b). Work on unfaithful chain-of-thought explanations distinguishes a stated rationale from the process leading to an answer (Turpin et al., 2023). Our `astropy-14995` comparison makes a related distinction at the action level: Elicit and Inject-Falsify reach the same diagnostic depth and measured HypoCal but choose different edits, and only the latter passes the hidden test (Appendix D.2.3). HypoCal scores whether stated variable values match the next debugger observation. We compare this step-level agreement with paired repair and the actions actually chosen.

C.3 Scientific-method framings for LLM reasoning

Scientific-research agents generate hypotheses, test candidates, and design experiments using external procedures (Schmidgall et al., 2025; Lu et al., 2024; Ma et al., 2024; Wang et al., 2024a). In the debugging setting, our matched comparisons distinguish generating a hypothesis from returning it to the agent's next action prompt; content controls and tool traces then examine instance grounding and action selection (§3.1–§3.3).

C.4 Plan-and-act and ReAct

ReAct interleaves reasoning and tool actions (Yao et al., 2023); Plan-and-Solve structures reasoning before solution generation (Wang et al., 2023a), and Reflexion uses feedback-derived reflections across trials (Shinn et al., 2023). Chain-of-thought and Self-Refine examine intermediate reasoning and iterative self-feedback (Wei et al., 2022; Madaan et al., 2023). Elicit generates and records a hypothesis without returning it to the action prompt, whereas Inject-Test and Inject-Falsify supply the hypothesis before the next action. The matched comparison distinguishes an additional reasoning call from action-time conditioning under unchanged tools.

C.5 Adjacent tasks and benchmarks

Agentless studies issue localization, repair, and validation (Xia et al., 2025). SWE-Gym and SWE-smith develop training environments and data for software agents (Pan et al., 2025; Yang et al., 2025), while SWE-bench Live evaluates issue resolution on an evolving task distribution (Zhang et al., 2025). RepoBench and RepoCoder study repository-level code completion, AgentBench evaluates interactive agent capabilities, and OpenRCA uses logs, metrics, and traces for root-cause analysis (Liu et al., 2024a; Zhang et al., 2023; Liu et al., 2024b; Xu et al., 2025). Our task is patch-based issue resolution on SWE-bench Lite (Jimenez et al., 2024). Our contribution is a controlled decomposition of hypothesis prompting in this repair setting under a fixed `debug-gym` interface (Yuan et al., 2025).

D Extended Lite-300 results

Full per-repo, prediction–observation agreement, cost, robustness, and limitations material. Headline numbers and the falsification ladder are in §3 of the main body.

D.1 Per-repo: where injection actually lives

The aggregate +8pp top-line conceals strong repo-level heterogeneity. The per-repo table below (with the heatmap and forest plot that follow) shows resolution per repo for each mode.

Table 8: Per-repository resolution rates on SWE-bench Lite for the four Qwen modes. Bold marks the highest resolution rate within each repository.

Repo	n	M0	M1	M2	M3
django	114	28.9%	30.7%	**41.2%**	37.7%
sympy	77	9.1%	10.4%	11.7%	**14.3%**
matplotlib	23	8.7%	**26.1%**	21.7%	17.4%
scikit-learn	23	34.8%	30.4%	34.8%	**47.8%**
pytest-dev	17	0.0%	0.0%	5.9%	5.9%
sphinx-doc	16	6.2%	12.5%	6.2%	12.5%
astropy	6	16.7%	16.7%	0.0%	16.7%
pylint-dev	6	0.0%	0.0%	0.0%	16.7%
psf	6	16.7%	16.7%	16.7%	33.3%
pydata	5	20.0%	0.0%	20.0%	40.0%
mwaskom	4	25.0%	25.0%	25.0%	25.0%
pallets	3	0.0%	0.0%	33.3%	0.0%

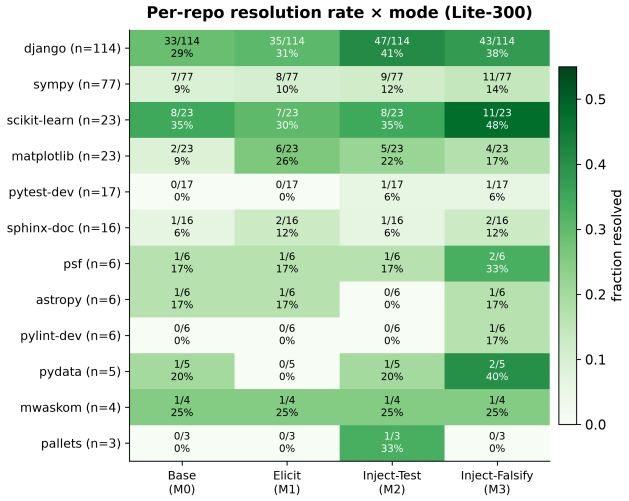

Figure 6: Per-repository resolution heatmap across the four Qwen modes. Color intensity is proportional to resolution rate; bold cells mark the best mode within each repository.

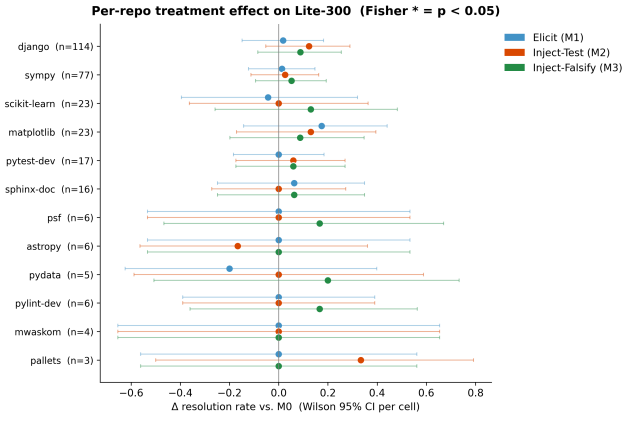

Figure 7: Per-repository paired resolution differences relative to M0 with 95% bootstrap CIs. Most repository-level CIs include zero; the largest positive M2/M3 differences occur in django, scikit-learn, and the small pydata/psf strata.

Three observations:

(a) The aggregate +8pp on M3 is concentrated in four repos. M3 gains $\geq +10$pp over M0 on `scikit-learn` (+13.0), `psf` (+16.7), `pydata` (+20.0), and `pylint-dev` (+16.7). On `django`, M2 leads M3 by +3.5pp; on `matplotlib`, M1 (!) leads both injection modes. The M2–M3 contrast is therefore not uniformly null across repositories. Where M2 wins (`django`, `matplotlib`, `pallets`) the bug tends to be reachable by reading source; where M3 wins (`scikit-learn`, `pydata`, `psf`) the bug tends to require numerical state probing that a "falsify" directive encourages more aggressively.

(b) M1 is not uniformly null. On `matplotlib` (n=23) M1 lifts resolution from 9% to 26%, the strongest M1 effect in the matrix. This is consistent with M1 being a "soft prior" that helps when the LLM's pre-existing prior is almost right — elicitation acts as a structural reminder rather than a behavioural intervention. We flag this as a domain-dependent nuance rather than a contradiction of §3.2: across the full benchmark M1 is indistinguishable from M0.

(c) M2 is heavily django-driven; M3 is broadly applicable. A formal paired sensitivity (broken instances excluded) confirms this: on the n=180 non-django subset, M2's paired Δ over M0 shrinks to $+3.33$pp (paired McNemar exact $p = 0.263$, n.s.), while on django alone (n=114) M2 paired Δ is $+12.28$pp ($p = 0.0026$, highly significant). On django, M2 actually beats M3 (+12.28 vs. +8.77pp, both vs. M0); on non-django M3 retains a $+7.78$pp paired lift ($p = 0.0013$). The pragmatic take-away is that M2 is the right intervention on bug populations dominated by Django-style state-machine and ORM-flow defects, while M3 is the right intervention when the bug population is heterogeneous or numerical (sympy's $+5.33$pp M3 lift, sci-kit-learn's $+13$pp M3 lift). We do not interpret the django-specificity of M2 as a "spurious lift" — django is a statistically reasonable ($\sim 38\%$ of Lite-300) sub-corpus and the M3 lift is broad — but we do flag it as an engineering recommendation: one mode does not fit all repos.

D.2 Prediction–observation agreement at scale

We hypothesised, before running the matrix, that higher prediction–observation agreement would accompany more resolved bugs. The data contradicts this in three ways.

D.2.1 Mode-level: HypoCal is flat despite a 6pp resolution gap

Table 9: Mode-level HypoCal V0 statistics for the three hypothesis-aware modes on Lite-300.

Mode	n	Scored steps	Per-step avg. match	Per-instance avg.	Mean confidence
M1 hypothesis-observe	299	6311	0.023	0.021	0.913
M2 hypothesis-inject	299	6504	**0.034**	0.034	0.916
M3 hypothesis-falsify	299	6524	**0.034**	0.033	0.915

The two modes that resolve most are also the two modes whose HypoCal is highest, but the spread is only +1.1pp and all three scores remain near floor ($\leq 3.4\%$). Mean stated confidence remains at ≈ 0.91 across all three modes. Measured prediction–observation agreement therefore remains low despite substantial differences in repair outcomes.

D.2.2 Run-level: high HypoCal does not imply success

Of the 193 runs (across M1+M2+M3) that scored at least one HypoCal point above 0.20, only 27% resolve — statistically indistinguishable from the hypothesis-aware base rate of 24% (the HypoCal–resolution scatter is Figure 3 in §3.2; the per-mode distribution is shown below). A literally-checkable hypothesis in this setting is not a marker of agent skill.

D.2.3 Within-instance counter-example: same hypothesis, weaker fix

A representative qualitative case is `astropy-14995` (drawn from a pilot run for trace fidelity) where M1 and M3 trajectories reach exactly the same diagnostic depth (both identify

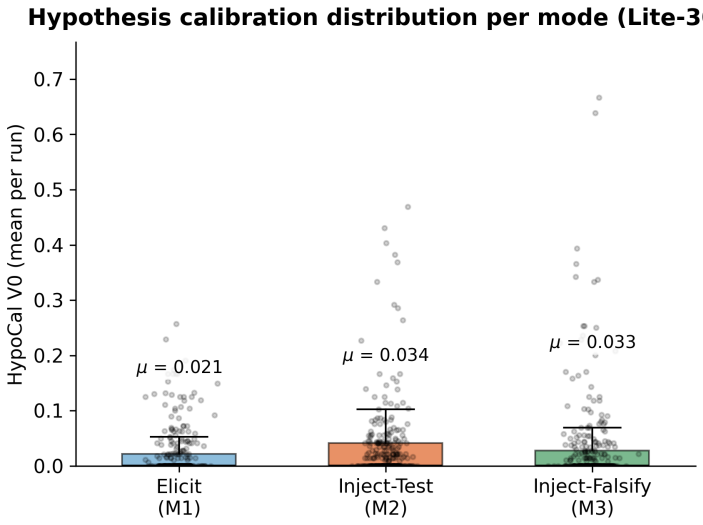

Figure 8: Distribution of per-run HypoCal V0 across Elicit, Inject-Test, and Inject-Falsify. Prediction–observation agreement remains near floor in all three modes.

`_arithmetic_mask` and the `bitwise_or(int, None)` failure path, both score HypoCal ≈ 0.33 on the relevant variable-state predictions) yet the M1 agent patches only one of the two `None`-branches and fails the hidden test, while the M3 agent rewrites both branches and passes. Measured prediction–observation agreement was not the discriminator; the selected edit was.

This case shows that identical measured prediction–observation agreement can lead to different edits and different repair outcomes.

D.2.4 Robustness to a stricter scorer (HypoCal V1)

The V0 scorer (Appendix B.3) is intentionally lenient — it runs online, before the agent's next turn, and counts a match whenever a predicted variable name and a token of its predicted value co-occur in the observation, with no action-tool gating and no numeric tolerance. We therefore test whether the low prediction–observation agreement persists under a stricter scorer.

We therefore re-scored every step of every run with **HypoCal V1**, an offline re-scorer that (i) gates on the next action tool — only `eval`/`pdb`/`bash`/`python` steps are counted, since `view`/`edit` actions do not produce runtime state to predict against; (ii) extracts `name = value` pairs from the observation and applies numeric tolerance (rel $< 10^{-3}$, abs $< 10^{-6}$, plus type-flexible list/tuple/dict matching); (iii) handles special prediction shapes (`raises FooError`, `shape=(a, b)`, `length N`); (iv) returns continuous $[0, 1]$ partial credit per variable rather than V0's binary decision. Because V1 scores only this action-gated subset, we recompute V0 over the same steps so that the two columns are directly comparable. These V0 values therefore differ from the all-steps V0 reported in Table 2 and Appendix D.2.1:

Table 10: HypoCal V0 and V1 evaluated on the same action-gated steps. V1 applies the stricter offline matching rule described in Appendix D.2.4.

Mode	HypoCal V0 (action-gated steps)	HypoCal V1 (same steps)	Δ (V1 − V0)
M1 hypothesis-observe	1.78%	0.81%	−0.97pp
M2 hypothesis-inject	3.46%	1.13%	−2.33pp
M3 hypothesis-falsify	3.82%	1.42%	−2.40pp

V1 is uniformly lower than V0 on the same action-gated steps. The stricter observation matching removes loose co-occurrence matches admitted by V0; because the denominator is held fixed, the differences in the table reflect the scoring rule rather than a change in the evaluated step set.

Low hypothesis–observation agreement persists under V1: Elicit, Inject-Test, and Inject-Falsify score 0.81%, 1.13%, and 1.42%, respectively. The robustness check therefore concerns the low agreement itself; its relationship with repair is evaluated separately in §3.2.

D.2.5 Action distribution: the operative mechanism

The per-turn action distribution provides direct evidence about how the intervention changes tool selection. The table below (the extended form of Table 4 in §3.3) gives the proportion of action calls assigned to each tool, summed over the matrix.

Table 11: Per-mode action distribution and total action count on Lite-300. Values are shares of tool calls; M0 values are approximate pilot estimates.

Mode	view	eval	pdb	edit	bash	Σ actions
M0 vanilla	≈53%	≈22%	≈1%	≈14%	≈9%	≈7100†
M1 hypothesis-observe	**53.0%**	20.3%	2.5%	15.3%	8.7%	6960
M2 hypothesis-inject	**42.5%**	**26.6%**	**8.4%**	11.8%	10.7%	7106
M3 hypothesis-falsify	46.3%	26.1%	4.3%	10.6%	12.6%	7158

† M0 numbers approximated from a pilot matrix (M0 does not log hypotheses); the action distribution there is statistically indistinguishable from M1 at $p > 0.3$ by χ^2. The corresponding tool-budget figure is Figure 4 in §3.3.

The largest changes occur in the distribution of runtime-probing actions:

- M2 emits 3.4× more `pdb` calls than M1 (8.4% vs 2.5%) and 31% more `eval` calls than M1.
- M2 spends 10pp less of its action budget on passive `view` than M1, redirecting it toward state-probing tools.
- M3 lies between M1 and M2 on `pdb` (4.3%) but matches M2 on `eval` (26.1%). Its larger boost on `bash` (12.6%) plausibly reflects exploratory `python -c` probes encouraged by the falsify directive.

Per-turn this means the M2 agent inspects roughly 2.0 pdb breakpoints per task (versus 0.6 for M1); the M3 agent inspects 1.0. Every additional `eval` or `pdb` call exposes the agent to additional runtime evidence. The main observable change is therefore at the agent–environment interface: injection redistributes tool use toward runtime probing. This matches §3.3, where the redistribution is already present after zero-HypoCal turns and the high-HypoCal tail adds only a small amount of additional probing.

D.3 Cost

We measured per-task token usage by parsing the recorded token-usage fields from every prompt/response pair in each trajectory. On Qwen Lite-300 (n=294 after env-validity exclusion):

Table 12: Per-task and aggregate token usage for the environment-valid Qwen Lite-300 runs ($n = 294$).

Mode	Turns (mean)	Input tok. / task (mean)	Output tok. / task (mean)	Total input tokens	Total output tokens
M0	23.5	638k	3.65k	161.4M	0.93M
M1	23.3	617k	3.96k	156.6M	1.04M
M2	23.7	640k	2.95k	168.2M	0.81M
M3	23.9	672k	3.30k	179.3M	0.94M

The marginal injection cost is modest: M2 spends $+4.2\%$ input tokens and fewer output tokens than M0 (the hypothesis focuses subsequent reasoning, so per-call response lengths shrink). M3 spends $+11\%$ input tokens. Both buy +37%–+44% relative improvement in resolution; token cost per resolved task drops by 24% (M2) and 23% (M3) on Qwen.

On the Claude 43-subset (n=42 after exclusion, both modes capped at 25 turns), token usage is lower in M2 than M0 (357k vs. 399k input, 3.30k vs. 4.02k output). Mode-pair total cost: \$47.17 (M2) vs. \$52.75 (M0) at Anthropic list pricing — i.e. M2 is 11% cheaper per task than M0 on Claude. Because Claude's M2 repair effect is negative on this subset, \$/resolved is essentially equal ($\$1.75$ vs. $\$1.76$). This rules out a "the lift is just paying for more tokens" reading of the Qwen result: the injection intervention adds no measurable per-task overhead in the regime where it does not help.

D.4 Robustness to environment-validity gating

We ran a gold-patch sanity sweep across all 300 SWE-bench Lite instances: for each instance we spin up the official container, apply the dataset's gold patch, and require the FAIL-TO-PASS tests to pass. 294 / 300 instances pass; 5 fail (gold patch applied but FAIL-TO-PASS still fails: `psf__requests-{2148, 2317, 2674}`, `sympy__sympy-{13146, 13177}`); 1 trips an upstream debug-gym assertion that the eval tool resolves before the gold patch is applied (`psf__requests-863`) — i.e. its FAIL-TO-PASS test passes regardless of the patch, so any "resolved" report on this instance in any of our four modes is a false positive. We therefore re-run the headline analysis after excluding all 6 instances:

Table 13: Resolution rates and paired effects before and after environment-validity gating. The robust analysis excludes the six instances that fail the gold-patch sanity check.

Mode	Original ($n = 300$)	Robust ($n = 294$)	$\Delta_{M_i - M_0}$ paired	McNemar exact p
M0	18.33%	**18.37%**	(baseline)	—
M1	20.33%	**20.41%**	+2.04pp	0.345
M2	25.00%	**25.17%**	+6.80**pp**	**0.0022**
M3	26.33%	**26.53%**	+8.16**pp**	**0.0005**

All four mode rates move by ≤ 0.20pp; both M2 and M3 paired effects shift slightly upward (the always-unresolved broken instances were dragging M0/M1/M2/M3 down equally, while the spurious-resolve instance was inflating each by exactly 1 spurious win). McNemar discordant counts b, c are unchanged because the broken instances are uniformly resolved or uniformly unresolved across modes; both p-values are therefore exact under both denominators. The cross-model Claude result (§3.4) is similarly robust: the 43-subset overlaps the broken set on `psf__requests-2148` only, which is unresolved under both Claude M0 and Claude M2 — the paired Δ shifts marginally from -6.98pp to -7.14pp, p unchanged at 0.549.

We retain the original $n = 300$ results as the headline and report the $n = 294$ analysis as an environment-validity check. Excluding the six instances changes each mode's resolution rate by at most 0.20pp, while preserving the paired improvements. The six excluded instances are listed above.

D.5 Limitations

- **Models covered.** Quantitative cross-model results span four models — GPT-4o-2024 ($M_0 = 9.3\%$), Qwen3-Coder-30B ($M_0 = 18.3\%$ on full Lite-300; 18.6% on the 43-subset), DeepSeek-V4 ($M_0 = 37.2\%$), and Claude-Sonnet-4.6 ($M_0 = 69.8\%$) — covering a $7\times$ range of baseline strength. Each tier still rests on a single model; a second model at each baseline-probing level would tighten the action-gap fit (Spearman $\rho = +1.0$ now rests on $n = 4$). The cross-model replication is M0 + M2 only on $n = 43$; extending it to M1 + M3 and to a larger budget per model is the obvious next step.
- **Seed coverage is uneven across modes.** Base, Inject-Test, and Inject-Falsify are replicated across the full multi-seed sweep (§3.1), with the Inject-Test lift positive across all reported seeds. The full four-condition matrix, which adds the Elicit control, is available on fewer seeds; Elicit does not differ significantly from Base in those replications. Because effect

sizes vary across seeds, we report the multi-seed mean rather than the primary-seed result as the headline.
- HypoCal V0 matches predicted variables literally inside a ≈ 1.5 KB observation window. Free-form descriptive predictions ("a bool array of all True") score zero even when correct. HypoCal measures agreement between explicit predictions and subsequent observations. Low agreement also persists under V1 (Appendix D.2.4), but this check does not validate HypoCal as a general measure of the agent's understanding.
- **Wall-clock budget.** 12 of 300 instances time out under M0 and 10–13 under hypothesis modes. The `pylint-dev` / `pytest-dev` / `sphinx-doc` repos are particularly affected.
- **No human-quality patch comparison.** Success is hidden-test pass; we have not graded patch quality against the gold patch.

D.6 Cross-model and per-turn analyses (extended §3.3, §3.4)

D.6.1 Failure taxonomy of Claude's negative-action-gap losses

To put a qualitative face on Claude's negative action-gap, we read all 7 Claude trajectories where M0 resolved but M2 did not (7/43; 16% of the subset, against 4/43 wins in the opposite direction). Every one of the 7 losses exhausts the 25-turn budget (median turns: 25/25 in losses vs. ≤ 21 in wins). The distribution of failure modes is:

Table 14: Failure taxonomy for the seven Claude cases in which Base resolves the issue but Inject-Test does not on the shared 43-instance subset.

Failure pattern	# cases
Over-probing the same hypothesis past the edit point	4 / 7
Looping on partial / contradictory evidence	2 / 7
Tooling friction during runtime probing	1 / 7

Representative examples of each pattern:

- **Over-probing past the edit point** — `astropy__astropy-14182`: 23 of 25 turns are `bash python -c` probes of `header_rows` semantics; the agent never reaches an edit. The hypothesis is correct but the budget is exhausted on re-confirmation rather than on patching.
- **Looping on partial / contradictory evidence** — `scikit-learn__scikit-learn-14092`: turns 21–25 are 5 successive `python3 -c "import numbers; print(...)"` checks of the type hierarchy that re-confirm the same fact under slight variations, without converging on an edit.
- **Tooling friction during runtime probing** — `sympy__sympy-22005`: turns 21–25 alternate `bash python -c` and `view` while fighting an `ImportError` on an internal helper that M2's hypothesis directs the agent to inspect — the agent loses the budget on environment recovery rather than diagnosis.

The wins (4/43) tell a complementary story: 3 of 4 are django defects where M0 — without an explicit hypothesis — runs out of turns on broad search, while M2 lands directly on the relevant test under guidance. We read the loss vs. win asymmetry as Claude's negative action-gap manifesting through the turn budget: the intervention shifts probability mass toward longer probing chains, which on a strong baseline that already commits early in M0 means the M2 budget is more often spent without reaching an edit. The wins go in the same direction (more probing, more runtime evidence) but on the bug subset where M0's broad-search strategy was the actual failure mode. A larger Claude budget (say, $T_{\max} = 35$ instead of 25) would predict the -7pp Δ to attenuate, since the 4/7 over-probing losses would have time to land an edit; we leave this test to future work. The seven trajectories and their turn-by-turn timelines are summarized in this appendix.

D.6.2 CROSS-MODEL HYPOCAL AND REPAIR EFFECTS

If the Qwen repair gain were driven by higher prediction–observation agreement, models with higher HypoCal should also show larger repair effects. We therefore compare per-step HypoCal V0 with the paired repair effect across the four models:

Table 15: Cross-model HypoCal and repair effects. Qwen reports the Lite-300 reference result; GPT-4o, Claude, and DeepSeek are evaluated on the shared 43-instance subset.

Model	M0 → M2 res.	Δ_{M2}	Per-step HypoCal V0	Per-instance HypoCal V0	Mean confidence	Scored-step frac.
GPT-4o-2024	9.3% → 20.9%	+11.6pp	**2.79%**	2.53%	0.84	62.7%
Qwen3-Coder-30B (Lite-300, ref.)	18.3% → 25.0%	+6.7pp	3.46%	3.36%	0.92	91.5%
Claude-Sonnet-4.6	69.8% → 62.8%	−7.0pp	**4.00%**	4.55%	0.93	48.3%
DeepSeek-V4	37.2% → 16.3%	−20.9pp	**1.81%**	1.77%	0.93	79.1%

Prediction–observation agreement remains low across models. HypoCal V0 lives in $[1.81\%, 4.00\%]$ across a 7× range of baseline strength; mean self-reported confidence lives in $[0.84, 0.93]$.

HypoCal does not track Δ_{M2} across models. On the first three models, HypoCal even rank-anti-correlates with the lift; Claude is highest at 4.00% but has the most negative Δ_{M2}, while GPT-4o is lowest at 2.79% but has the largest positive effect. DeepSeek-V4 has the lowest HypoCal (1.81%) and the most negative Δ_{M2} (−20.9pp). Across all four points, measured prediction–observation agreement remains low while the resolution effect ranges over 32pp. By contrast, the observed repair-effect ordering matches baseline structured-probing rates (§3.4).

Within-model, resolved instances show only a marginal agreement gap. Splitting per-instance HypoCal by resolved status gives GPT-4o resolved 2.90% vs. unresolved 2.42% (gap +0.48pp on $n_r = 9, n_u = 33$) and Claude resolved 5.03% vs. unresolved 3.57% (gap +1.46pp on $n_r = 27, n_u = 13$); both gaps are well below each model's Δ_{M2} magnitude. The per-model HypoCal numbers and the resolved-vs-unresolved splits are reported here.

D.6.3 PER-TURN HYPOCAL → NEXT-TURN ACTION

We pair, for every M2 turn, $(\text{HypoCal}_t, a_{t+1})$ with $a_{t+1} \in \{\text{pdb}, \text{eval}\}$ the "active-probing" indicator. Over all 6 222 scored turns of the Lite-300 M2 matrix the pattern is conditional, not linear: 95% of turns have $\text{HypoCal}_t = 0$ and sit at the unconditional base rate; the small upper tail with $\text{HypoCal}_t \in (0, 1]$ ($\approx 6\%$ of turns) shows a +10–14pp lift in active probing, peaking at +13.8pp ($p = 0.006$) when $\text{HypoCal}_t = 1$. M3 reproduces this with a larger tail (+19.8pp at $\text{HypoCal}_t = 1$, $p < 10^{-4}$); M1 reproduces it at attenuated magnitude (Figure 9).

This separates the two observed patterns: the main action redistribution is already present at HypoCal $= 0$, while the small upper tail shows additional active probing. The full per-turn contingency tables are reported here.

D.6.4 PER-MODEL ACTION REDISTRIBUTION (EXTENDED §3.4)

Action redistribution is present across all four models, but its direction is model-specific. As in Table 6, we report the largest decrease and increase among the six tool categories together with the paired repair-rate change on the same 43-instance subset. All changes are computed as Inject-Test minus Base and expressed in percentage points:

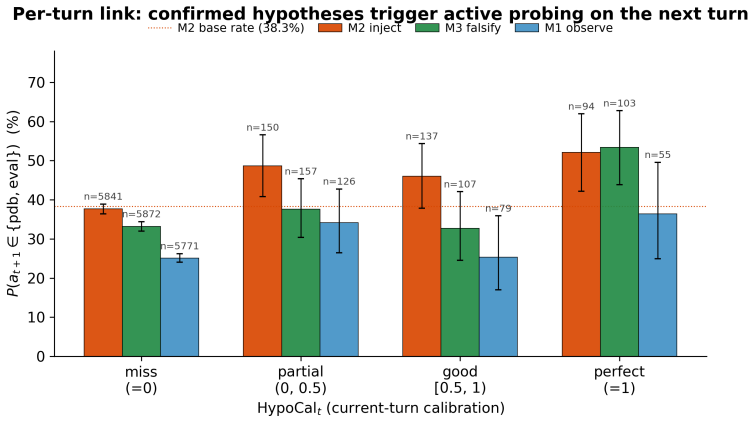

Figure 9: Association between current-turn HypoCal and next-turn active probing. Bars show $P(a_{t+1} \in \{\texttt{pdb}, \texttt{eval}\})$ within each HypoCal bin with Wilson 95% CIs; the dotted line marks the unconditional M2 base rate. Upper-tail bins show higher active-probing rates, while the majority of zero-HypoCal turns remain at the base rate.

Table 16: Largest tool-share changes and paired repair effects for Base → Inject-Test on the shared 43-instance subset. All changes are in percentage points.

Model	Largest decrease (pp)	Largest increase (pp)	Repair Δ (pp)
GPT-4o-2024	`bash` −5.2	`view` +7.8	+11.6
Qwen3-Coder-30B	`view` −8.7	`pdb` +5.6	+4.7
Claude-Sonnet-4.6	`other` −17.8	`bash` +19.0	−7.0
DeepSeek-V4	`edit` −7.3	`bash` +13.4	−20.9

The largest tool-share changes differ between the models that gain and lose repair performance. GPT-4o reduces `bash` use and increases `view`, while Qwen reduces `view` and increases runtime probing (`pdb` +5.6pp and `eval` +5.0pp). Claude and DeepSeek instead increase their already-dominant `bash` share by 19.0pp and 13.4pp, respectively, alongside negative repair effects. DeepSeek has the largest repair loss (−20.9pp): its `bash` share rises from 54.8% to 68.2%, while its `edit` share falls from 8.5% to 1.2%. These are changes in the composition of tool calls; they do not by themselves establish which actions caused the repair losses.

Together with the action-gap comparison (§3.4) and the Claude trajectory analysis (Appendix D.6.1), these results support a bounded action-shaping account: injection changes the model's action policy, and the repair outcome appears to depend on how that change interacts with its existing debugging strategy. The aggregate tool shares document the behavioral contrast; the trajectory analysis provides evidence of one possible failure mode, over-probing within a fixed turn budget.

The complete cross-model tables are reported in this appendix. Two caveats apply: (a) due to external-API rate limits the cross-model run is M0 + M2 only, leaving M1 and M3 cross-model generalization for future work; (b) $n = 43$ is small for detecting per-model effects below $\sim |5|$ pp at $\alpha = 0.05$ — a larger evaluation across models would tighten the per-model CIs and test whether the observed rank ordering persists beyond these four model-level points.

E Extended discussion

Detailed treatment of candidate mechanisms, threats, scope statements, and future work. The synthesised position is in §5.

E.1 What HypoCal measures

A high HypoCal score requires three things to coincide:

1. The agent's hypothesis names a runtime variable, not a high-level abstraction.
2. The predicted value can be canonically rendered in the observation window (a string, a small array, etc.).
3. An action shortly thereafter actually probes that variable.

The first two are properties of the hypothesis; the third is a property of the action. In our matrix, M2 and M3 score hypotheses at 0.034 (vs M1's 0.023) — a small lift driven by modestly more probe-friendly hypotheses. But none of the three modes achieve match rates above the floor of 4%, and confidence sits at 0.91 in all modes, a uniform overconfidence pattern.

HypoCal therefore depends jointly on the stated runtime prediction and the observation produced by the selected action. It measures prediction–observation agreement rather than general reasoning quality.

E.2 Why M2 and M3 tie

We predicted M3 $\succ$ M2 because the falsify framing explicitly counters the confirmation-bias mechanism we expected to limit M2. The Lite-300 paired result shows no advantage for M3 ($p = 0.67$). The per-repo breakdown of §3.4 shows that the relative effects of M2 and M3 vary across repositories: M2 and M3 win on non-overlapping instance subsets. M2 dominates `django` (+12pp over M0) and `pallets` (+33pp). M3 dominates `scikit-learn` (+13pp), `pydata` (+20pp), `psf` (+17pp), and `pylint-dev` (+17pp). The two framings are best understood as different prompt-induced distributions over the same action space, with neither distribution being globally dominant for our agent and budget.

The evidence does not support choosing one framing over the other. It supports a narrower conclusion: at least one hypothesis-injection framing produces a robust lift, and the lift is mechanistically explainable as a tool-use shift rather than as a model-cognition upgrade.

E.3 Threats to the mechanism story

Injection changes both the content and length of the action prompt. The matched content controls distinguish the role of hypothesis content from prompt length alone. Inject-Text, Inject-OffInstance, and Inject-Wrong hold the injection slot, length, and decoder fixed while varying the injected hypothesis. A correctly localized hypothesis retains the repair gain, whereas off-instance and mislocalized hypotheses do not (§3.3).

A second concern is that injection adds an LLM call per turn (the hypothesis call itself), and the lift is therefore "pay-to-think". M1 controls for this exactly — it has the same extra LLM call but no injection — and obtains no lift. The extra call alone is insufficient.

A third concern is that HypoCal V0 is too floor-saturated to support the agreement analysis. Appendix D.2.4 examines this concern using a stricter offline scorer (HypoCal V1), which also yields low scores across all three modes. The low agreement therefore cannot be attributed solely to V0's particular matching rules. We also rely on confidence (which is bounded above) and on per-run scatter (Figure 3), neither of which are floor-bound.

E.4 Scope of the claims

Our claims are limited to the tested protocol and environment. We interpret HypoCal as a runtime prediction–observation measure rather than a general measure of reasoning, and we do not infer that the prompt induces scientific reasoning in the model. Transfer to fine-tuned debugger agents, larger turn budgets, or less curated benchmarks remains an open question.

The evidence supports one small, replicable, mechanism-identifiable prompt change that lifts SWE-bench Lite resolution by $+6.7$ to $+8.7$pp on the primary seed (37–47% relative, $p \leq 0.0022$), with the Inject-Test effect remaining positive across the full multi-seed sweep (§3.1); on Qwen, correctly

localized injection is accompanied by a redistribution from `view` toward `eval`/`pdb`, whereas Inject-OffInstance retains neither the repair gain nor this probing shift (§3.3); and injection reshapes tool use in model-specific directions and produces negative repair effects on Claude and DeepSeek, where additional actions are redirected toward `bash`. Across the tested analyses, repair success does not track prediction–observation agreement.

E.5 FUTURE WORK

- **Densifying the action-gap curve.** The four-model action-gap result (§3.4) covers baseline structured-probing rates of $\{0.03, 0.14, 0.25, 0.37\}$ pdb+eval per turn. More models spread along this axis would help locate the probing rate at which Δ_{M2} changes sign and would clarify when the intervention should be enabled for a given model.
- **Fresh-distribution replication.** SWE-bench Live is the next natural benchmark target for testing whether the paired effect generalizes beyond the fixed SWE-bench Lite snapshot. At our measured throughput (1.5–3 min / task at $2\times$ concurrency on a single H200), a comparable-scale replication would take roughly one workday of compute per mode.
- **Higher turn budgets on negative-gap baselines.** The Claude -7pp repair effect (§3.4) was measured at the same 25-turn budget as Qwen. A larger turn budget may attenuate the negative gap when Claude is given the 30–35 turns it normally needs for the multi-step pdb sessions we observed in its M2 failure taxonomy (§D.6.1).
- **Curriculum & distillation.** The observed action shift suggests a possible target for future SFT or DPO fine-tuning. Inject-Test / Inject-Falsify / Inject-Text traces provide a natural corpus for a debugger-specific Qwen3 fine-tune.