

Who Should Own the Loop? Harness Decomposition for Small-Model Repository Repair

Francesco Dente^{*✉}

EURECOM
Biot, France
francesco.dente@eurecom.fr

Dario Satriani^{*✉}

University of Basilicata
Potenza, Italy
dario.satriani@unibas.it

Donatello Santoro

University of Basilicata
Potenza, Italy
donatello.santoro@unibas.it

Enzo Veltri

University of Basilicata
Potenza, Italy
enzo.veltri@unibas.it

Paolo Papotti

EURECOM
Biot, France
paolo.papotti@eurecom.fr

Abstract

Repository-level software repair is commonly approached by placing a language model in a free-form loop: the model explores the codebase, runs commands, edits files, interprets failures, and decides when the issue is resolved. This design works well for highly capable models, but it places many responsibilities on the model at once. When a smaller model fails, it is therefore difficult to tell whether it cannot produce the required patch, or whether it becomes lost while organizing the steps that lead to it. This paper studies an alternative design in which the repair process is decomposed into explicit stages. In particular, we separate finding the relevant code, constructing and using an executable signal of the bug, and generating the final edit. This perspective treats the harness not as a passive wrapper, but as part of the system being designed. We evaluate eight open-weight models from two families, ranging from 0.8B to 31B parameters, on a subset of SWE-bench Verified. Our results show that harness-managed control flow can substantially improve the effectiveness of the smallest models. For example, Gemma 4 E2B improves from 5.0% in the free-form loop to 27.4% in the decomposed repair loop and Gemma 4 E4B from 14.6% to 45.7%. Because the stages interact only through the context they build, they need not all be performed by the same model: handing localization and reproduction to stronger open-weight models, while the small model remains responsible only for the edit, raises the same two models further, to 39.8% and 53.7%. These findings suggest that small models can be useful for repository repair when responsibilities are divided across explicit stages that can be independently assigned to the component best suited to each.

CCS Concepts

• **Computing methodologies** → **Planning with abstraction and generalization**; • **Software and its engineering** → **Application specific development environments**.

^{*}Both authors contributed equally to this research.



This work is licensed under a Creative Commons Attribution 4.0 International License.
AgenticDev '26, Munich, Germany
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2985-0/2026/10
<https://doi.org/10.1145/3843282.3844442>

Keywords

small language models, LLM agents, agent harness design

ACM Reference Format:

Francesco Dente, Dario Satriani, Donatello Santoro, Enzo Veltri, and Paolo Papotti. 2026. Who Should Own the Loop? Harness Decomposition for Small-Model Repository Repair. In *Proceedings of the 1st International Workshop on Agentic AI for Next-Generation Software Development (AgenticDev '26)*, October 12–16, 2026, Munich, Germany. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/3843282.3844442>

1 Introduction

Resolving an issue in a real software repository (e.g. interpreting a bug report, locating the responsible code among thousands of files, and producing a patch that fixes the defect without breaking anything else) has become a flagship task for large language models. The dominant approach is agentic: a *harness*, the scaffolding program that manages the model's prompts, tools, and control flow, places the model in a loop [23, 26]. At each turn the model emits an action (typically a shell command or a file edit) which the harness executes in the repository environment, appending the output to the conversation; the cycle repeats until the model declares the issue fixed. In the prevailing *free-form* design the model owns every decision: which files to read (*localization*), whether to write a reproduction script (*reproduction*), which code to generate (*patching*), when to trust a candidate patch (*verification*) and submit it. All the structure of the trajectory comes from the model itself. Small language models (SLMs; here from 0.8B to 31B parameters) are attractive candidates for such systems: they are inexpensive to serve, fast, fine-tunable, and deployable on-premise [4]. Free-form harnesses, however, are designed around the capabilities of frontier models [18], and when placed in such systems, small models fail on repository-level tasks. This observation motivates the central question of our work: when an SLM fails inside an agentic harness on a real repository task, *where* exactly does it fail, and *which* parts of the harness must be removed or restructured (that is, moved from the model's responsibility into the harness's) for it to succeed?

To answer this question, we frame a successful agentic trajectory primarily as a *context-construction* process: whatever the agent does (open a file, run a command, execute a test) contributes to resolving the issue only through the output it appends to the model's context. The agentic loop thus exists to drive the context from its initial state (the issue text and access to a code repository) to a final state

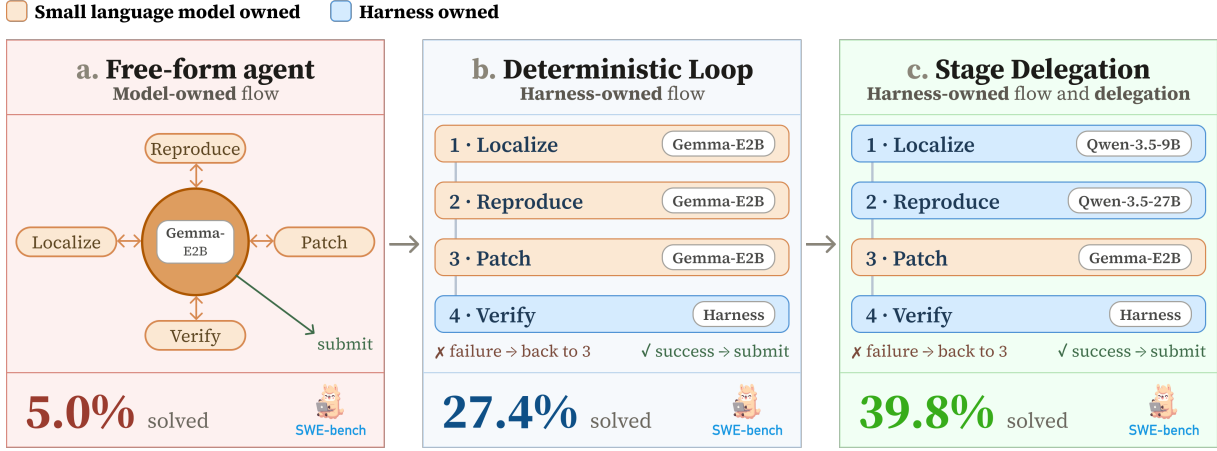


Figure 1: Who owns each stage, and what it buys (Gemma 4 E2B, SWE-bench Verified easy subset). (a) The model drives the whole loop through bash. (b) Control flow is moved into the harness, while the model still localizes, reproduces and patches. (c) Localization and reproduction are delegated to different models, using the small model only for generating the code patch.

that additionally holds: (i) the content of the file(s) that must be edited, and (ii) an executable signal reproducing the issue, together with its failing output. The stages of the resolution interact only through this shared context: localization determines which file contents enter it; execution feedback which run outputs; and patch generation consumes the assembled whole. Once this context is in place, the task essentially reduces to a *self-contained, function-level fix* that even a small model may be able to attempt. Under this lens, a minimal harness demands two core capabilities of the model: *localization* (find where to edit) and *execution feedback* (construct and exploit a reproduction signal); everything else reduces to patch generation. And because the stages communicate only through the context they build, they can be reassigned: the patch generator is indifferent to whether its context was assembled by the model itself wandering the repository, or supplied directly by the harness.

We make this framing operational by progressively factoring localization and execution feedback out of the model and into a deterministic harness. The result is a controlled spectrum of arms along an *agentic-freedom axis*: at one end, an existing unaltered minimal harness (Mini-SWE-Agent [26]) where the model drives the full process through free-form bash; at the other, a fully deterministic, harness-owned repair loop. We evaluate two open-weight families, the dense Qwen 3.5 [22] models (0.8B, 2B, 4B, 9B, and 27B parameters) and the Gemma 4 [7] models (E2B, E4B, and 31B), on the easy subset of SWE-bench Verified [10].

These experiments led to three empirical findings:

- (i) **Stage-decomposed pipelines are viable end-to-end.** With no oracle information at all, a single model that owns every stage (Figure 1 (b)) already outperforms the free-form agent (Figure 1 (a)) at the small end: when self-provided, Gemma 4 E2B resolves 27.4% of instances versus 5.0% for free-form, and Gemma 4 E4B resolves 45.7% versus 14.6%.
- (ii) **The factored stages are independently routable, and delegating them to stronger models makes small solvers usable.** Because the decomposed stages communicate only through the shared context, localization and reproduction can be handed to

stronger models (Figure 1 (c)) without retraining or altering the solver: with a Qwen 3.5 9B localizer and 27B reproducer, the pipeline lifts Qwen 3.5 0.8B from 0.7% to 14.8%, 2B from 16.1% to 23.0%, Gemma 4 E2B from 5.0% to 39.8%, and E4B from 14.6% to 53.7%.

(iii) **Structure, not information, is the binding constraint below 4B.** Handing a model the gold file locations and a validated reproduction signal is not enough: injected into the free-form loop, these artifacts leave the smallest models at their floor, while the deterministic harness delivering exactly the same artifacts makes them usable, lifting Gemma 4 E2B from under 7% to 42.4% resolved, with $\sim 3\times$ fewer tokens. The advantage narrows as models grow and reverses at and above 9B, where the free-form loop regains a small lead that the deterministic harness trades for a 5–10 $\times$ token reduction in budget. Code and dataset are available on [Github](#).

2 Related Work

Coding Harnesses. Coding harnesses vary widely in their degree of autonomy and in the complexity of the surrounding scaffold: Agentless [24] forgoes autonomy entirely and provides a fixed locate-generate-validate pipeline; Mini-SWE-Agent [26] defines a light bash-based environment contained in a minimal Python codebase; OpenHands [23] provides full CodeAct-style tooling in a multi-step reasoning environment; other pipelines add program analysis, as in AutoCodeRover’s structure aware search and fault localization [27], CodePlan’s dependency-driven edits [3], and RepairAgent’s autonomous tool-using repair loop [5]. We adopt Mini-SWE-Agent as our representative harness: its minimal scaffold makes it particularly suitable for small language models.

Task Decomposition. Close to our setting, ORACLE-SWE [12] decomposes the issue resolution pipeline into distinct signals (reproduction test, regression test, edit location, execution context, and API usage) and quantifies the upper-bound contribution of each by injecting ground-truth signals into the prompt of a minimal scaffold. Otter [1] targets the reproduction test signal specifically,

proving that reproduction tests improve the validation of candidate patches. Both operate at frontier scale, whereas we find that for SLMs the same injections leave the agent at its floor unless the harness structure changes with them. A complementary line *trains* small models to fit a chosen decomposition: SoRFT [14] fine-tunes on file-, function-, and line-localization and editing subtasks; SWE-Fixer [25] trains a retrieval-plus-editing pipeline that resolves issues in two inferences; Co-Patcher [21] trains component-specific 14B models for localization, generation, and validation; and SWE-Protégé [11] trains a 7B agent to selectively escalate to a frontier expert within the free-form loop; LocAgent [6] shows that a fine-tuned small localizer (32B) can approach the accuracy of frontier models in localization. These works modify the model to fit a chosen decomposition; in contrast, we hold the models frozen and vary the harness, extending our analysis down to 0.8B models to measure *which* stages must be factored out at which scale.

3 Methodology

Motivation. Our evaluation design follows from the central question stated in Section 1: when an SLM fails inside an agentic harness on a real repository task, where exactly does it fail, and which parts of the harness must be removed or restructured for it to succeed? An end-to-end evaluation cannot answer this question: it tells us *whether* a model-harness pair succeeds, not *why* it fails. We therefore treat the harness itself as the experimental axis: we decompose the issue-resolution loop into its core stages and compare harness configurations that reassign, capability by capability, responsibilities from the model to deterministic harness code, so that a change in resolved rate can be attributed to a specific capability.

3.1 Task and Dataset

Function-level benchmarks [2, 13, 28] measure standalone generation from natural-language descriptions and do not exercise the repository-level capabilities we study [8, 10]. We therefore evaluate on SWE-bench Verified [10], a human-validated benchmark of 500 real GitHub issues. Each instance pairs an issue report with a snapshot of the repository at the commit the issue was filed against; a candidate patch is graded by applying it in a containerized replica of the project environment and running the project’s own test suite, including held-out tests introduced by the reference fix. We restrict evaluation to the *easy* subset (<15-min difficulty annotation) because on the harder buckets small models floor near 0% and no harness effect can be measured. From these we exclude 14 instances whose *reference* patches fail in our evaluation environment, leaving $n=180$.

3.2 Harness Configurations

An agentic harness is the software layer that surrounds a language model and turns a stateless completion service into an agent: at minimum, a task-oriented *execution loop* governing the observe-act cycle, and a *tool registry* through which the model acts on its environment [15, 16]. For issue resolution, the execution loop can be broken into four stages: (i) *localization*, identifying, among thousands of files, the file(s) and code regions to modify; (ii) *issue reproduction*, turning the textual report into an executable signal that fails on the current code; (iii) *patching*, producing the concrete

source modification; and (iv) *verification*, running the reproduction signal against the candidate edit and revising accordingly. In a free-form agent the model carries out all four. The code edit is irreducibly generative, and a reproduction signal pays off only through the feedback loop that executes it, so we fold stages (ii) and (iv) into a single *execution feedback* capability. This leaves two capabilities that can be moved out of the model, *localization* and *execution feedback*, and we compare the three configurations that remove them cumulatively.

In the two deterministic configurations, following the same intuition as Agentless [24], the model never chooses the next action: the harness owns the control flow and invokes the model only at pre-defined generation points. However, in our deterministic settings, we treat the allocation of stages between the model and harness as the experimental variable and follow three shared design choices. First, a capability assigned to the harness is solved by an *oracle*: for localization, the harness shows the model the unmodified, base-commit contents of the files touched by the reference patch, while the reference patch content itself never enters the prompt; for execution feedback, it supplies a validated reproduction script derived from the issue. Second, the model expresses its edit as *search/replace* blocks that the harness locates in the shown files and converts into a unified diff by textual matching. Third, we enforce a strict *test-leakage safeguard*: the held-out evaluation tests are never shown, executed, or summarized during patch construction; all feedback the model receives derives from the issue text and code.

FREE-FORM. The unmodified Mini-SWE-Agent [26]. The model owns all four stages, interacting with a sandboxed checkout of the repository through free-form bash commands: one command per turn, each executed in a fresh subshell, for up to 50 steps. The submitted patch is the final `git diff` of the working tree. This configuration sits at the fully model-controlled end of the axis and serves as a comparison for both deterministic configurations.

+Loc. (localization solved). The harness provides the issue description and the base-commit contents of the oracle files, and the model must emit the complete fix in a single generation pass: nothing is executed and no execution feedback of any kind is returned. A bounded retry ($k=10$ attempts) re-prompts the model only when no patch materialized at all, the output contained no parseable edit block, or no search text matched the shown files, and the first patch that applies is submitted immediately: retries repair format failures, leaving the zero-feedback nature of the configuration unchanged. This measures patch generation once localization is solved.

+Loc. +Exec. (localization and execution feedback solved). Adds the harness-owned execution feedback channel on top of +Loc. The feedback signal is a reproduction trigger: a minimal code snippet, written from the issue and the localized file contents, that raises while the defect is present. The trigger is authored by a frontier model (GPT-5.4 [17]) and replayed for every solver and run. The repair loop runs for up to $k=10$ rounds: the model’s edits are applied and the trigger is re-executed; a clean run submits the patch, while a failing run feeds back the execution output together with the numbered region of the shown file indicated by the last in-file traceback frame. The verdict is always computed by the harness through re-execution, never by the model’s own judgment. The oracle files are shown once, feedback outputs are clipped to a fixed

Table 1: Resolved rate (%), mean $\pm$ std over 3 runs) and average tokens per task. Every column receives the same gold information and differs only in the harness that delivers it: the FREE-FORM columns inject it into the unmodified agent loop (gold file paths appended to the task and the reproduction script placed in the workspace), while the deterministic columns delegate the stages to the harness.

Model	FREE-FORM		Deterministic	
	+LOC.	+LOC. +EXEC.	+LOC.	+LOC. +EXEC.
<i>Qwen 3.5</i>				
0.8B	0.7 $\pm$ 0.3 (63k)	0.7 $\pm$ 0.3 (64k)	11.7 $\pm$ 0.6 (105k)	15.7$\pm$0.8 (96k)
2B	16.3 $\pm$ 0.6 (593k)	17.6 $\pm$ 0.8 (621k)	30.4$\pm$0.3 (43k)	21.1 $\pm$ 0.0 (73k)
4B	59.1 $\pm$ 1.4 (502k)	59.3$\pm$3.4 (446k)	51.9 $\pm$ 0.6 (24k)	58.5 $\pm$ 0.8 (46k)
9B	70.2$\pm$2.5 (459k)	69.3 $\pm$ 0.6 (353k)	56.9 $\pm$ 1.2 (21k)	65.9 $\pm$ 0.8 (40k)
27B	80.6$\pm$1.1 (326k)	79.1 $\pm$ 0.8 (266k)	72.6 $\pm$ 1.2 (15k)	74.3 $\pm$ 0.8 (31k)
<i>Gemma 4</i>				
E2B	6.7 $\pm$ 3.1 (111k)	6.1 $\pm$ 1.5 (126k)	37.8 $\pm$ 1.1 (23k)	42.4$\pm$0.3 (43k)
E4B	17.8 $\pm$ 0.0 (44k)	21.7 $\pm$ 0.6 (73k)	53.1 $\pm$ 1.4 (13k)	58.1$\pm$0.6 (41k)
31B	79.6$\pm$0.3 (169k)	78.3 $\pm$ 4.4 (160k)	78.1 $\pm$ 1.7 (16k)	75.7 $\pm$ 1.4 (34k)

size, and if the budget is exhausted the last patch that applied is submitted.

End-to-end configurations. The two deterministic configurations above are oracle-assisted: the factored capabilities are solved by construction, so their results are upper bounds. To measure the same pipeline without any oracle input, we also evaluate both configurations end-to-end, replacing each oracle with a model. Localization becomes a two-stage generation procedure: the model first selects candidate directories from the repository tree, then up to five files shown as code skeletons, followed by a content-based reranking pass. The reproduction trigger is authored per task from the issue and the localized file contents as in +LOC. +EXEC.. We evaluate two assignments of these stages: *self-provided*, in which the solver model performs both itself, and *delegated*, in which secondary models are assigned to these stages and the solver only writes patches.

4 Results

We organize the results around two research questions: **RQ1**: given the same oracle information (where to edit, and a validated reproduction signal), does the harness structure that delivers it determine whether SLMs can resolve the task? **RQ2**: without oracle information, is the decomposed pipeline still viable end-to-end, and can delegating its stages to stronger models recover the oracle-assisted gains? Table 1 supports RQ1 and Table 2 supports RQ2.

Metrics and evaluation. We report the standard SWE-bench resolved rate: the percentage of instances whose submitted patch passes the project’s held-out test suite. We run each configuration three times and report the mean $\pm$ standard deviation across runs. Each run uses the recommended sampling parameters: Qwen 3.5 at temperature 0.6, top- p 0.95, top- k 20; Gemma 4 at temperature 1.0, top- p 0.95, top- k 64.

4.1 RQ1: Structure vs. Information

Every column of Table 1 supplies the model with the same two artifacts (the location of the edit and a validated reproduction signal) and differs only in the harness that delivers them. The information is thus held constant while the harness structure varies: any difference in resolved rate is therefore attributable to structure alone.

Small models (Qwen 3.5 0.8B–4B, Gemma 4 E2B–E4B). Injected into the unmodified free-form loop, the oracle information is nearly inert: Qwen 3.5 0.8B resolves 0.7% whether given the file paths alone or the paths together with the reproduction script, and Gemma 4 E2B remains below 7% in both conditions. Delivered by the deterministic harness, the identical artifacts yield 15.7% for Qwen 3.5 0.8B and 42.4% for Gemma 4 E2B. The same pattern holds one size up: 21.1% vs. 17.6% for Qwen 3.5 2B, and 58.1% vs. 21.7% for Gemma 4 E4B. On average the deterministic harness is also several times cheaper (roughly 6 $\times$ fewer tokens), the one exception being Qwen 3.5 0.8B, where the free-form agent floors and halts so quickly that it spends less. At Qwen 3.5 4B the two structures reach parity (58.5% vs. 59.3%), with the deterministic harness still spending roughly a tenth of the tokens.

A qualitative case. One instance illustrates the mechanism directly. The task asks for SymPy’s Python code printer to support indexed expressions, and the issue report states the fix outright. The model need only find the right file and write that method into it. The 0.8B model under the free-form agent locates the correct file and states the right plan, but it repeatedly produces output that the harness cannot execute as a valid command, never performing the edit it had already described correctly. The deterministic configuration removes this obstacle: instead of operating a shell, the model is shown the file and asked for the change directly.

Mid-scale ($\geq 9B$). Only at this scale does the free-form structure yield an advantage. With both stages provided (+LOC. +EXEC.), it improves the resolved rate by 2.5–5 points, at 5–9 $\times$ more tokens. This sharpens the contrast with frontier-scale signal-injection studies. ORACLE-SWE [12] reports double-digit gains from injecting via prompt a reproduction test into a frontier-model operating a FREE-FORM agent; below 9B, the same injection leaves the agent at its floor. Signal-injection findings do not transfer down the size axis: for SLMs the binding constraint is structural, not informational.

Takeaways for RQ1. Harness-managed control flow makes small models usable below 4B. The advantage reverses above 9B, where free-form control regains a 2.5–5 point lead at 5–9 $\times$ the tokens.

4.2 RQ2: End-to-End Viability and Stage Delegation

Table 1 reports upper bounds: localization and trigger authoring were solved by construction. Table 2 removes that assistance and runs the identical repair loop against artifacts produced by real components, compared against the stock FREE-FORM agent.

Self-provided. The solver performs its own localization and authors its own reproduction trigger (Section 3). On Gemma 4 the pipeline dominates free-form outright at the small end: E2B leaps from 5.0% to 27.4% and E4B from 14.6% to 45.7%. Above 2B on Qwen 3.5, self-localization misses cap the pipeline below the free-form agent, but at 5–14 $\times$ fewer tokens.

Table 2: End-to-end configurations: resolved rate (% , mean $\pm$ std over 3 runs) and average tokens per task. FREE-FORM is the stock agent. In the deterministic *self-provided* columns the solver performs its own localization and authors its own reproduction trigger. In the *delegated* columns these stages are handed to separate Qwen 3.5 models (*Loc.* localizer, *Exec.* reproducer) and the solver only writes patches. Token counts in the delegated columns include the delegated executions costs.

Model	FREE-FORM	Deterministic self-provided		Deterministic delegated			
		+LOC.	+LOC. +EXEC.	Loc.	Exec.	+LOC.	+LOC. +EXEC.
Qwen 3.5							
0.8B	0.7±0.6 (52k)	3.1±0.3 (117k)	3.7±0.6 (81k)	9B	27B	11.1±0.6 (133k)	14.8±0.6 (131k)
2B	16.1±1.0 (617k)	13.5±1.6 (58k)	8.9±1.5 (79k)	9B	27B	28.5±0.3 (71k)	23.0±1.3 (115k)
4B	57.0±1.3 (516k)	44.3±0.3 (37k)	41.9±2.3 (64k)	9B	27B	48.1±1.4 (52k)	51.5±0.6 (87k)
9B	64.8±1.3 (497k)	52.8±1.0 (50k)	56.5±0.8 (78k)	27B	27B	54.1±0.6 (52k)	61.1±1.0 (80k)
27B	78.9±2.0 (371k)	68.5±1.3 (45k)	70.4±1.8 (77k)	27B	27B	68.5±1.3 (45k)	70.4±1.8 (77k)
Gemma 4							
E2B	5.0±1.0 (149k)	29.4±1.0 (45k)	27.4±1.4 (62k)	9B	27B	36.1±1.5 (51k)	39.8±0.3 (85k)
E4B	14.6±2.7 (79k)	44.6±1.4 (31k)	45.7±1.8 (64k)	9B	27B	48.7±0.8 (42k)	53.7±2.6 (78k)
31B	74.6±1.7 (244k)	68.9±2.0 (30k)	66.9±1.4 (62k)	27B	27B	72.8±2.2 (46k)	71.1±0.6 (75k)

Delegated. Because the stages interact solely through the context they progressively build, each can be assigned to a different executor without changing the patcher. As a first form of routing, we delegate localization to Qwen 3.5 9B for solvers below 9B and to Qwen 3.5 27B for the rest, and always delegate the reproduction trigger (the most delicate stage) to Qwen 3.5 27B. This recovers most of what self-provision loses. On Qwen 3.5, the 0.8B solver rises from 3.7% to 14.8% and the 2B from 8.9% to 23.0%, and the 9B from 56.5% to 61.1%. On Gemma 4 the same substitution lifts E2B from 27.4% to 39.8%, E4B from 45.7% to 53.7%, and 31B from 66.9% to 71.1%. Every delegated solver lands within ~ 5 points of its fully-assisted +LOC. +EXEC. cell in Table 1 (except Qwen 3.5 4B): a primitive open-weight component substitutes for the oracle at small cost.

Against the free-form agent. Below 4B the delegated pipeline is the only usable option for Qwen 3.5. At 4B and 9B it concedes 3–6 points, and at 27B 8.5 points, for ~ 5 – $6\times$ fewer tokens. On Gemma 4 the delegated pipeline dominates free-form below 9B: 39.8% vs. 5.0% for E2B at roughly half the tokens, and 53.7% vs. 14.6% for E4B. These numbers are a floor: the localizer is two prompts over file paths and code skeletons, with no retrieval tooling, and no per-stage budget tuning. Strengthening localization [6], optimizing per-stage model and budget assignment, and specializing the execution-feedback component are left to future work.

Takeaways for RQ2. The decomposition survives the removal of every oracle: a single small model owning all stages already beats the free-form agent below 9B on Gemma 4. Delegating the factored stages to a stronger open-weight model brings every solver within ~ 5 points of its oracle-assisted ceiling.

5 Conclusion and Future Directions

We studied repository-level issue resolution as a joint model-systems problem: which responsibilities should remain with the language model, and which should be encoded in the harness? Across the configurations considered, no single allocation dominates at every scale. For the smallest tested models, directly presenting localized code and managing execution feedback in the harness often makes patch

generation substantially more effective than a bash-only free-form loop. Larger models recover some of the flexibility and accuracy of free-form control, generally at higher token use. The end-to-end experiments further show that the patch generator need not also localize and reproduce the issue: delegating those stages to another component recovers much of the oracle-assisted performance. The harness should therefore be treated as part of the system being designed, rather than as a fixed wrapper around one model.

Our current harness uses the same stage graph for every issue. A natural next step is to compile each issue into a typed directed acyclic graph of repair operators, for example, repository search, file and region localization, dependency inspection, trigger synthesis, patch generation, validation and contract checking [9]. Each operator would declare an input/output contract, a validator, and estimates of its cost and reliability. A *physical planner* could then assign each operator to a deterministic tool or model, begin with the cheapest viable implementation, and escalate only failed or low-confidence nodes. Independent nodes could run in parallel, while validated artifacts could be cached across patch attempts. This extends existing repository-level planning [3] and is analogous to LLM data systems that choose among alternative operator implementations and execution plans to trade result quality against cost [19, 20]. The optimization objective should be expected end-to-end resolution under a compute or latency budget.

Limitations. The present evidence is restricted to the easy slice of SWE-bench Verified ($n=180$) where reference patches are largely single-file, two model families, one primary free-form harness, and particular implementations of localization and execution feedback. The observed scale transition should therefore be read as a property of these model-harness pairs. Future evaluations should include harder and multi-file issues and additional model-harness pairs.

Acknowledgment

This work was partially supported by the AI4SWEng project (grant agreement No. 101189909), and by the 3IA Côte d’Azur Investments in the IA-cluster project (ANR-23-IACL-0001).

References

- [1] Toufique Ahmed, Jatin Ganhotra, Rangeet Pan, Avraham Shinnar, Saurabh Sinha, and Martin Hirzel. 2025. Other: Generating Tests from Issues to Validate SWE Patches. In *Forty-second International Conference on Machine Learning*. <https://openreview.net/forum?id=b0Ys6JOZu>
- [2] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program Synthesis with Large Language Models. *arXiv:2108.07732* [cs.PL] <https://arxiv.org/abs/2108.07732>
- [3] Ramakrishna Bairi, Atharv Sonwane, Aditya Kanade, Vageesh D. C., Arun Iyer, Suresh Parthasarathy, Sriram Rajamani, B. Ashok, and Shashank Shet. 2024. CodePlan: Repository-Level Coding using LLMs and Planning. *Proc. ACM Softw. Eng.* 1, FSE, Article 31 (July 2024), 24 pages. [doi:10.1145/3643757](https://doi.org/10.1145/3643757)
- [4] Peter Belcak, Greg Heinrich, Shizhe Diao, Yonggan Fu, Xin Dong, Saurav Muralidharan, Yingyan Celine Lin, and Pavlo Molchanov. 2025. Small Language Models are the Future of Agentic AI. *arXiv preprint arXiv:2506.02153* (2025). [doi:10.48550/arXiv.2506.02153](https://doi.org/10.48550/arXiv.2506.02153)
- [5] Islem Bouzenia, Premkumar Devanbu, and Michael Pradel. 2025. RepairAgent: An Autonomous, LLM-Based Agent for Program Repair. In *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering* (Ottawa, Ontario, Canada) (ICSE '25). IEEE Press, 2188–2200. [doi:10.1109/ICSE55347.2025.00157](https://doi.org/10.1109/ICSE55347.2025.00157)
- [6] Zhaoling Chen, Robert Tang, Gangda Deng, Fang Wu, Jialong Wu, Zhiwei Jiang, Viktor Prasanna, Arman Cohan, and Xingyao Wang. 2025. LocAgent: Graph-Guided LLM Agents for Code Localization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (Eds.). Association for Computational Linguistics, Vienna, Austria, 8697–8727. [doi:10.18653/v1/2025.acl-long.426](https://doi.org/10.18653/v1/2025.acl-long.426)
- [7] Google DeepMind. 2026. Gemma 4: Byte for byte, the most capable open models. <https://blog.google/innovation-and-ai/technology/developers-tools/gemma-4/>
- [8] Francesco Dente, Dario Satriani, and Paolo Papotti. 2026. Constraint Decay: The Fragility of LLM Agents in Backend Code Generation. *arXiv preprint arXiv:2605.06445* (2026). [doi:10.48550/arXiv.2605.06445](https://doi.org/10.48550/arXiv.2605.06445)
- [9] Francesco Dente, Dario Satriani, Donatello Santoro, Enzo Veltri, and Paolo Papotti. 2026. Do LLMs Understand Backend Data-Layer Semantics? A Controlled Study of Cross-Endpoint Issues. In *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing*.
- [10] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-world Github Issues?. In *The Twelfth International Conference on Learning Representations*. <https://openreview.net/forum?id=VTF8yNQm66>
- [11] Patrick Tser Jern Kon, Archana Pradeep, Ang Chen, Alexander P Ellis, Warren Hunt, Zijian Wang, John Yang, and Samuel Thompson. 2026. SWE-Protégé: Learning to Selectively Collaborate With an Expert Unlocks Small Language Models as Software Engineering Agents. *arXiv preprint arXiv:2602.22124* (2026). [doi:10.48550/arXiv.2602.22124](https://doi.org/10.48550/arXiv.2602.22124)
- [12] Kenan Li, Qirui Jin, Liao Zhu, Xiaosong Huang, Yijia Wu, Yikai Zhang, Xin Zhang, Zijian Jin, Yufan Huang, Elsie Nallipogu, Chaoyun Zhang, Yu Kang, Saravan Rajmohan, Qingwei Lin, Wenke Lee, and Dongmei Zhang. 2026. ORACLE-SWE: Quantifying the Contribution of Oracle Information Signals on SWE Agents. *arXiv:2604.07789* [cs.MA] <https://arxiv.org/abs/2604.07789>
- [13] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. *arXiv:2305.01210* [cs.SE] <https://arxiv.org/abs/2305.01210>
- [14] Zexiong Ma, Chao Peng, Pengfei Gao, Xiangxin Meng, Yanzen Zou, and Bing Xie. 2025. SoRFT: Issue Resolving with Subtask-oriented Reinforced Fine-Tuning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (Eds.). Association for Computational Linguistics, Vienna, Austria, 11427–11441. [doi:10.18653/v1/2025.acl-long.559](https://doi.org/10.18653/v1/2025.acl-long.559)
- [15] Qianyu Meng, Yanan Wang, Liyi Chen, Yihang Li, Wei Wu, Wenyuan Jiang, Qimeng Wang, Chengqiang Lu, Yan Gao, Yi Wu, and Yao Hu. 2026. Agent Harness for Large Language Model Agents: A Survey. *Preprints* (April 2026). [doi:10.20944/preprints202604.0428.v3](https://doi.org/10.20944/preprints202604.0428.v3)
- [16] Xuying Ning, Katherine Tieu, Dongqi Fu, Tianxin Wei, Zihao Li, Yuanchen Bei, Jiaru Zou, Mengting Ai, Zhining Liu, Ting-Wei Li, Lingjie Chen, Yanjun Zhao, Ke Yang, Bingxuan Li, Cheng Qian, Gaotang Li, Xiao Lin, Zhichen Zeng, Ruizhong Qiu, Sirui Chen, Yifan Sun, Xiyuan Yang, Ruida Wang, Rui Pan, Chenyuan Yang, Dylan Zhang, Liri Fang, Zikun Cui, Yang Cao, Pan Chen, Dorothy Sun, Ren Chen, Mahesh Srinivasan, Nipun Mathur, Yinglong Xia, Hong Li, Hong Yan, Pan Lu, Lingming Zhang, Tong Zhang, Hanghang Tong, and Jingrui He. 2026. Code as Agent Harness. *arXiv:2605.18747* [cs.CL] <https://arxiv.org/abs/2605.18747>
- [17] OpenAI. 2026. Introducing GPT-5.4. <https://openai.com/index/introducing-gpt-5-4/>
- [18] Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. 2025. Training Software Engineering Agents and Verifiers with SWE-Gym. In *Forty-second International Conference on Machine Learning*. <https://openreview.net/forum?id=Cq1BNvHx74>
- [19] Liana Patel, Siddharth Jha, Melissa Z. Pan, Harshit Gupta, Parth Asawa, Carlos Guestrin, and Matei Zaharia. 2025. Semantic Operators and Their Optimization: Towards AI-Based Data Analytics with Accuracy Guarantees. *Proc. VLDB Endow.* 18, 11 (2025), 4171–4184. [doi:10.14778/3749646.3749685](https://doi.org/10.14778/3749646.3749685)
- [20] Dario Satriani, Enzo Veltri, Donatello Santoro, Sara Rosato, Simone Varriale, and Paolo Papotti. 2025. Logical and Physical Optimizations for SQL Query Execution over Large Language Models. *Proc. ACM Manag. Data* 3, 3 (2025), 181:1–181:28. [doi:10.1145/3725411](https://doi.org/10.1145/3725411)
- [21] Yuheng Tang, Hongwei Li, Kaijie Zhu, Michael Yang, Yangruibo Ding, and Wenbo Guo. 2025. Co-PatchR: Collaborative Software Patching with Component-specific Small Reasoning Models. In *Advances in Neural Information Processing Systems*, D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (Eds.), Vol. 38. Curran Associates, Inc., 42041–42069. https://proceedings.neurips.cc/paper_files/paper/2025/file/3be60cd94716ca358cc8fbc842c3716-Paper-Conference.pdf
- [22] Qwen Team. 2026. Qwen3.5: Accelerating Productivity with Native Multimodal Agents. <https://qwen.ai/blog?id=qwen3.5>
- [23] 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. 2025. OpenHands: An Open Platform for AI Software Developers as Generalist Agents. In *International Conference on Learning Representations*, Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (Eds.), Vol. 2025. 65882–65919. https://proceedings.iclr.cc/paper_files/paper/2025/file/a4b6ad6b48850c0c331d1259fc66a69c-Paper-Conference.pdf
- [24] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. 2025. Demystifying LLM-Based Software Engineering Agents. *Proc. ACM Softw. Eng.* 2, FSE, Article FSE037 (June 2025), 24 pages. [doi:10.1145/3715754](https://doi.org/10.1145/3715754)
- [25] Chengxing Xie, Bowen Li, Chang Gao, He Du, Wai Lam, Difan Zou, and Kai Chen. 2025. SWE-Fixer: Training Open-Source LLMs for Effective and Efficient GitHub Issue Resolution. In *Findings of the Association for Computational Linguistics: ACL 2025*, Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (Eds.). Association for Computational Linguistics, Vienna, Austria, 1123–1139. [doi:10.18653/v1/2025.findings-acl.62](https://doi.org/10.18653/v1/2025.findings-acl.62)
- [26] John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (Eds.), Vol. 37. Curran Associates, Inc., 50528–50652. [doi:10.52202/079017-1601](https://doi.org/10.52202/079017-1601)
- [27] Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. 2024. AutoCodeRover: Autonomous Program Improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Vienna, Austria) (ISSTA 2024). Association for Computing Machinery, New York, NY, USA, 1592–1604. [doi:10.1145/3650212.3680384](https://doi.org/10.1145/3650212.3680384)
- [28] Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widayarsi, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen GONG, James Hoang, Armel Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, Binyuan Hui, Niklas Muennighoff, David Lo, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. 2025. BigCodeBench: Benchmarking Code Generation with Diverse Function Calls and Complex Instructions. In *International Conference on Learning Representations*, Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (Eds.), Vol. 2025. 66602–66656. https://proceedings.iclr.cc/paper_files/paper/2025/file/a6a90bcc2aa470c3871b2d39a67d26e8-Paper-Conference.pdf

Received 2026-07-23; accepted 2026-08-20