Better Harnesses, Smaller Models: Building 90% Cheaper Agents via Automated Harness Adaptation

Chenyang Yang, Xinran Zhao, Tongshuang Wu, and Christian Kästner
Carnegie Mellon University

***Abstract*—Frontier LLM agents are automating many business tasks, but their high inference cost makes large-scale deployment unsustainable. Small language models (SLMs) offer a cheaper alternative, yet they typically fall short when swapped into a harness designed for a frontier LLM. We show that for many routine business tasks, SLM agents can match LLM performance at 90% lower cost, when paired with an adapted harness that can be automatically discovered by a meta agent. The key insight is that much of the task difficulty is shared across instances and can be lifted from the model into the harness via tailored instructions, tools, and orchestration loops. To study this systematically, we create a framework that maps agent failure modes to harness adaptation strategies, and build a harness optimizer that automatically discovers effective adaptations from failure trajectories. Across seven business-oriented agentic tasks and three SLM families, we found optimized harnesses significantly improve performance on 16 of 21 task-SLM pairs, with seven pairs closing the SLM-LLM performance gap and the best SLM agent recovering 89.7% of LLM performance at 4% of the cost. [1] Our analysis further shows that adaptation works best for tasks with more repetitive workflows and for SLMs with sufficient base capabilities. Together, these results suggest that harness adaptation can expand the practical deployment range of SLM agents in routine business tasks.**

***Index Terms*—agents; small language models; harness adaptation; cost-efficient AI deployment**

I. Introduction

Large language models (LLMs) power many agentic systems that are increasingly deployed in real-world business workflows [1], [2]. While frontier models deliver impressive performance, they incur substantial inference cost, high latency, and data-privacy concerns – businesses have poured billions of dollars into incorporating agents into their workflows [3], and that continuous spending can quickly become unsustainable [4].

Small language models (SLMs)[2] have emerged as an alternative for replacing LLMs on agentic tasks, with their lower cost, better latency, and less privacy concerns [8]. While SLMs fall short of LLMs on general-purpose agentic coding tasks [9], many business-relevant tasks do not require open-ended creativity, but only reliable execution of routine workflows within a constrained environment [8] – this makes SLMs an attractive option for replacing LLMs in these contexts.

As a concrete example, suppose we want to deploy an agent to collect, review, and communicate various budget requests in a company, a routine business function that many try to automate. We can deploy an agent with a frontier LLM that will perform very well (97.3% accuracy with `gemini-3.1-pro` on curated evaluation data), but is also expensive ($0.22 per query) to deploy in production. Alternatively, we can deploy the same agent with an SLM. The agent, however, performs worse (75.0% with `gemma-4-26b-a4b`) and cannot make a reliable replacement (Figure 1).

In this paper, we demonstrate that, for many routine agentic tasks in business settings, **we can build specialized SLM agents that are 90% cheaper yet with on-par performance with LLM agents**. This is achieved by pairing SLMs with *well-adapted harnesses* that can be discovered automatically with a meta agent. This is because, for these tasks, a lot of task difficulty is shared across instances and can be lifted from the model to the harness, through (automatically) tailored instructions, tools, and orchestration loops. In our budget-approval example, while SLMs may struggle to create complex plans or select appropriate tools on their own, a well-designed harness can scaffold an SLM with a clear plan skeleton, a reduced tool set, and enforced constraints via hooks, improving SLM performance to 98.3% (Figure 1, right).

To build effective specialized agents with SLMs, we first need to understand what makes a task hard and how that difficulty can be absorbed into the harness. To this end, we synthesize a list of agent failure modes and harness adaptation strategies from existing literature (Section II), covering common failure modes such as `tool-use` and `knowledge` failures, and adaptation strategies on contexts, tools, and orchestration loops. This gives us a framework for diagnosing why an SLM fails on a task and identifying and explaining which harness adaptations are likely to help: For example, when we observe an SLM struggle with choosing tools from a large set (`tool-use` failures), we can adapt the harness by creating high-level tools and filtering unused ones.

While harness adaptations can be effective for many tasks, it is unclear when harness adaptation is sufficient to close the SLM-LLM gap, and which task or model conditions make adaptation more effective. To systematically understand if, when, and how we can build (cost-)effective SLM agents with harness adaptation, we conduct an empirical study across seven curated business tasks and three SLMs from different model families (Section IV). Rather than engineering a harness by hand for each task–SLM pair, we develop a harness optimizer that automatically discovers effective adaptations: We design the harness optimizer as a meta-agent that systematically diag-

[1] Code available at https://github.com/malusamayo/migration-analysis.

[2] We use SLMs operationally for cheaper, smaller, open-weight models that can be deployed locally. Our experiments use models with 3B to 8B (8B to 30B) active (total) parameters (e.g., `qwen3-30b-a3b` [5]). In comparison, frontier open-sourced models have more than 1T model parameters [6], [7].

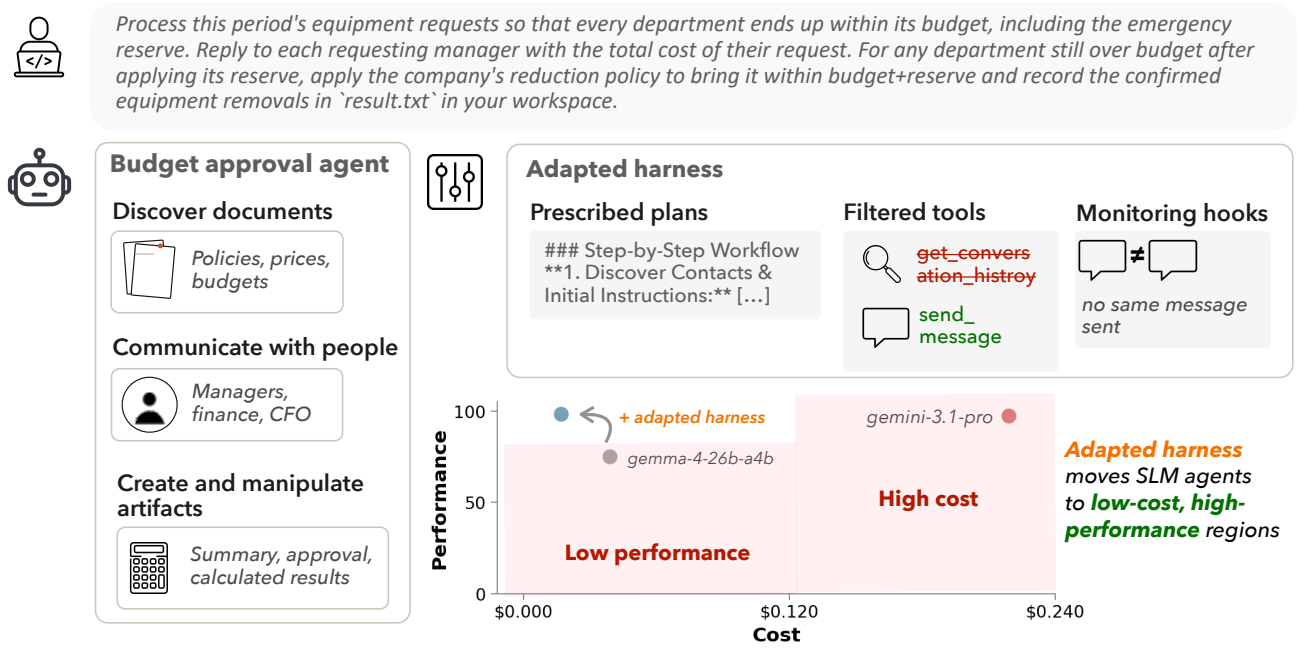

Fig. 1: Small language models often perform poorly when naively swapped into an agent harness designed for frontier LLMs. An adapted harness can provide detailed plans to guide workflow execution, a reduced tool set to avoid invocation mistakes, and monitoring hooks to avoid loops. These adaptations make an SLM on par with an LLM with 8% of the cost.

noses agent failures and proposes targeted harness adaptations (Section III), building on recent advances of automated harness optimization [10], [11]. This allows us to systematically study the effectiveness of SLM harness adaptation without laborious manual engineering. From the study, we find that,

- **Harness adaptation substantially improves the cost–performance tradeoff of SLM agents.** Optimized harnesses significantly improve performance on 16 out of 21 SLM-task pairs, with seven pairs closing the SLM-LLM performance gap. The best adapted SLM can recover 89% of LLM-agent performance with 96% cost reduction.
- **Harness adaptation is grounded in common failure modes.** Successful adaptations most often address `instruction-following` (81%) and `knowledge` (81%) failures. The dominant strategies include adding contexts (86%), creating new tools (43%), and managing tools (29%).
- **Harness adaptation is more effective for less diverse tasks.** Optimized harnesses work best when task instances share a stable workflow (+21.1% accuracy when we go from the most diverse task setting to the least diverse setting).
- **Harness adaptation is more effective for SLMs with stronger base capabilities.** More capable SLMs benefit more (+48.8% vs. +15.5%) from harness adaptation, suggesting that harnesses can absorb substantial task difficulty but cannot fully replace missing core model capabilities.

In summary, this paper contributes:

- A task suite of 7 diverse agentic tasks designed for studying agent deployment on repetitive business tasks.
- Empirical findings from experiments on 7 tasks × 3 SLMs, showing that harness adaptations recover substantial LLM performance at a fraction of the cost, and that we can characterize when and why adaptations work.
- A framework mapping failure modes to harness adaptation strategies, explaining how the adapted harnesses are effective.
- A harness optimizer that automatically searches for effective harness adaptations for SLM agents.

II. Characterizing Language Model Agent: Failures and Adaptations

To identify promising harness adaptations, we first need to understand why agents fail and what harness adaptations are available. In this section, we first establish a framework to characterize agent failures and map them to relevant harness adaptation strategies.

A. Agent failure modes

Failures in agentic task outcomes come from one or multiple *local* failures in the trajectory. For example, a budget-approval agent that approves invalid requests may have selected the wrong tool at one step, violated a policy instruction at another, relied on an incorrect assumption, lost track of an earlier budget limit near the end, or created an infeasible plan in the beginning.

We organize such local failure modes by the *capabilities* they implicate, such as `tool-use`, `knowledge`, `instruction-following`, `long-context`, and `planning`. We derive these capabilities by surveying recent model cards [12]–[14] and the agentic benchmarks they report (e.g., [15]–[23]). Indexing failures by capability allows us to link local failures to the underlying model abilities that may have broken down, and to the adaptations likely to address them. We next describe each capability-indexed failure mode in detail (Figure 2, left):

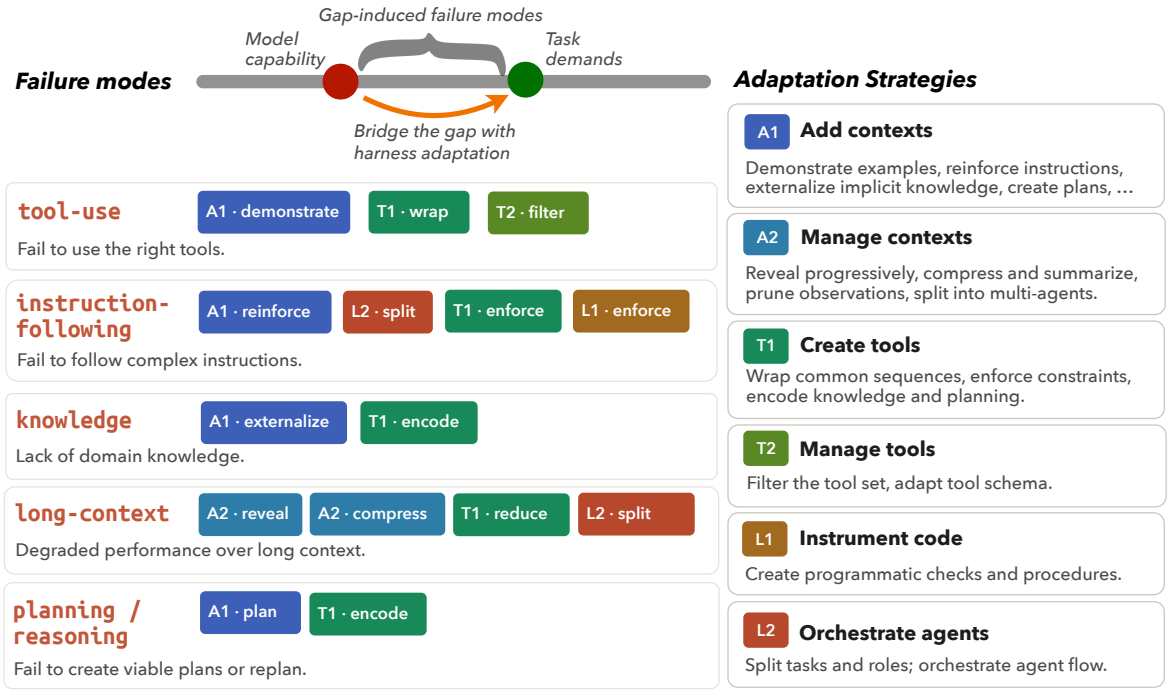

Fig. 2: **Mapping Agent Failure Modes to Harness Adaptations.** When model capabilities fall short of task demands, this mismatch manifests as agent failures in trajectories. These gaps can be bridged with harness adaptation, which we group into three categories: adapting contexts, tools, and agent loops.

a) **tool-use** *failures:* Tool-use failures occur when the model selects the wrong tool, produces a malformed tool call, or uses a tool incorrectly. They are especially common when the tool set is large, tool schemas are complex, or the task requires composing multiple tools.

b) **instruction-following** *failures:* Instruction-following failures occur when the model violates provided system or user instructions, which can include task requirements, output formatting rules, workflow guidelines, etc. They are especially common when instructions are long and complex.

c) **knowledge** *failures:* Knowledge failures occur when the model lacks the domain, environment, or commonsense assumptions needed to complete an under-specified task. Because real-world instructions are often under-specified [24], agents are expected to fill in missing details from pretraining knowledge or from interacting with the environment. When this knowledge is missing, agents may call the wrong tools, misinterpret task constraints, or follow sub-optimal workflows.

d) **long-context** *failures:* Long-context failures occur when the model fails to retrieve and use information from a long trajectory. Failures may appear as forgetting earlier instructions, repeating already-failed actions, overlooking relevant observations, or being distracted by irrelevant context.

e) **planning / reasoning** *failures:* Planning and reasoning failures occur when the model cannot decompose a high-level goal into appropriate sub-goals, choose an effective next action, or revise its plan after new observations. These failures are central to agentic tasks because the model is expected to repeatedly decide what to do next based on partial and evolving information. They may appear as premature answers, inefficient exploration, failure to backtrack, or local actions that do not contribute to the overall goal.

Comparison with existing failure analysis. Prior work has also proposed empirical taxonomies for understanding agent failures. Cemri et al. [25] introduce MAST, a taxonomy of multi-agent failures covering system design issues, inter-agent misalignment, and task verification. Zhu et al. [26] propose AgentErrorTaxonomy, which attributes single-agent failures to memory, reflection, planning, action, and system-level modules. In contrast, we index failures by model *capabilities*, making it easier to connect observed failures to capability gaps and targeted harness adaptations.

B. Agent harness adaptations

To address observed agent failures, offloading the task demand from the model to the harness can be an effective strategy. For example, a **knowledge** failure may be mitigated by *adding contexts* of domain instructions, a **tool-use** failure by transforming the available tools, a **long-context** failure by hiding irrelevant observations, and an **instruction-following** failure by enforcing constraints outside of the model outputs.

We present an overview of harness adaptation strategies that recur in existing literature and popular harnesses (e.g., [27], [28]). We group these strategies by the harness component they modify: Context adaptations change what information is presented to the model, tool adaptations change the actions available to the model, and agent-loop adaptations change the control logic around model inference (Figure 2, right).

a) Context adaptations: Context adaptations shape the context a model acts on, including system prompts, user

prompts, tool descriptions, memory, and other observable information like files and databases in the environment.

Adding contexts is the most straightforward and commonly used adaptation strategy: It helps with knowledge gaps by externalizing implicit knowledge into instructions, with planning gaps by providing explicit plans, with tool-use gaps by providing more detailed tool descriptions and examples, and with instruction-following gaps by reinforcing key constraints. It has been well studied in the literature as prompt engineering and context engineering [29], and widely used by practitioners to steer agent behaviors (e.g., AGENTS.md [30]). Adding contexts, however, also incurs long-context burden and more complex instruction-following. **Managing contexts** properly is thus essential to prevent overwhelming SLMs. Common strategies include revealing contexts progressively [31], offloading contexts into external storage [32], compressing contexts with summaries [33], pruning irrelevant observations [34], and separating contexts into distinct components like multi agents [35].

b) Tool adaptations: Tool adaptations reshape the action space an agent operates in – they are commonly implemented as (1) MCP servers [36], (2) command-line tools, or (3) executable scripts (e.g., as in agent skills [31]). **Creating tools** is a powerful strategy to mitigate tool-use gaps: By wrapping recurring complex tool use patterns and sequences into simpler interfaces, we can support easier and more stable tool use [37]. It also helps with knowledge and planning gaps by encoding domain knowledge and plans into the tool implementation, with instruction-following gaps by enforcing certain constraints programmatically, and with long-context gaps by providing more compact tool observations. When many tools are created, we will also need to **manage tools** properly. By filtering out irrelevant tools, we can make tool selection easier and more efficient [38]. By tailoring tool schema, we can make tools more intuitive and easier to use [39].

c) Agent loop adaptations: Finally, we can also adapt the agent loop structure to mitigate failures from having a single unreliable model to take all actions with accumulating contexts. **Instrumenting code** is a common strategy to mitigate the unreliability of the plain agent loop. This can be done by adding deterministic checks (e.g., agent hooks [40]) to the loop that can be triggered when certain conditions are met, such as before or after calling certain tools. This helps with briding instruction-following gaps by enforcing some constraints programmatically. In safety contexts, this is also referred to as symbolic guardrails [41] to catch unsafe behaviors. **Orchestrating agents** is another strategy to go beyond single agent limitations, through building multi-agent systems [42], often to help with long-context problems. Many different topologies and orchestration strategies exist in the literature: For example, we can build systems where there is one main agent that spawns and orchestrates subagents [43], systems where different agents are peers and can communicate [44], or systems with independent agents that communicate in the end. Multi-agent system is an active research field – a more comprehensive discussion can be found in [42].

Trade-offs of harness adaptations. Creating more complex harnesses comes with costs: For example, while **adding contexts** is straightforward and often effective, it can lead to new failures on long-context and instruction-following due to growing context complexity. **Managing contexts** can help alleviate growing contexts, but comes with risks of additional planning / reasoning failures as models are tasked with more exploration work. **Creating tools** provides more powerful scaffolds, but can exacerbate tool-use failures when many tools are created. **Orchestrating agents** may increase inference cost and latency, and create pressure on tool-use for managing agents. Indeed, there is no one-size-fits-all solution when we create harnesses for SLMs. Instead, we need to create harnesses specialized to the task contexts and observed failures, so that we can make deliberate tradeoffs among the complexities introduced for each specific task.

C. Adapting harnesses for small language models

Our framework is *intentionally* not tied to model size and applies to language model agents broadly. This is because SLMs are not a static concept: their capabilities continue to improve over time. It is therefore more useful to reason about models in terms of *where their capabilities fall on a spectrum* (Figure 2, top) than to assume a static set of SLM limitations that may quickly become outdated.

That said, harness adaptation is especially relevant for small language models. This is because adaptation matters most when a task's capability demands exceed what a model can reliably provide, and this mismatch is more common for SLMs. As we will show in our experiments (Section IV), frontier LLMs are often strong enough that they can perform well with general-purpose harnesses. SLMs, by contrast, more often exhibit capability gaps that make creating specialized harness adaptation especially useful.

III. Automating SLM Harness Adaptation

The framework in Section II suggests that many SLM failures can be mitigated through appropriate harness adaptations. In practice, however, finding the right adaptation is challenging because the design space is large. Identifying the most effective strategy requires extensive trial and error, as it is common to have multiple failure modes intertwined (e.g., failing to use appropriate tools can interact with growing contexts), and there are usually multiple plausible adaptations that might address the observed issues (Figure 2). This process can be time-consuming for human engineers, especially when the harness may need to be updated repeatedly with evolving models [45].

Learning from the bitter lesson that human-knowledge-based methods ultimately fall short of general methods that leverage search and learning [46], we argue that **harness design should also be automated as a search problem** driven by data and evaluation, rather than relying on manual trial and error. We build on recent advances of automated harness optimization [10], [11], [47] and create a meta-agent powered harness optimizer that automates the search for identifying effective agent harnesses (Figure 3). The optimizer takes an SLM, a target task with training and validation data, and an

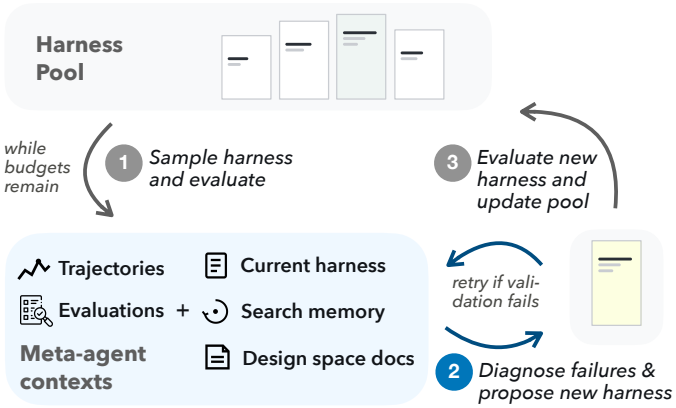

Fig. 3: Overview of the automated harness optimization loop.

initial agent harness as input, and outputs an optimized agent harness. We expose a general yet concrete search space to the optimizer, and create an optimization loop that searches this space – we present details of our design below.

A. Search Space

We instantiate the search space with the API interfaces of `software-agent-sdk`, an open-source framework for building software agents [48]. This gives the meta-agent a concrete set of editable components instead of an unconstrained space. In our implementation, the harness can modify:

- **Contexts:** system prompts (task instructions, examples), skills, and dynamic contexts that are appended on need.
- **Tools:** primitive tools (file editor, bash, browser), custom tools defined via classes, custom scripts.
- **Hooks:** custom scripts that are triggered based on tool use.
- **Context management:** external file systems, context condensers.
- **Sub-agents:** custom agents with dedicated contexts.

These components instantiate a practical subset of the adaptation strategies discussed in Section II, which makes the search space expressive enough to cover the failures we care about while still being structured enough for reliable editing.

B. Optimization Loop

We design an optimization loop that repeatedly (1) samples a harness and evaluates it, (2) diagnoses failures and proposes new harnesses, and (3) evaluates the harnesses (Figure 3):

1) **Sample and evaluate a harness.** We maintain a pool of previously tried harnesses and their validation scores. At each iteration, the outer loop selects one harness from this pool for further refinement. We currently use a GEPA-style genetic search procedure [49] to sample candidates from the Pareto front – i.e., the sampled harness will be optimal on at least one training instance. The selected harness is then executed and evaluated on a sampled batch of training instances. We log full trajectories, including tool calls, intermediate observations, outputs. We also save evaluation scores and feedback.

```python
SYS_PROMPT =  """You are an administrative operations agent.
Use the available tools to complete the assigned task
accurately."""

agent = Agent(
    llm=LLM(model="gemma-4-26b-a4b"),
    tools=[Tool(name=TerminalTool.name),
           Tool(name=FileEditorTool.name)],
    system_prompt=SYS_PROMPT,
    mcp_config=mcp_config, # 6 company-specific MCP tools
)
```

A generic harness ***provides simple prompt & raw MCP tools – SLM must create the plan and pick the right tools.***
SLM performs poorly (75%) compared to an LLM (97.3%).

```python
“""anti_loop_hook.py"""
[…]
if tool_name == "agent_company_send_message":
    if recipient in msgs and msgs[recipient] == message:
       print(json.dumps({
           "decision": "deny",
           "reason": "ERROR: You are sending the EXACT SAME
           message to this recipient again. STOP. Do not
           send this message again. Proceed to the next
           step of your task or finish."}))
[…]
```

```python
SYS_PROMPT =  """You are an administrative operations agent
handling equipment requests and budget approvals.

### Step-by-Step Workflow

**1. Discover Contacts & Initial Instructions:**]
[…]
"""

agent = Agent(
      llm=LLM(model="gemma-4-26b-a4b"),
      tools=[Tool(name=TerminalTool.name),
           Tool(name=FileEditorTool.name)],
      system_prompt=SYS_PROMPT,
      mcp_config=mcp_config, # 5 MCP tools after filtering
      hooks=[Hook(command="python3 anti_loop_hook.py")],
      filter_tools_regex=“^(?!get_conversation_history$).*"
)
```

An automatically adapted harness ***hands the SLM a fixed plan, a filtered tool set, as well as a hook that prevents looping.***
SLM now recovers LLM's performance at 8% of the cost.

Fig. 4: Simplified harness code identified by the meta-agent for the budget-approval task. The harness creates a new Python script as a tool-use hook, filters down tool sets, and creates more detailed workflow instructions.

2) **Diagnose failures and propose new a harness.** The inner-loop meta-agent then inspects these trajectories together with the sampled harness code and identifies what went wrong. The meta-agent edits the harness by changing harness components, such as adding a tool, inserting a reusable skill, etc. The meta-agent receives four main sources of context to make decisions:
   - **Task trajectories:** the latest batch of task trajectories annotated with outcome signals.
   - **Current harness:** the full Python implementation of

the harness being edited.
- **Search memory:** summaries of past proposals and their observed effects, so the agent does not repeatedly try the same ineffective fix.
- **Design-space documentation:** API references that document the available harness components.

Before evaluation, we first run a cheap sanity check on the proposed harness to catch invalid code and obviously broken harness implementations. If the proposal fails this check, the meta-agent is asked to repair it and try again up to a maximum number of attempts.

3) **Validate and save.** Once the new harness passes the sanity check, we evaluate it on the same sampled batch of training instances and observe if there is any improvement. If yes, we run a full evaluation on the validation set and add it to the pool if it improves over prior candidates.

C. Example of an Optimized Harness

In the budget-approval task (Figure 1), the default harness exposed a broad set of MCP tools and requires the model to plan the task end-to-end (Figure 4, top). This setup works for a frontier LLM but not an SLM that struggles to infer the correct procedure reliably and invoke the correct tools.

Our harness optimizer identified a few changes that successfully address the failure modes after 23 iterations. It narrowed the action space, rewrites the system prompt into an explicit step-by-step procedure, and introduces a custom `anti_loop_hook.py` safeguard that prevents the model from getting stuck in a loop (Figure 4, bottom). Rather than relying on the SLM to plan and recover from tool-use mistakes on its own, the harness externalizes the fragile parts of the workflow into prompts, filters, and runtime checks. The SLM now recovers the frontier model's performance with 8% of the cost.

IV. Experiments

We next employ the harness optimizer to study if, when, and how we can build (cost-)effective SLM agents with harness adaptations:

- **RQ1: To what extent can harness adaptation make SLM agents competitive with frontier-LLM agents?** We compare SLM agents with generic and adapted harnesses against frontier-LLM agents on accuracy, cost, and latency.
- **RQ2: How does task diversity impact harness adaptation effectiveness?** We examine whether adaptation is more effective for more repetitive tasks.
- **RQ3: How does model capability impact harness adaptation effectiveness?** We analyze how adaptation effectiveness varies across SLMs with different capabilities.
- **RQ4: What harness adaptation strategies are most common for improving SLM performance?** We inspect optimized harnesses through our framework (Section II), connecting observed improvements to failure modes and adaptation strategies.

A. Experiment Setups

We evaluate harness adaptation in a controlled setting where each task is paired with an initial generic harness, several candidate SLMs, and a frontier-LLM baseline. For each task-SLM pair, the optimizer searches for specialized harness that maximizes validation scores, and we evaluate the selected harness on a held-out test set. This design lets us analyze not only whether adaptation improves performance (RQ1), but also how the gains relate to task properties (RQ2), model capabilities (RQ3), and the types of adaptation strategies employed (RQ4).

Tasks. We purposefully select seven agentic tasks designed to reflect real-world business deployment scenarios. These tasks cover diverse routine business workflows such as auditing records, managing budgets, maintaining websites, writing software tests, and refactoring code. For each task, we curate a dataset of independent instances from public benchmarks. Each task comes with an objective evaluation metric, such as exact-match decisions, executable checks, or runnable code.

To study how task diversity affects the effectiveness of harness adaptation (RQ2), we curate these tasks with varying diversity: Lower-diversity tasks consist of instances that differ mainly in local inputs but follow the same procedure, whereas higher-diversity tasks include instances that require distinct solution strategies or control flows.

For each task, we use a 20/20/60 train/validation/test split. The meta-agent uses the training set to collect trajectories and propose harness updates, while the validation set is used to evaluate and select candidate harnesses. The test set is held out for the final evaluation of the selected harness. The tasks are summarized in Table I.

Models. We select and evaluate three models that represent the current generation of open-weight SLMs with strong instruction-following and tool-use abilities: `qwen3-coder-30b-a3b`, `ministral3-8b`, and `gemma-4-26b-a4b`. We compare them against a frontier LLM `gemini-3.1-pro-preview` with a generic harness.

B. Methodology

We organize the experiments around the four research questions introduced above.

RQ1: Understanding whether harness adaptation closes the SLM-LLM gap. To answer RQ1, we compare three agent configurations on each task: SLMs with the generic harness, the same SLMs with optimized harnesses, and the frontier LLM with the generic harness. We measure test accuracy, cost, and latency of each task-model configuration. Each configuration is run three times, and we report the average result to account for variations across agent runs. The cost is calculated based on tracked token usages and the reported official API pricing for each model. For `gemini-3.1-pro-preview` and `gemma-4-26b-a4b`, we use the API pricing from Google Vertex AI. For `qwen3-coder-30b-a3b` and `ministral3-8b`, we use the API pricing from AWS Bedrock.

For automated harness adaptation, we use `gemini-3.1-pro-preview` as the meta-agent model and implement the

TABLE I: We curate a diverse set of seven business-relevant agentic tasks. Each task is grounded in existing benchmarks with verifiable outcomes.

Task	Description	N	Common workflow	Evaluation criteria	Task creation process
attendance-auditing	Audit attendance and payroll records, then produce a summary spreadsheet.	100	Gather records; process data; write summary.	Check against ground-truth	Create task templates based on TheAgentCompany benchmark [50] and generate new instances.
budget-approval	Review budget requests against predefined budget constraints.	100	Collect requests; retrieve prices and budget policies; reply to requests.	Check against ground-truth	Create task templates based on TheAgentCompany benchmark [50] and generate new instances.
stock-alert	Monitor product stock levels and alert under pre-defined conditions.	100	Inspect data; compute indicators; send alert.	Check against ground-truth	Select from LOCA-Bench [51] and reuse its pipeline to generate new instances.
anomaly-detection	Analyze IoT time-series data to identify anomalies.	100	Inspect data; run analysis; verify anomalies; upload summary.	Check against ground-truth	Select from LOCA-Bench [51] and reuse its pipeline to generate new instances.
playwright-testing	Generate end-to-end playwright tests for a static website.	100	Inspect website; write tests; run and debug failures.	Run generated tests	Reuse instances from WebGenBench [52] with complete HTML websites generated by a frontier LLM (`gemini-3.1-pro`).
website-management	Complete shopping-site administration tasks through a browser interface.	50	Inspect website (in a loop); return exact final answer.	Check against ground-truth	Sample instances from CMS subset of WebArena [22] and keep the instances that a frontier LLM can solve.
code-refactoring	Refactor a repository based on user instructions.	100	Understand repository; inspect relevant files; edit code; iterate.	Run tests to check refactored code's AST	Reuse instances from RefactorBench [53].

agent harnesses using `software-agent-sdk` [54]. Each optimization run has a budget of $20,[3] which covers meta-agent analysis, task-agent rollouts, and candidate harness evaluations. For each task-model pair, we run the optimization procedure three times and keep the harness with the best validation score.

RQ2: Analyzing how task diversity impacts harness adaptation effectiveness. Our hypothesis is that higher task diversity lowers adaptation effectiveness by impacting how reusable a harness can be. If task instances share recurring workflows and failure patterns, a single adapted harness can encode those patterns and improve many instances; otherwise, it is harder to create a comprehensive harness to cover all diverse failures. We test the hypothesis with two sets of experiments:

First, we measure task diversity in our task suite and test whether it correlates with optimized performance. We operationalize task diversity as behavioral variation across LLM agent trajectories. For each task, we collect LLM rollouts, extract their tool-call sequences, and compute the average pairwise normalized Levenshtein distance between these sequences. This distance serves as a proxy for workflow diversity: tasks whose instances induce similar tool-use sequences have low diversity, whereas tasks whose instances require different tools or tool orderings have higher diversity.

Because this cross-task analysis may be confounded by other task differences, we also construct diversity-controlled variants of two tasks: attendance and budget. For each task, we create a pool of templates, where each template represents a distinct workflow. In the budget task, for example, templates may require total-cost tracking, remaining-budget calculation, emergency-fund handling, or policy-based approval decisions. We then vary the number of templates included in the dataset to control the level of workflow diversity.

RQ3: Analyzing how model capability impacts harness adaptation effectiveness. We hypothesize that higher model capability increases adaptation effectiveness, because it determines how much task difficulty can remain with the SLM. While harnesses can offload some difficulty, the model must still follow instructions, use tools correctly, and complete the core sub-tasks that cannot be offloaded. To analyze the role of model capability, we use the average benchmark scores reported by Artificial Analysis [55] as a proxy for models' general capability, and use these scores to contextualize performance trends.

RQ4: Analyzing how optimized harnesses improve performance. To answer RQ4, we manually inspect the optimized harnesses produced for each task-model pair. We categorize the discovered edits according to the framework in Section II, recording which failure modes are addressed and which adaptation strategies are used.

C. Threats to Validity

Despite various mitigations described as part of the experimental design above, like all experiments, ours has limitations and should be interpreted accordingly. **Internal validity:** The agent runs and the optimization process are highly stochastic, even with our mitigation of multiple repeated runs. **External validity:** Our findings are based on a particular set of tasks,

[3]We choose a budget of $20 as we observed optimization saturation in early piloting tasks given this budget. This results in a total budget of $1260 for our main optimization experiments.

TABLE II: Summary of performance and inference cost across evaluation settings. Each cell reports accuracy (top), average cost per instance in USD (middle), and average end-to-end latency (bottom).

Model	Attn.	Budget	Stock	Anom.	Playwr.	Web.	Refact.	Avg.
Large Language Models								
`gemini-3.1-pro-preview`	96.5 $0.894 206s	97.3 $0.219 70s	86.7 $5.785 319s	98.9 $1.450 148s	85.7 $0.667 213s	76.7 $2.011 157s	86.1 $1.116 155s	89.7 $1.735 181s
Small Language Models								
`gemma-4-26b-a4b`	91.7 $0.025 116s	75.0 $0.039 77s	5.0 $0.041 428s	5.0 $0.058 367s	9.9 $0.018 461s	1.1 $0.071 425s	32.2 $0.046 423s	31.4 $0.043 328s
+ *optimized harness*	95.7 $0.027 215s	98.3 $0.017 33s	58.9 $0.089 177s	99.4 $0.033 61s	98.1 $0.021 85s	45.6 $0.165 206s	65.0 $0.147 171s	80.2 $0.071 135s
`qwen3-coder-30b-a3b`	23.7 $0.030 64s	61.0 $0.025 73s	9.4 $0.210 148s	1.1 $0.076 100s	31.4 $0.058 176s	1.1 $0.110 98s	60.6 $0.089 91s	26.9 $0.085 107s
+ *optimized harness*	73.7 $0.025 57s	79.3 $0.024 53s	96.7 $0.061 51s	93.9 $0.031 39s	93.6 $0.062 136s	24.4 $0.122 96s	62.2 $0.121 100s	74.8 $0.064 76s
`ministral-3-8b`	0.2 $0.154 310s	28.8 $0.012 22s	0.0 $0.099 132s	0.0 $0.167 304s	0.4 $0.172 370s	5.6 $0.056 65s	31.7 $0.112 154s	9.5 $0.110 194s
+ *optimized harness*	63.3 $0.002 9s	53.7 $0.037 95s	0.0 $0.098 123s	0.0 $0.167 296s	22.3 $0.181 401s	5.6 $0.088 110s	30.0 $0.120 168s	25.0 $0.099 172s

best good poor

models, and one optimizer implementation, and may not transfer cleanly to other tasks, future models, or different optimizer implementations. We focus on tasks with clear success metrics and clean environments, which do not fully capture the complex deployment scenarios of specialized agents. **Reproducibility:** Because we use closed-source frontier LLM APIs that are black-box and may evolve over time, exact replication of the optimization may be difficult once these APIs change or get deprecated. However, we believe our findings generalize well to any frontier LLMs with strong capabilities.

D. Results

SLMs with optimized agent harnesses can recover 89% of LLM performance at 4% cost (RQ1). First, we highlight that for 16 out of 21 task-SLM pairs, we have seen a significant performance boost for small language models when their harnesses are optimized, with seven pairs closing the SLM-LLM gap (Table II). The best performing SLM, `gemma-4-26b-a4b`, can recover 89% of LLM performance at 4% cost, with 25% latency reduction.

While there is a one-time offline optimization cost per task ($20 in our experiments), this cost is amortized after deployment because the optimized harness can be reused across agent runs. Based on the per-run cost savings we observe, the optimization cost is already recovered after 13 runs on average across tasks and models.

Harness adaptation is more effective on less diverse tasks (RQ2). We next analyze when harness adaptation is more effective. Across the seven tasks, we observed a strong, statistically significant negative correlation (Spearman $\rho = -0.96$) between task diversity and optimized harness performance (Figure 5, left). This is further supported by our controlled experiment on task diversity: Among the diversity-controlled task variants, we observed that optimized harnesses work worse with more diverse instances, dropping performance from 89.1% to 68.0% (Figure 5, right).

Qualitatively, we observe that tasks like `attendance-auditing` are easier to adapt, as their instances have shared failure modes (e.g., missing knowledge and bad output formatting). In contrast, tasks like `code-refactoring` are harder to adapt, as different repositories represent different contexts, and different user queries contain divergent requirements for refactoring, which are hard to fully account for in the harness.

Harness adaptation is more effective when the model has stronger base capabilities (RQ3). Across SLMs, we observe a clear trend that models with stronger capabilities perform better and improve more with optimized harnesses (Figure 6). This reflects how weaker SLMs struggle with the intrinsic task difficulty that cannot be offloaded to the harness. For example, `playwright-testing` requires strong coding capability, of understanding HTML webpages and writing tests, that cannot be fully delegated to the harness. Similarly, `stock-alert` tasks require strong **long-context** capabilities, which are difficult to fully capture within a harness.

Instruction-following and knowledge failures are the dominant failure modes (RQ4). Reviewing the optimized

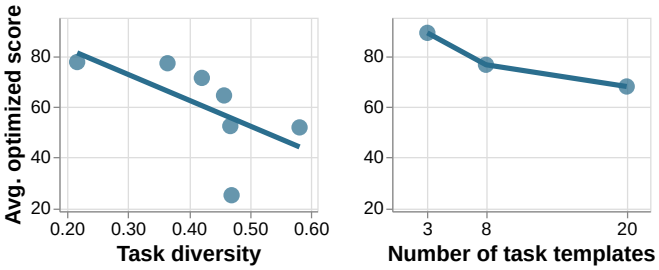

Fig. 5: More diverse tasks are harder to optimize. Each point averages across different models (left), or different tasks (right).

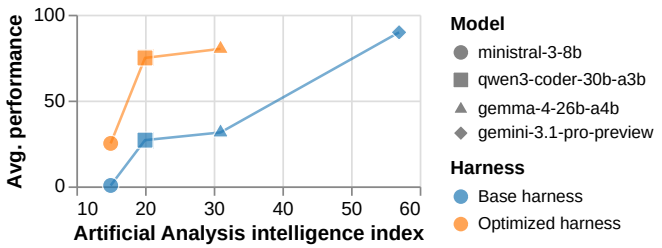

Fig. 6: Less capable models are harder to optimize.

harnesses, we found that the most commonly addressed failure modes are instruction-following (81%) and knowledge failures (81%), followed by tool-use (62%) and long-context failures (33%). These failure modes are most commonly addressed by adding contexts (86%), creating tools (43%), and managing tools (29%). For example, the best `gemma-4-26b` harness for the `anomaly-detection` task combines three strategies together: it creates a custom `query_mock_bigquery` tool that addresses long-context issues from the default `bigquery_run_query` MCP tool; it filters the agent's active tool set from 40+ MCP tool down to seven, and externalizes the environment's table-naming convention in the system prompt.

Noticeably, we do not see successful harness adaptation of creating sub-agents. We hypothesize that this is an artifact from current SLMs' limited agent management capabilities – i.e., they struggle with tracking and managing sub-agents' work progress and coordinating among different agents. Alternatively, this can be seen as limitations of the meta-agent, which does not generate a good harness for easier sub-agent management.

Different SLMs require different adaptation strategies (RQ4). Across different SLMs, we observe major differences in how these models can be improved. For example, `ministral-3-8b` sometimes struggles with using file editing tools, while `qwen3-coder-30b-a3b` occasionally fails to emit correct JSON tool calls (but instead emit raw XMLs). Each model's failure patterns require tailored harness adaptation that does not cleanly transfer. This indicates that, there is no one-size-fits-all harness, and that harness optimization needs to be re-run each time we migrate to a new SLM.

For example, for the `playwright-testing` task, the optimized harness for `gemma-4-26b-a4b` adds contexts to externalize knowledge and enforce instructions, and create hooks to enforce successful pytest runs. Meanwhile, the optimized harness for `ministral-3-8b` creates new tools to mitigate the model's file-editing failures.

V. Discussion

A. How to optimize agent harnesses effectively?

Our experiments show it is promising to automatically optimize agent harnesses for SLMs. Throughout the optimizer development, we also learned a few lessons about building effective harness optimizers, which we share below.

The main design question of building a harness optimizer is on how to the allocate optimization budgets well to find genuine improvements. In our optimizer, the budget can be spent on (1) analyzing failures, or (2) evaluating candidate fixes, with multiple iterations: $B = \sum_{t=1}^{T} \left(b_t^{\text{analyze}} + b_t^{\text{evaluate}} \right)$. Given a fixed budget, we can choose to (1) analyze deeper, (2) evaluate more extensively, or (3) run more iterations.

Lesson 1: Improve analysis quality, with better contexts and intelligence. Harness optimization is bottlenecked by diagnosis quality – spending budget on evaluation or additional iterations is useful only when the meta-agent can propose good harness candidates. We found that the meta-agent performs better when it receives high-fidelity failure evidence rather than aggressively summarized traces. In early iterations, we converted task trajectories from structured JSON into post-processed markdown, but this sometimes removed details that were useful for understanding failures. Passing raw JSON worked better in practice: although it is less human-friendly, the frontier meta-agent could manipulate and inspect it effectively while benefiting from the extra information.

The same lesson applies to the choice of meta-agent model. Replacing `gemini-3.1-pro-preview` with a cheaper model `gemini-3-flash-preview` affords more iterations, but the lower-quality diagnoses and edits more than offset the extra exploration, yielding worse final harnesses.

At the same time, a good frontier model also does not require extensive hand-holding. We found that showing successful trajectories from frontier models or manually written heuristics about which adaptations match which failure modes did not consistently improve outcomes. This suggests that, once given faithful failure evidence and an editable harness, a strong meta-agent is well-positioned to infer targeted repairs on its own.

Lesson 2: Prioritize exploration quality and diversity, not quantity. Because full agent evaluation is expensive, harness optimization offers far fewer search steps than single-turn prompt optimization. This makes the quality and diversity of proposals more important than simply maximizing the number of iterations. For quality, it is important to use a good frontier model for proposing new harnesses, as we have discussed before. For diversity, we observed two helpful design choices:

First, keep a memory of prior attempts. Without logging previous edits and their outcomes, the meta-agent tends to rediscover similar fixes, wasting budget on redundant local exploration. Second, run several independent searches than to

spend the same budget on one long trajectory. Multiple runs cover more distinct regions of the design space and increase the chance that at least one run discovers a strong adaptation.

B. Implications

For agent developers. When developing new agents for routine business tasks where cost matters, developers should consider and experiment with a wide range of SLMs, instead of throwing all tasks to a frontier LLM. SLMs can be particularly effective, when the task is more repetitive and more task demands can be offloaded to the harness. To choose an appropriate SLM, a good proxy is benchmark scores (cf. Figure 6), which we observe to correlate with better performance after harness adaptations. Heuristically, small Mixture-of-Experts (MoE) models (e.g., `gemma-4-26b-a4b`) are good candidates, as they provide great model capabilities with a low inference cost.

To adapt SLM harnesses, developers should consider an automated approach beyond manual engineering. This would require them to curate task instances, with clear evaluation criteria, to optimize against. Generally, we observe a meta-agent powered by frontier LLMs to be fairly strong on the harness optimization task.

For model developers. Training practically useful SLMs goes beyond optimizing benchmark scores. To achieve better cost-performance tradeoffs, model developers should *optimize for harness compatibility*. For example, a good SLM should reliably work with various prompt scaffolds (instruction-following) and tool scaffolds (tool-use). This would require SLMs to be exposed to a diverse range of environments, with different prompts and tools available, during their training.

For researchers. Automated harness optimization is an emerging research topic with many open challenges and opportunities. Designing better optimizers (cf. Section V-A) will benefit from evidence from larger-scale experiments and a more thorough exploration of the design space. For example, future research can develop more fine-grained task and model profiles based on individual failure modes / capabilities to better predict when a task is likely to benefit from harness adaptation, and what kinds of harness edits are most likely to help.

Another promising direction is to move beyond a single adapted harness per task. For more diverse tasks that contain clusters of repetitive instances, researchers could study *mixtures-of-harnesses*: collections of specialized harnesses paired with routing systems that select the appropriate harness for each instance. This may avoid forcing one harness to cover too many cases, which can make the harness overly complex and difficult for SLMs to work with. Similarly, when task distributions drift over time, online monitoring and adaptation systems could maintain multiple harness versions and update routing or harness as new failures emerge.

VI. Related Work

Building effective agent harnesses is a widely discussed topic among practitioners [56], and we have outlined the major capabilities and harness components that the literature discusses in Section II. Here, we discuss additional related work on using SLMs for agentic tasks and automating agent harness design.

Using Small Language Models for Agentic Tasks. With growing interest in using SLMs for agentic tasks [8], many industry labs have been training SLMs with strong agentic capabilities [14]. These provide us with a good foundation to start with – without basic tool-use , or instruction-following capabilities, it would be very hard-pressed to build any effective agents to mitigate the model's limitations.

On the harness side, we have seen work emerging to address SLM limitations. For example, Lee et al. [39] propose to adapt tool schema to help with smaller models with limited tool-use capabilities with unfamiliar schema. Srivastava et al. [57] design a whole set of human-engineered heuristics to mitigate SLM limitations (e.g., instruction-following with long prompts). These approaches heavily depend on observations of specific SLM behaviors and may not generalize well to newer models with improved capabilities. In contrast, our work intentionally does not make assumptions on what SLMs are capable of, but employs a meta-agent to empirically diagnose their trajectories and create tailored harnesses.

Automating Agent Design. Automating agent design has been explored a lot in the literature. ADAS [58] is one of the early work that uses an LLM to optimize agents defined in code. Other work has extended this to agents with explicitly modular components (e.g., planning, memory) [59], to topology in multi-agent systems [60], [61], etc. Most of these work focus on agentic systems with pre-defined workflows (vs. tool-use agents in our work) and tasks that do not require interacting with environments. They also tend to focus on creating harnesses that improve performance by increasing model compute (e.g., test-time scaling), instead of by moving work from models to harnesses as in our work.

Another line of work of self-improving agents [10], [47] have explored using agents to modify its own code. A common observation is that these agent often make new tools to improve task performance. Similarly, Lee et al. [11] have explored automating agent harness designs with a meta agent. These work's goals are to improve agent performance on given tasks, often the ones that are challenging even for frontier models (e.g., TerminalBench [21]), while we focus on identifying SLM harnesses that recover frontier-agent performance at lower cost. Concurrent work on Life-Harness [62] also adapts agent harnesses for frozen models, but focuses on evolving a harness that transfers across benchmark environments and model backbones, whereas we study harness adaptation as a cost-performance strategy for making specialized SLM agents competitive with frontier-LLM agents on routine business tasks.

VII. Conclusion

In this work, we present a first in-depth empirical analysis of if, when, and how harness adaptations can afford construction of (cost-)effective SLM agents. We show that, given a well-described design space, a harness optimizer can automatically identify effective harness adaptations that significantly boost SLM performance, making it possible to build effective agents

at a fraction of the cost of LLMs. Our analysis found that this works best for more repetitive tasks and SLM with strong base capabilities. We can also explain *why* the optimized harnesses are effective, by connecting the changes to the observed failure modes and established adaptation strategies.

ACKNOWLEDGMENT

This work is supported by Apple gift funding, the Amazon AI Research gift fund, and the Gemma Academic Program GCP Credit Award. We thank Andrew Walkingshaw and Eric Liang Yang for their discussion and feedback throughout the project. We also thank Manisha Mukherjee and Vasu Vikram for their feedback on this work.

REFERENCES

[1] Amazon Web Services, "How Cox automotive launched AI agents at scale using Amazon Bedrock AgentCore," https://aws.amazon.com/ko/solutions/case-studies/cox-auto-case-study/, 2026.

[2] Salesforce, "Engine provides personalized service to 1 million travelers with Agentforce," https://www.salesforce.com/customer-stories/engine/, 2026.

[3] Reuters, "Anthropic clinches $380 billion valuation after $30 billion funding round," https://www.reuters.com/technology/anthropic-valued-380-billion-latest-funding-round-2026-02-12/, 2026.

[4] L. Bratton, "Uber CTO shows how claude code can blow up AI budgets," https://www.theinformation.com/newsletters/applied-ai/uber-cto-shows-claude-code-can-blow-ai-budgets, 2026.

[5] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv *et al.*, "Qwen3 technical report," *arXiv preprint arXiv:2505.09388*, 2025.

[6] K. Team, T. Bai, Y. Bai, Y. Bao, S. Cai, Y. Cao, Y. Charles, H. Che, C. Chen, G. Chen *et al.*, "Kimi K2. 5: Visual Agentic Intelligence," *arXiv preprint arXiv:2602.02276*, 2026.

[7] DeepSeek-AI, "Deepseek-v4: Towards highly efficient million-token context intelligence," 2026.

[8] P. Belcak, G. Heinrich, S. Diao, Y. Fu, X. Dong, S. Muralidharan, Y. C. Lin, and P. Molchanov, "Small language models are the future of agentic ai," *arXiv preprint arXiv:2506.02153*, 2025.

[9] OpenHands Team, "Introducing the openhands index," https://openhands.dev/blog/openhands-index, 2026.

[10] J. Zhang, S. Hu, C. Lu, R. Lange, and J. Clune, "Darwin godel machine: Open-ended evolution of self-improving agents," *arXiv preprint arXiv:2505.22954*, 2025.

[11] Y. Lee, R. Nair, Q. Zhang, K. Lee, O. Khattab, and C. Finn, "Meta-harness: End-to-end optimization of model harnesses," *arXiv preprint arXiv:2603.28052*, 2026.

[12] OpenAI, "Introducing gpt-5.2," *OpenAI*, 2025. [Online]. Available: https://openai.com/index/introducing-gpt-5-2/

[13] Anthropic, "Introducing claude opus 4.7," *Anthropic*, 2026. [Online]. Available: https://www.anthropic.com/news/claude-opus-4-7

[14] Q. Team, "Qwen3.5: Accelerating productivity with native multimodal agents," February 2026. [Online]. Available: https://qwen.ai/blog?id=qwen3.5

[15] S. G. Patil, H. Mao, C. Cheng-Jie Ji, F. Yan, V. Suresh, I. Stoica, and J. E. Gonzalez, "The Berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models," in *Forty-second International Conference on Machine Learning*, 2025.

[16] C. Bandi, B. Hertzberg, G. Boo, T. Polakam, J. Da, S. Hassaan, M. Sharma, A. Park, E. Hernandez, D. Rambado *et al.*, "Mcp-atlas: A large-scale benchmark for tool-use competency with real mcp servers," *arXiv preprint arXiv:2602.00933*, 2026.

[17] J. Zhou, T. Lu, S. Mishra, S. Brahma, S. Basu, Y. Luan, D. Zhou, and L. Hou, "Instruction-following evaluation for large language models," *arXiv preprint arXiv:2311.07911*, 2023.

[18] J. Wei, N. Karina, H. W. Chung, Y. J. Jiao, S. Papay, A. Glaese, J. Schulman, and W. Fedus, "Measuring short-form factuality in large language models," *arXiv preprint arXiv:2411.04368*, 2024.

[19] N. Jain, K. Han, A. Gu, W.-D. Li, F. Yan, T. Zhang, S. Wang, A. Solar-Lezama, K. Sen, and I. Stoica, "Livecodebench: Holistic and contamination free evaluation of large language models for code," *arXiv preprint arXiv:2403.07974*, 2024.

[20] T. Patwardhan, R. Dias, E. Proehl, G. Kim, M. Wang, O. Watkins, S. P. Fishman, M. Aljubeh, P. Thacker, L. Fauconnet *et al.*, "Gdpval: Evaluating ai model performance on real-world economically valuable tasks," *arXiv preprint arXiv:2510.04374*, 2025.

[21] M. A. Merrill, A. G. Shaw, N. Carlini, B. Li, H. Raj, I. Bercovich, L. Shi, J. Y. Shin, T. Walshe, E. K. Buchanan *et al.*, "Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces," *arXiv preprint arXiv:2601.11868*, 2026.

[22] S. Zhou, F. F. Xu, H. Zhu, X. Zhou, R. Lo, A. Sridhar, X. Cheng, T. Ou, Y. Bisk, D. Fried *et al.*, "Webarena: A realistic web environment for building autonomous agents," *arXiv preprint arXiv:2307.13854*, 2023.

[23] J. Wei, Z. Sun, S. Papay, S. McKinney, J. Han, I. Fulford, H. W. Chung, A. T. Passos, W. Fedus, and A. Glaese, "Browsecomp: A simple yet challenging benchmark for browsing agents," *arXiv preprint arXiv:2504.12516*, 2025.

[24] C. Yang, Y. Shi, Q. Ma, M. X. Liu, C. Kästner, and T. Wu, "What prompts don't say: Understanding and managing underspecification in llm prompts," *arXiv preprint arXiv:2505.13360*, 2025.

[25] M. Cemri, M. Z. Pan, S. Yang, L. A. Agrawal, B. Chopra, R. Tiwari, K. Keutzer, A. Parameswaran, D. Klein, K. Ramchandran *et al.*, "Why do multi-agent llm systems fail?" *arXiv preprint arXiv:2503.13657*, 2025.

[26] K. Zhu, Z. Liu, B. Li, M. Tian, Y. Yang, J. Zhang, P. Han, Q. Xie, F. Cui, W. Zhang *et al.*, "Where llm agents fail and how they can learn from failures," *arXiv preprint arXiv:2509.25370*, 2025.

[27] Anthropic, "Claude code overview," https://code.claude.com/docs/en/overview, 2026.

[28] OpenAI, "Codex," https://developers.openai.com/codex, 2026.

[29] Q. Zhang, C. Hu, S. Upasani, B. Ma, F. Hong, V. Kamanuru, J. Rainton, C. Wu, M. Ji, H. Li *et al.*, "Agentic context engineering: Evolving contexts for self-improving language models," *arXiv preprint arXiv:2510.04618*, 2025.

[30] T. Gloaguen, N. Mündler, M. Müller, V. Raychev, and M. Vechev, "Evaluating agents. md: Are repository-level context files helpful for coding agents?" *arXiv preprint arXiv:2602.11988*, 2026.

[31] Anthropic, "Skill authoring best practices," https://platform.claude.com/docs/en/agents-and-tools/agent-skills/best-practices, 2026.

[32] A. L. Zhang, T. Kraska, and O. Khattab, "Recursive language models," *arXiv preprint arXiv:2512.24601*, 2025.

[33] M. Kang, W.-N. Chen, D. Han, H. A. Inan, L. Wutschitz, Y. Chen, R. Sim, and S. Rajmohan, "Acon: Optimizing context compression for long-horizon llm agents," *arXiv preprint arXiv:2510.00615*, 2025.

[34] Y.-A. Xiao, P. Gao, C. Peng, and Y. Xiong, "Improving the efficiency of llm agent systems through trajectory reduction," *arXiv preprint arXiv:2509.23586*, 2025.

[35] I. Xu, G. Zeng, Z. He, C. Jin, A. Pareja, D. Gutfreund, C. Gan, and Z.-W. Hong, "Boad: Discovering hierarchical software engineering agents via bandit optimization," *arXiv preprint arXiv:2512.23631*, 2025.

[36] Anthropic, "Introducing the model context protocol," https://www.anthropic.com/news/model-context-protocol, Nov. 2024, announcement, accessed 2026-05-02.

[37] Z. Z. Wang, A. Gandhi, G. Neubig, and D. Fried, "Inducing programmatic skills for agentic tasks," *arXiv preprint arXiv:2504.06821*, 2025.

[38] K. Yang, Y. Liu, S. Chaudhary, R. Fakoor, P. Chaudhari, G. Karypis, and H. Rangwala, "Agentoccam: A simple yet strong baseline for llm-based web agents," *arXiv preprint arXiv:2410.13825*, 2024.

[39] J. Lee, W. Song, J. Han, H. Pyun, and Y. Jo, "Don't adapt small language models for tools; adapt tool schemas to the models," *arXiv preprint arXiv:2510.07248*, 2025.

[40] Anthropic, "Intercept and control agent behavior with hooks," https://code.claude.com/docs/en/agent-sdk/hooks, 2026.

[41] Y. Hong, Y. She, E. Kang, C. S. Timperley, and C. Kästner, "Symbolic guardrails for domain-specific agents: Stronger safety and security guarantees without sacrificing utility," *arXiv preprint arXiv:2604.15579*, 2026.

[42] Y. Kim, K. Gu, C. Park, C. Park, S. Schmidgall, A. A. Heydari, Y. Yan, Z. Zhang, Y. Zhuang, Y. Liu *et al.*, "Towards a science of scaling agent systems," *arXiv preprint arXiv:2512.08296*, 2025.

[43] A. Fourney, G. Bansal, H. Mozannar, C. Tan, E. Salinas, F. Niedtner, G. Proebsting, G. Bassman, J. Gerrits, J. Alber *et al.*, "Magentic-one: A

generalist multi-agent system for solving complex tasks," *arXiv preprint arXiv:2411.04468*, 2024.

[44] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, J. Wang, C. Zhang, Z. Wang, S. K. S. Yau, Z. Lin *et al.*, "Metagpt: Meta programming for a multi-agent collaborative framework," in *The twelfth international conference on learning representations*, 2023.

[45] Anthropic, "Harness design for long-running application development," https://www.anthropic.com/engineering/harness-design-long-running-apps, 2026.

[46] R. Sutton, "The bitter lesson," http://www.incompleteideas.net/IncIdeas/BitterLesson.html, 2019.

[47] C. S. Xia, Z. Wang, Y. Yang, Y. Wei, and L. Zhang, "Live-swe-agent: Can software engineering agents self-evolve on the fly?" *arXiv preprint arXiv:2511.13646*, 2025.

[48] X. Wang, S. Rosenberg, J. Michelini, C. Smith, H. Tran, E. Nyst, R. Malhotra, X. Zhou, V. Chen, R. Brennan *et al.*, "The openhands software agent sdk: A composable and extensible foundation for production agents," *arXiv preprint arXiv:2511.03690*, 2025.

[49] L. A. Agrawal, S. Tan, D. Soylu, N. Ziems, R. Khare, K. Opsahl-Ong, A. Singhvi, H. Shandilya, M. J. Ryan, M. Jiang *et al.*, "Gepa: Reflective prompt evolution can outperform reinforcement learning," *arXiv preprint arXiv:2507.19457*, 2025.

[50] F. F. Xu, Y. Song, B. Li, Y. Tang, K. Jain, M. Bao, Z. Z. Wang, X. Zhou, Z. Guo, M. Cao, M. Yang, H. Y. Lu, A. Martin, Z. Su, L. Maben, R. Mehta, W. Chi, L. Jang, Y. Xie, S. Zhou, and G. Neubig, "Theagentcompany: Benchmarking llm agents on consequential real world tasks," 2024. [Online]. Available: https://arxiv.org/abs/2412.14161

[51] W. Zeng, Y. Huang, and J. He, "Loca-bench: Benchmarking language agents under controllable and extreme context growth," *arXiv preprint arXiv:2602.07962*, 2026.

[52] Z. Lu, Y. Yang, H. Ren, H. Hou, H. Xiao, K. Wang, W. Shi, A. Zhou, M. Zhan, and H. Li, "Webgen-bench: Evaluating llms on generating interactive and functional websites from scratch," 2025. [Online]. Available: https://arxiv.org/abs/2505.03733

[53] D. Gautam, S. Garg, J. Jang, N. Sundaresan, and R. Z. Moghaddam, "Refactorbench: Evaluating stateful reasoning in language agents through code," *arXiv preprint arXiv:2503.07832*, 2025.

[54] X. Wang, S. Rosenberg, J. Michelini, C. Smith, H. Tran, E. Nyst, R. Malhotra, X. Zhou, V. Chen, R. Brennan, and G. Neubig, "The openhands software agent sdk: A composable and extensible foundation for production agents," 2025. [Online]. Available: https://arxiv.org/abs/2511.03690

[55] Artificial Analysis, "Artificial analysis," https://artificialanalysis.ai/.

[56] E. S. and B. Zhang, "Building effective agents," *Anthropic Engineering Blog*, 2024. [Online]. Available: https://www.anthropic.com/engineering/building-effective-agents

[57] G. Srivastava, A. Hussain, C. Wang, Y. C. Lin, and X. Wang, "Effgen: Enabling small language models as capable autonomous agents," *arXiv preprint arXiv:2602.00887*, 2026.

[58] S. Hu, C. Lu, and J. Clune, "Automated design of agentic systems," *arXiv preprint arXiv:2408.08435*, 2024.

[59] Y. Shang, Y. Li, K. Zhao, L. Ma, J. Liu, F. Xu, and Y. Li, "Agentsquare: Automatic llm agent search in modular design space," *arXiv preprint arXiv:2410.06153*, 2024.

[60] G. Zhang, L. Niu, J. Fang, K. Wang, L. Bai, and X. Wang, "Multi-agent architecture search via agentic supernet," *arXiv preprint arXiv:2502.04180*, 2025.

[61] H. Zhou, X. Wan, R. Sun, H. Palangi, S. Iqbal, I. Vulić, A. Korhonen, and S. Ö. Arık, "Multi-agent design: Optimizing agents with better prompts and topologies," *arXiv preprint arXiv:2502.02533*, 2025.

[62] T. Xu, H. Wen, and M. Li, "Adapting the interface, not the model: Runtime harness adaptation for deterministic llm agents," *arXiv preprint arXiv:2605.22166*, 2026.