How do different agent harnesses affect the cost-efficiency and task-solving performance of coding models on software-engineering tasks?

Fares Rahmani

Abstract TODO: Write the abstract after the study is complete.

Table of contents

List of Figures

List of Tables

1 Introduction

Placeholder section: Replace the bullets below with the paper's content. No findings, company facts, or sources have been added yet.

1.1 Problem statement

- TODO: State the problem to be investigated.

1.2 Research question

- TODO: Formulate the research question.

1.3 Objective

- TODO: State the objective of the Praxisarbeit.

1.4 Scope

- TODO: Define the scope and boundaries of the work.

2 Current State in the Company

2.1 Internal Survey

An internal survey conducted between August and September 2026 received 57 responses. It was distributed through four Google Chat rooms: two team rooms and two rooms focused on AI users and AI products. Several employees were also contacted directly. This sampling favors employees with high AI interest and adoption. The results therefore describe an AI-active subgroup rather than company-wide usage.

Respondents reported one first-choice configuration and up to three additional configurations used at least weekly. The two Claude Code answer labels were merged. Adoption rates count each respondent once per harness or model; configuration analyses count each distinct respondent–harness–model pair once. "Other" and automatically selected models are excluded where a named model or provider is required.

The model list was fixed when the survey was published and was not updated for later releases. Models released afterwards, including DeepSeek V4.1 Flash and GLM-5.3-Flash, were therefore not available as named choices. Their adoption may consequently be understated in the survey.

2.2 Harness-Model Configurations

The combined configurations show a clear Claude-centered pattern. Claude Code with Claude Opus 5 was the most common primary configuration, selected by 18 of 57 respondents (31.6%), and appeared in 24 weekly workflows (42.1%). Claude Code with Claude Sonnet 5 appeared in 15 workflows (26.3%), followed by Claude Code with Claude Sonnet 4.6 in 11 (19.3%). Figure 1 shows the eight most common named configurations.

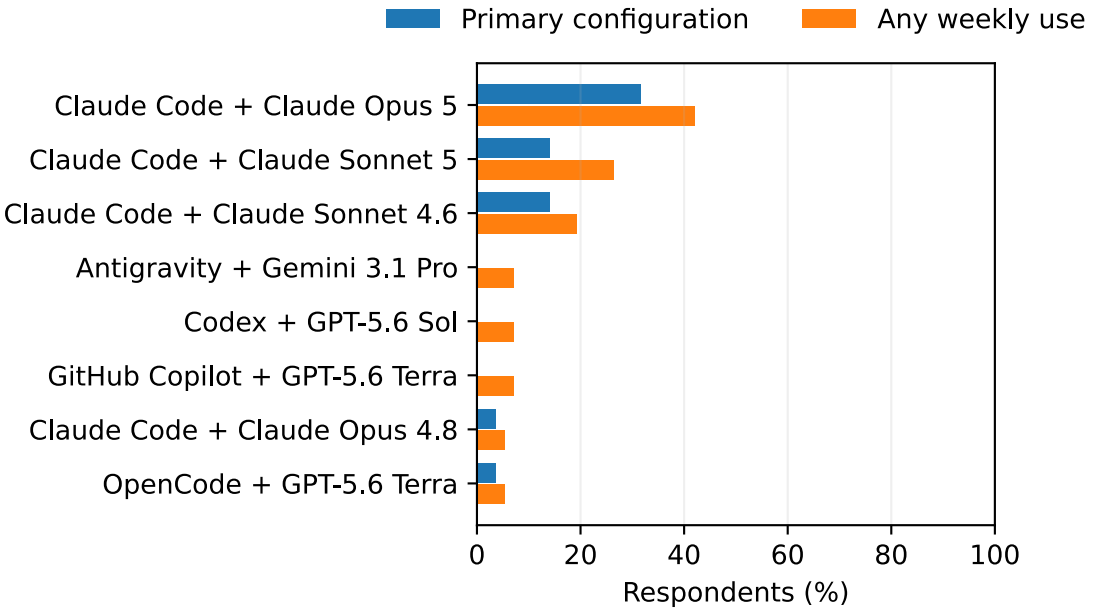

Figure 1: Most common named harness–model configurations among survey respondents (n = 57).

Source: own analysis of the internal survey. Unnamed configurations are omitted.

2.3 Harness Usage

Claude Code was the first-choice harness for 41 of 57 respondents (71.9%) and was used weekly by 52 (91.2%). OpenCode ranked second with 8 first-choice responses (14.0%) and 15 weekly users (26.3%). Codex and GitHub Copilot each had 9 weekly users (15.8%); Pi had 5 (8.8%). Figure 2 compares first-choice and weekly use.

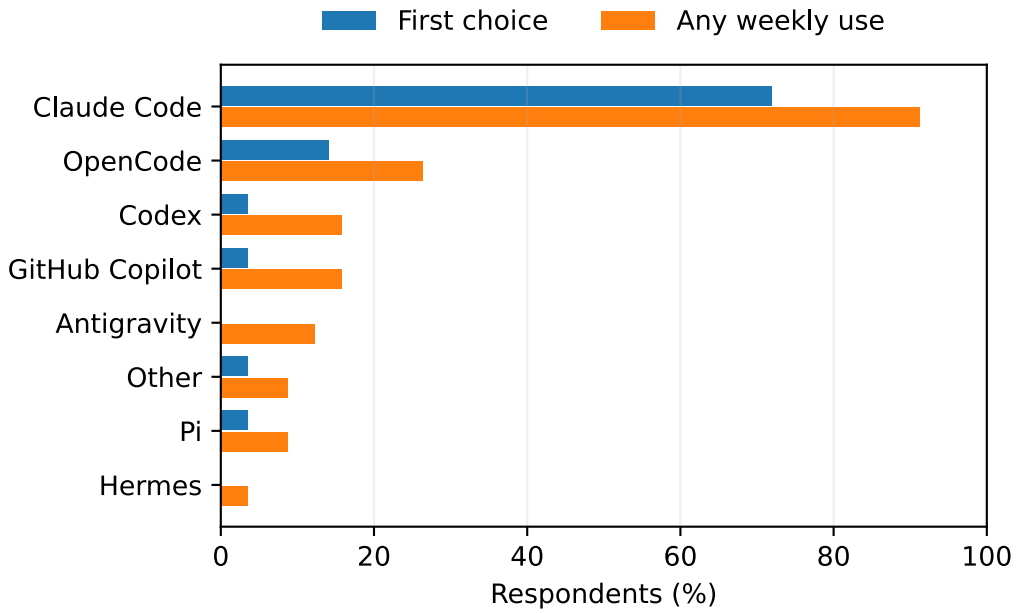

Figure 2: First-choice and weekly harness usage among survey respondents (n = 57).

Source: own analysis of the internal survey.

Multiple setups were common. A total of 42 respondents (73.7%) reported at least one additional setup or model used weekly, and 32 (56.1%) used two or more harnesses. Claude Code with OpenCode was the most common multi-harness combination, reported by 12 respondents.

Alternative harnesses mostly complemented Claude Code. Among weekly users, 12 of 15 OpenCode users and 8 of 9 Codex users also used Claude Code. The same applied to all 9 GitHub Copilot users, all 7 Antigravity users, and all 5 Pi users.

2.4 Model Usage

Anthropic models were similarly concentrated. An Anthropic model was primary for 42 respondents (73.7%), and 51 (89.5%) used at least one named Anthropic model weekly.

Claude Opus 5 was the most common primary model with 20 respondents (35.1%), followed by Claude Sonnet 4.6 with 11 (19.3%) and Claude Sonnet 5 with 8 (14.0%). Weekly use was 27 respondents for Opus 5 (47.4%), 15 for Sonnet 5 (26.3%), and 13 for Sonnet 4.6 (22.8%). Figure 3 shows the full distribution of named models.

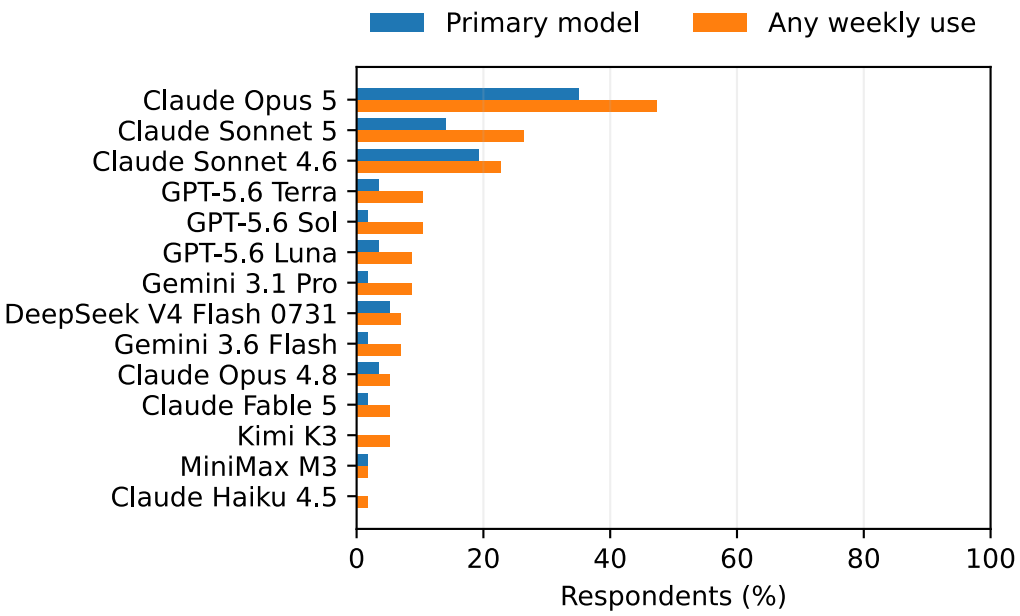

Figure 3: Primary and weekly usage of named models among survey respondents (n = 57).

Source: own analysis of the internal survey. Unnamed models are omitted.

Non-Anthropic models were common but rarely used alone. Of the 57 respondents, 27 (47.4%) reported only named Anthropic models, 24 (42.1%) combined Anthropic with another provider, and 5 (8.8%) used only non-Anthropic models. One response could not be classified by provider. Figure 4 shows the three classifiable groups.

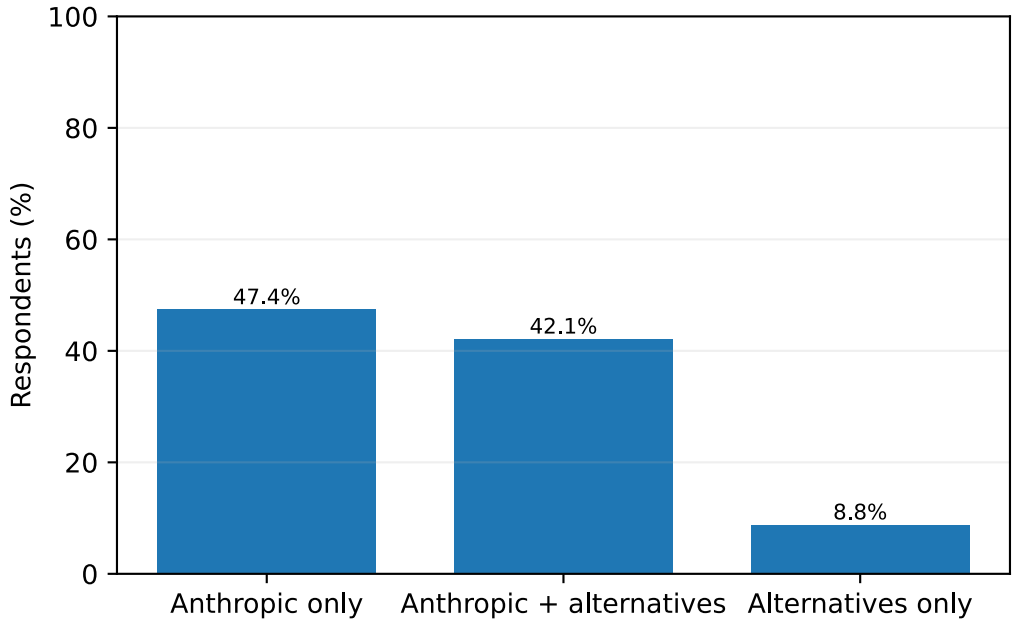

Figure 4: Anthropic-only and alternative-provider model use among survey respondents.

Source: own analysis of the internal survey. One respondent (1.8%) is omitted because no named model provider could be identified.

Among these respondents, alternative providers therefore mostly complement rather than replace Anthropic.

2.5 Model Cost Tiers

Named models were grouped by the average cost per task of their best-listed DeepSWE v1.1 configuration [1]. Costs below USD 2 are classified as low, USD 2 to below USD 5 as medium, and USD 5 or more as high. MiniMax M3 has no comparable DeepSWE result and is classified as low cost from its official API pricing [2]. Claude Haiku 4.5 also lacks a comparable DeepSWE result. It is manually assigned to the medium-cost tier based on its official API price of USD 1 per million input tokens and USD 5 per million output tokens [3]. This is a pricing-based fallback rather than a DeepSWE-derived classification. Unnamed models remain excluded.

The high-cost tier contains Claude Opus 5, Claude Sonnet 5, Claude Sonnet 4.6, Claude Fable 5, Claude Opus 4.8, and GPT-5.6 Sol. The medium-cost tier contains GPT-5.6 Terra, Gemini 3.1 Pro, Gemini 3.6 Flash, Kimi K3, and Claude Haiku 4.5. The low-cost tier contains GPT-5.6 Luna, DeepSeek V4 Flash 0731, and MiniMax M3.

Model cost differs by harness. In Figure 5, 55 of 57 classifiable Claude Code configurations (96.5%) use high-cost models and 2 (3.5%) use medium-cost models. OpenCode spans all three tiers: 40.0% high, 33.3% medium, and 26.7% low cost. Three of the four classifiable Pi configurations are low cost. Small samples for less common harnesses limit these comparisons.

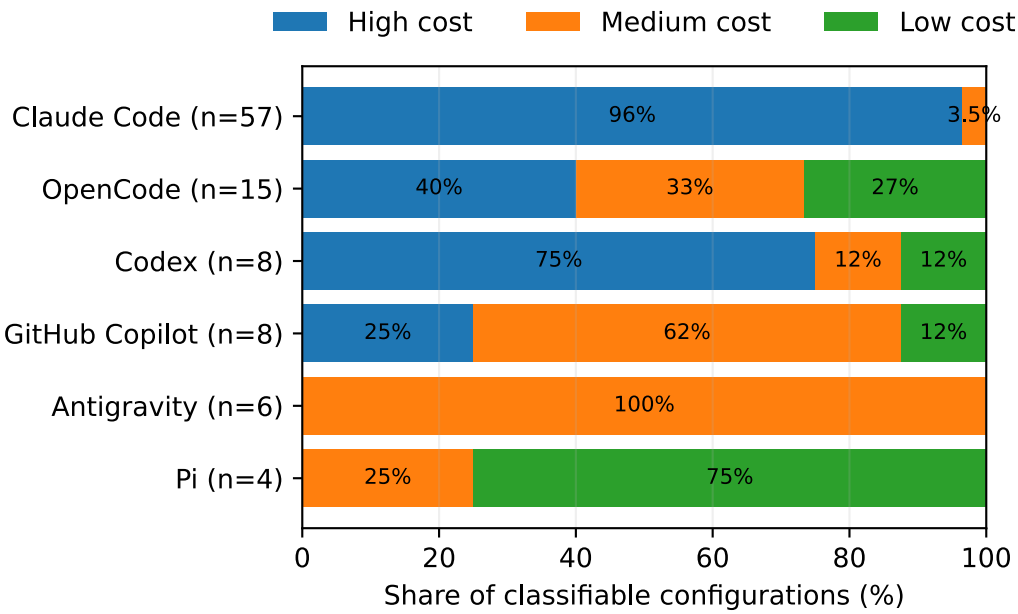

Figure 5: Model cost-tier distribution by harness for classifiable weekly configurations.

Source: own analysis of the internal survey. Tier boundaries use DeepSWE v1.1 average cost per task [1], with pricing-based fallbacks for MiniMax M3 [2] and Claude Haiku 4.5 [3].

Cost tier also differs between primary and secondary configurations. High-cost models account for 81.1% of classifiable primary configurations and 55.1% of secondary configurations. Medium-cost models rise from 7.5% of primary to 34.7% of secondary configurations, while low-cost models account for 11.3% and 10.2%, respectively (Figure 6).

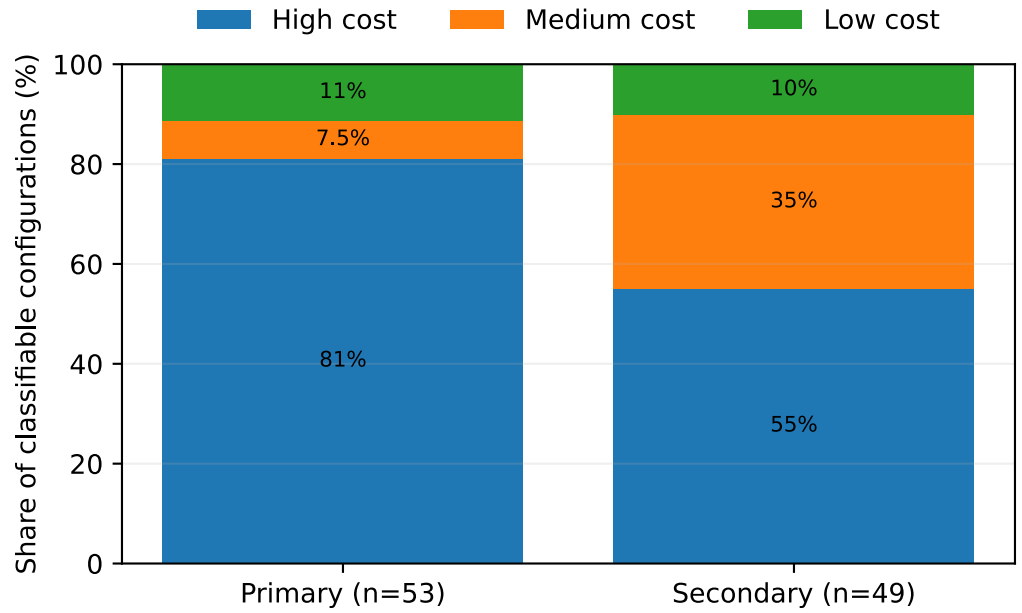

Figure 6: Model cost-tier distribution for primary and secondary survey configurations.

Source: own analysis of the internal survey. Primary and secondary bars contain 53 and 49 classifiable configurations, respectively.

The respondent-level data show the same association. Of the 17 respondents using at least one medium-cost model, 13 (76.5%) used multiple named providers and 13 (76.5%) used multiple harnesses. Among the 40 respondents without a medium-cost model, 11 (27.5%) used multiple named providers. This exploratory result suggests that medium-cost models are more common in diversified workflows.

2.6 Reported Cost

Monthly coding-agent cost was reported in categories rather than exact amounts. Of the 57 responses, 33 gave a monetary range, 15 selected a flat-rate subscription, 8 selected "I don't know", and 1 was blank. Among the 33 monetary responses, 22 (66.7%) reported at least EUR 100 per month. Figure 7 keeps the original categories instead of converting ranges to estimated values.

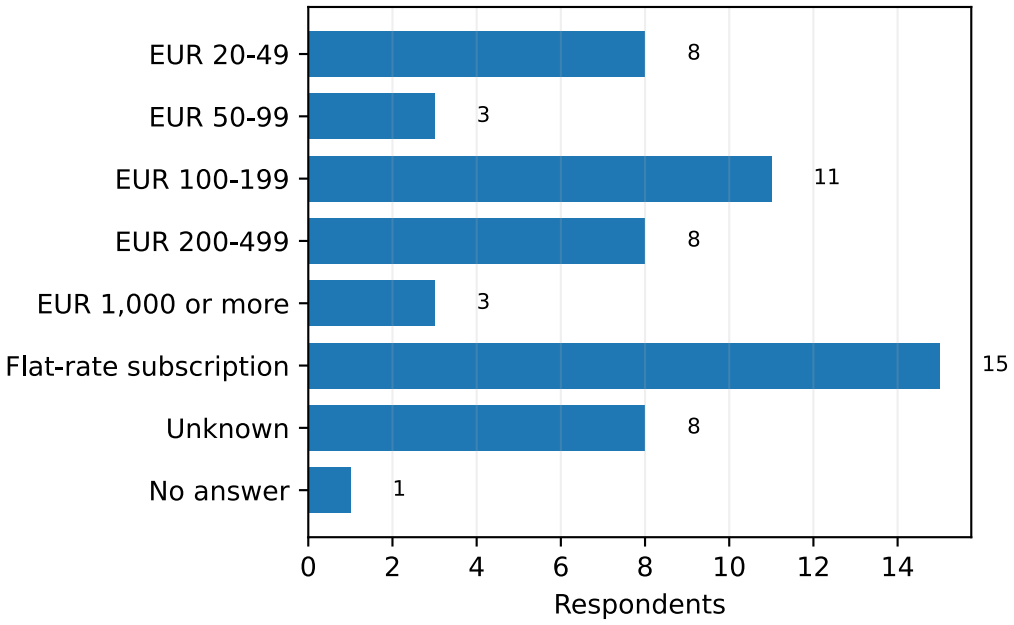

Figure 7: Reported monthly AI coding-agent cost categories among survey respondents (n = 57).

Source: own analysis of the internal survey.

The cost categories are not directly comparable as continuous values, so no average expenditure is derived.

Overall, the sample shows a strong Claude Code and Anthropic baseline with alternatives layered around it. High-cost models dominate primary configurations, while medium-cost models are more common in secondary and diversified setups. Because the survey targeted AI-active groups, these results characterize advanced internal users rather than company-wide adoption.

3 Methodology

This study evaluates coding-agent harnesses as part of a complete model–harness system. The central experimental principle is to preserve behavior that belongs to the harness while controlling external differences that would otherwise distort the comparison. Native capabilities such as delegation and provider-specific compaction therefore remain enabled where supported, whereas routing, context limits, model selection, and usage accounting are constrained when necessary to keep measurements comparable.

3.1 Benchmark and task selection

DeepSWE v1.1 was selected during planning in July–August 2026 because it was a current public benchmark explicitly designed to distinguish frontier coding agents on original, long-horizon software-engineering tasks. Its grading also fits this study: submissions are evaluated by isolated functional verifiers with a binary pass/fail reward rather than by an LLM judge [4]. This provides a reproducible success criterion without constructing and validating a new internal task set.

Running the complete benchmark across all model–harness combinations was outside the available budget. A deterministic subset of ten tasks was therefore selected from the public DeepSWE v1.1 results. For each programming language, one hard and one medium task was selected. Difficulty is derived from the solve rate of valid historical `mini-swe-agent` runs. The reference panel excludes Claude Opus 5, GPT-5.6 Luna, DeepSeek V4 Flash, and Kimi K3 because these were the four model configurations considered benchmark targets when the task sample was selected on 2026-08-18. Their own public DeepSWE results were therefore not allowed to influence the task difficulty or token-intensity values used for sampling. The final model matrix changed later; this exclusion set is part of the historical sampling procedure rather than a description of the final evaluated models. Hard tasks occupy the 75th–100th difficulty percentile, including exactly 75, while medium tasks occupy the 50th percentile up to but excluding 75. Within each language and difficulty stratum, the task with the highest median historical token usage is selected. This deliberately favors demanding tasks on which differences in context handling and token efficiency are more likely to become visible.

The selection was run on 2026-08-18 using the then-current public DeepSWE v1.1 API responses. `data/deepswe_task_selection_v1.1.json` records the resulting task IDs together with SHA-256 hashes of the `tasks.json`, `trials.json`, and `release.json` responses used at that time. The committed task IDs define the fixed task set for the experiment; the hashes provide provenance for the historical sampling inputs. Reproducing the benchmark therefore uses the committed task set rather than rerunning the selection against later API data. The complete selection procedure is implemented in `scripts/select_deepswe_tasks.py`.

3.2 Model and harness selection

The harness set consists of Claude Code, OpenAI Codex CLI, Pi, and OpenCode V2. Claude Code and Codex represent first-party coding agents from major model providers. OpenCode was the second-most common first-choice harness in the internal survey described in Sec-

tion 2, while Pi provides a comparatively lightweight alternative. OpenCode V2 was selected instead of V1 because it is the current major redesign and is more representative of the tool's future development.

The model set spans several providers and price classes: Claude Opus 5 at medium reasoning, and GPT-5.6 Luna, Kimi K3, DeepSeek V4.1 Flash, and GLM-5.3-Flash at max reasoning. The purpose is not to identify the strongest model in isolation, but to determine whether harness behavior remains effective across different model families.

3.3 Experimental controls

Each trial evaluates exactly one model through one harness. Harness-native delegation remains enabled because subagent orchestration is part of the behavior under study. However, delegated calls are constrained to the same benchmark model as the parent session. This prevents a nominally single-model trial from becoming a multi-model ensemble. The Pier adapters aggregate usage across the complete parent/child session tree while retaining the root session as the main trajectory.

Provider-specific harness optimizations are otherwise retained. For example, a harness may use its native compaction or caching behavior when the selected model supports it. Equivalent features are not recreated through custom plugins in competing harnesses, because the objective is to evaluate the harnesses as practical systems rather than force their internal implementations to be identical.

GPT-5.6 Luna supports a larger API context window, but the first-party Codex 0.151.0 model profile uses 272,000 tokens as its default context window and requires an explicit override to use more [5]. OpenAI also applies higher pricing when a request exceeds 272,000 input tokens [6]. The experiment therefore adopts the same 272,000-token operating window recommended by the first-party Codex default for every harness. Allowing only some harnesses to use the larger window would make cost depend on inconsistent context configuration and could move one harness into a different pricing tier.

External web-search capability is disabled in the benchmark configuration wherever the harness exposes a controllable search tool. This keeps trials limited to the repository, benchmark environment, and harness-native development tools rather than allowing external information retrieval.

3.3.a Model profiles

Where a harness ships a provider or model profile for a tested model, that profile is used as the base configuration. Only settings required to reach the designated inference endpoint are changed, such as the base URL, credentials, or gateway-facing model identifier. Harness-supplied compatibility settings, reasoning mappings, context and output limits, and other model-specific behavior are retained unless they conflict with an experimental control defined in this chapter. A custom profile is introduced only when the pinned harness version has no usable entry for the tested model or when the selected provider route requires narrowly scoped transport metadata.

Pi follows this policy for the tested open-weight models. Kimi K3 retains Pi 0.84.4's bundled Moonshot/Kimi metadata while only the provider endpoint and API key are changed. GLM-5.3-Flash retains the bundled profile except for two transport-compatibility fields. Pi's Z.AI request path can send `thinking` alongside `reasoning_effort`, and can add the Z.AI-specific `tool_stream` extension when tools are present [7]. The Fireworks-backed GLM route rejected these Z.AI-specific request forms during acceptance testing, so the benchmark uses `thinkingFormat: "openai"` and disables `zaiToolStream`. The reasoning level and model limits remain unchanged. OpenCode V2 follows the same profile policy: built-in catalogue entries are reused where possible, with explicit metadata only for missing models or required transport compatibility.

Codex 0.151.0 ships a native profile for GPT-5.6 Luna, so Luna uses that profile with only the endpoint and authentication path redirected [5]. The same Codex release does not ship native profiles for Claude Opus 5, Kimi K3, DeepSeek V4.1 Flash, or GLM-5.3-Flash. Unknown models instead receive a minimal fallback descriptor without configured multi-agent mode or tool mode, model-specific reasoning levels, and several other profile features available to recognized models [8]. Using this fallback would therefore evaluate those models with a substantially reduced Codex capability set.

For Claude Opus 5, Kimi K3, DeepSeek V4.1 Flash, and GLM-5.3-Flash, the experiment derives Codex metadata from the frozen GPT-5.6 Sol profile in Codex 0.151.0 and replaces the model-specific identity, context, modality, reasoning, and compatibility fields required for each route. These entries use Multi-Agent V1 and disable Responses Lite for the non-OpenAI routes while retaining the remaining Sol-derived behavior. This is a documented profile intervention rather than a neutral configuration choice: it is used so that these models are evaluated with the broader Codex agent feature set available to recognized models instead of the intentionally minimal unknown-model fallback. Codex results for these models are therefore conditional on this profile configuration.

3.3.b Model routing

Kimi K3, DeepSeek V4.1 Flash, and GLM-5.3-Flash are routed through AiOrbit, the internal LiteLLM proxy used by the organization, and are served through Fireworks behind this proxy. Claude Opus 5 calls the Anthropic API directly.

GPT-5.6 Luna is accessed through a ChatGPT subscription using the pinned CLIProxyAPI instance [9]. CLIProxyAPI stores the OpenAI OAuth credential centrally and exposes the same subscription-backed Luna route to Pi, Claude Code, Codex, and OpenCode V2. Trial containers receive only the local bridge endpoint and bridge credential; the OpenAI OAuth credential is not injected into Pier. This keeps the Luna provider route constant across harnesses while avoiding harness-specific subscription authentication.

AiOrbit, Fireworks, Anthropic, and CLIProxyAPI are infrastructure outside the experimental manipulation. They may transform requests or introduce provider-side failures that the study

cannot control. The results therefore characterize the harnesses under these documented deployment paths rather than an abstract provider-independent environment.

3.3.c Provider concurrency

Fireworks-backed jobs use model-specific concurrency caps so provider throttling does not become an experimental variable [10]. The Pi/Claude Code/Codex jobs run up to 30 Kimi K3 trials, eight GLM-5.3-Flash trials, and four DeepSeek V4.1 Flash trials concurrently. OpenCode V2 contains ten trials per model and runs ten Kimi or GLM trials concurrently, but five for DeepSeek. These settings affect job scheduling only; task definitions, model settings, scoring, and reported metrics are unchanged. The exact values are frozen in the benchmark configuration.

3.4 Benchmark infrastructure

The experiment does not use upstream Pier unchanged. It uses the project-specific `FZR-forks/pier` fork, which has diverged materially from the upstream runner used by DeepSWE [11], [12]. The fork became necessary because the selected model–harness matrix required additional adapters, stricter model isolation, and more complete usage accounting than upstream Pier exposed at the time of the experiment.

The fork adds Pi and OpenCode V2 support, complete delegated-session accounting for Claude Code and Codex, hard same-model enforcement for delegated calls, corrections for Codex cumulative token snapshots and inherited child history, cache-write-aware token and cost accounting, support for the internal CA required to reach AiOrbit, and preservation of failed retry-attempt evidence. Some Claude Code trajectory changes were selectively backported from Harbor; other changes are specific to this experiment. As a result, the PA1 runner is substantially more capable for this benchmark matrix than the upstream Pier checkout and must be treated as a distinct, versioned part of the experimental apparatus rather than as stock Pier.

These changes were kept outside the benchmark task itself. They affect agent installation, routing, model enforcement, trajectory conversion, and measurement, but do not modify the DeepSWE task description, repository, verifier, or reward calculation presented to the agent.

3.4.a OpenCode V2 integration

OpenCode V2 is a complete rewrite of V1 and exposes a different server and session API. The existing V1 adapter therefore cannot drive V2. PA1 implemented a separate `opencode-v2` Pier adapter for the benchmark [12]. The benchmark freezes OpenCode V2 at version 2.0.8 after validating the packaged binary and the new adapter together. PA1 keeps the release version, platform-specific archive checksums, frozen model catalogue, and provider configuration in the benchmark configuration instead of hardcoding them into Pier. This separation keeps the adapter generic across V2 releases.

The adapter starts a private authenticated OpenCode server for every trial and never shares that server between trials. After the agent finishes, the adapter waits for the session tree

to settle and collects paginated root, delegated, utility, and compaction-session records. It reconstructs recursive usage totals from these records without counting replayed messages twice. If collection remains incomplete, or if the server terminates unexpectedly, the adapter keeps the available trajectory and error evidence but does not report complete token and cost aggregates.

OpenCode's native delegation remains enabled because delegation is part of the harness behavior under test. OpenCode 2.0.8 allows a subagent to select any model exposed by the catalogue, so root-agent pinning alone does not enforce a single-model trial. PA1 therefore disables catalogue fetching, loads a frozen catalogue narrowed to the selected model and reasoning variant, and enforces that selection for root, delegated, utility, and compaction requests. This control preserves OpenCode's native multi-agent behavior while preventing a benchmark trial from silently using another model.

PA1 also keeps model identity separate from transport configuration. Luna keeps OpenCode's built-in OpenAI Responses profile so that provider-specific behavior such as native compaction remains available. Kimi K3, DeepSeek V4.1 Flash, and GLM-5.3-Flash keep their canonical OpenCode model identities, while the configuration redirects only the transport package, gateway endpoint, and provider-facing model identifier to the fixed Fireworks-backed route. GLM additionally marks Fireworks as the transport-level canonical provider because OpenCode's Z.AI transform otherwise adds request fields that the Fireworks route rejects. These transport changes solve compatibility problems without replacing the model with a synthetic provider profile.

Two additional controls keep OpenCode comparable with the other harnesses. First, OpenCode does not reliably project its configured output limit into OpenAI-compatible requests, so PA1 injects the model-specific output ceiling directly into the request body [13]. Second, PA1 applies the same context and reasoning policy used by the other harnesses: primary Kimi, DeepSeek, GLM, and Luna runs use the configured `max` reasoning variant, Opus uses `medium`, and Luna uses the common 272,000-token operating window. PA1 does not add a custom retry algorithm to OpenCode. OpenCode keeps its native request retry behavior, while Pier may repeat a complete trial only for transport or gateway failures.

PA1 validates every primary OpenCode configuration with the pinned real binary before admitting it to the benchmark. The acceptance path exercises both Chat Completions and Responses transports, explicit output caps, model restriction, usage reconciliation, native compaction, simultaneous-trial isolation, and a dedicated delegation smoke test. The smoke jobs use low reasoning and small output limits only to validate the integration; they do not change the primary benchmark settings.

3.5 Compatibility interventions

Compatibility changes are permitted only when they are required to make a harness communicate correctly with the intended model endpoint. Such changes are kept narrowly scoped and are not used to add capabilities to a harness.

3.5.a Luna through ChatGPT subscription

The benchmark uses CLIProxyAPI as a common access layer for GPT-5.6 Luna. Pi and OpenCode V2 use its OpenAI-compatible endpoint, Codex keeps its native OpenAI Responses profile while redirecting the endpoint, and Claude Code uses CLIProxyAPI's Anthropic-compatible translation. All four harnesses therefore reach the same ChatGPT-backed Luna account through the same pinned bridge implementation [9].

The OAuth state is stored in CLIProxyAPI's persistent authentication directory on the benchmark runner. Pier containers receive only a local bridge API key. Luna jobs are run serially so subscription-side concurrency does not become a harness-specific difference. The bridge's retries, credential cooldowns, and model fallbacks remain disabled.

3.5.b Codex with third-party models

Codex 0.151.0 uses the OpenAI Responses protocol, while the Fireworks-backed models are reached through AiOrbit using Chat Completions. Direct translation through the LiteLLM gateway was not sufficient for Codex's request and reasoning semantics. A pinned CLIProxyAPI instance is therefore used as a protocol bridge:

`Codex -> Responses -> CLIProxyAPI -> Chat Completions -> AiOrbit -> Fireworks`.

The bridge performs protocol translation only. Its own retries, credential cooldowns, and model fallbacks are disabled so it cannot improve the measured harness behavior.

During the benchmark period, an upstream CLIProxyAPI defect was identified in which a streamed chunk containing both `reasoning_content` and visible `content` could produce an invalid Responses item order [14]. Rather than upgrade the complete bridge and introduce unrelated changes, the benchmark keeps the frozen v7.2.146 base and backports only the upstream fix [15]. Logs from the completed 2026-08-31 Kimi K3 trial runs showed that the defect's triggering condition did not occur, so those runs were retained.

3.6 Cost accounting

Cross-harness cost is calculated from recorded token and cache usage using a frozen pricing table in `benchmark/pricing.yaml`. Official model API prices are used rather than the invoice price of the underlying proxy, inference host, or subscription. GPT-5.6 Luna is executed through a flat-rate ChatGPT subscription, so its reported dollar cost is an API-equivalent normalization derived from recorded token usage rather than the marginal amount charged for the benchmark run. This separates model and harness efficiency from provider-specific billing arrangements.

If a trial is discarded and repeated because of a transport or gateway failure, tokens consumed by the discarded attempt remain part of the reported cost. Removing them would systematically favor configurations that fail more often. In contrast, benchmark task timeouts are treated as outcomes of the evaluated system rather than infrastructure failures and are not automatically rerun.

3.7 Reproducibility

The selected DeepSWE task IDs are fixed in `data/deepswe_task_selection_v1.1.json`, while the runnable benchmark software is pinned separately. The Luna subscription route additionally depends on account-specific OAuth state that cannot be published; the repository instead fixes the CLIProxyAPI version, bridge configuration, and login procedure required to recreate the route with an authorized ChatGPT account. DeepSWE remains fixed at revision `0b9fabbb63b9104d678fe965e1632f2dd9eaa2ea`, with Codex CLI 0.151.0, Claude Code 2.1.251, Pi 0.84.4, and OpenCode V2 2.0.8. The current Pier checkpoint for every newly started run is `ac868ad54893db98143f6e9a2d8c783faeb64a61`, the merge commit containing the OpenCode V2 provider-URL hotfix [16]. OpenCode additionally pins the platform-specific package checksums and the frozen 2.0.8 model catalogue. The compatibility-bridge image digest, model profiles, pricing table, generated configurations, concurrency settings, and task-selection audit data are stored with the benchmark setup. Pier writes a `lock.json` into each result directory so that every run can be traced to the exact configuration used.

Three Pier checkpoints occur in retained primary data. The historical Kimi K3 Pi, Claude Code, and Codex runs use `ff65bae55c9a8ff15ddd3c2967c81a936713dd4d`. The DeepSeek V4.1 Flash Pi, Claude Code, and Codex runs use the immediately pre-hotfix checkpoint `7636cbee99ed947c64e0b350be031ec31e47dfee`. DeepSeek OpenCode V2 and every benchmark run started after the hotfix use `ac868ad54893db98143f6e9a2d8c783faeb64a61`. These differences are retained as explicit experimental provenance rather than hidden behind a single nominal runner version.

3.7.a DeepSeek V4.1 checkpoint transition

The DeepSeek V4.1 Flash runs for Pi, Claude Code, and Codex were already executed on Pier `7636cbee99ed947c64e0b350be031ec31e47dfee` when the OpenCode V2 gateway incompatibility was discovered. At that checkpoint, the OpenCode V2 adapter imposed an adapter-specific provider-URL policy that accepted HTTPS endpoints and explicit loopback HTTP endpoints but rejected the benchmark CLIProxyAPI route exposed to trial containers as `http://172.17.0.1/v1`. The other benchmark adapters did not impose this transport restriction.

Pier PR #15 removed only this OpenCode V2-specific URL validation and its associated imports/tests, leaving configured provider URLs to the same egress-host handling used by the other adapters [16]. The change does not modify DeepSWE tasks, model configuration, retry policy, usage accounting, or the Pi, Claude Code, or Codex adapters. It was therefore treated as a narrowly scoped compatibility hotfix rather than a new experimental condition.

The completed DeepSeek Pi, Claude Code, and Codex trials remain on `7636cbee99ed947c64e0b350be031ec31e47dfee`; rerunning them solely to align the repository SHA would consume additional benchmark budget without a code path through which this OpenCode V2-only patch could change their execution. DeepSeek OpenCode V2 instead uses the fixed merge checkpoint `ac868ad54893db98143f6e9a2d8c783faeb64a61`, and that

checkpoint becomes the frozen Pier revision for all subsequent new runs. This preserves one consistent pre-hotfix checkpoint for the already completed unaffected DeepSeek harnesses while enabling OpenCode V2 to reach the same CLIProxyAPI gateway used by the remaining benchmark setup.

3.8 Historical 2026-08-31 Kimi K3 batch

The Kimi K3 Pi, Claude Code, and Codex runs completed on 2026-08-31 are the earliest retained runner-checkpoint exception. They were completed before the final web-search policy and before the OpenCode V2 adapter was added, so they are documented separately rather than making their older setup appear to be the baseline for the rest of the experiment.

The first difference concerns the web-search tool surface. The 2026-08-31 Kimi configurations did not explicitly remove search-related tools. To preserve that tool surface, the later OpenCode V2 Kimi configuration also exposes OpenCode's native web-search tool to the model. This does not provide functional Internet search in the benchmark: the configured route does not provide a working search backend, so the tool cannot retrieve external web content. The exception therefore concerns the tool interface visible to the model, not actual web access. The retained 2026-08-31 logs also contain no completed Internet-backed search. All standard-wave model configurations disable web-search tools explicitly.

The second difference is the Pier revision. The 2026-08-31 runs used `ff65bae55c9a8ff15ddd3c2967c81a936713dd4d`. Later benchmark work first moved to `7636cbee99ed947c64e0b350be031ec31e47dfee`, which contains the independent OpenCode V2 adapter and subsequent OpenCode V2 reliability changes, and new runs now use the hotfixed `ac868ad54893db98143f6e9a2d8c783faeb64a61`. The changes after the Kimi checkpoint are either OpenCode V2-specific or shared logging behavior that does not alter the environment passed to Pi, Claude Code, or Codex. The historical Kimi runs use only those three harnesses, so the later Pier changes do not affect their task execution or model selection and should not affect their accounting. The experiment therefore retains the completed Kimi measurements instead of rerunning them only to align the runner commit.

4 Results and Analysis

> **Placeholder section:** Do not add results until the underlying data and analysis are available and documented.

4.1 Task success

- TODO: Report task-success results from the recorded data.

4.2 Cost

- TODO: Report cost results from the recorded data.

4.3 Token usage

- TODO: Report token-usage results from the recorded data.

4.4 Cache usage

- TODO: Report cache-usage results from the recorded data.

4.5 Runtime

- TODO: Report runtime results from the recorded data.

4.6 Comparative analysis

- TODO: Compare the results using the defined methodology and metrics.
- TODO: Add executable Quarto figure cells here once the benchmark result data is frozen. See `scripts/README-figures.md` for the Pareto-frontier and line-chart patterns.

5 Conclusion

Placeholder section: Add conclusions only after the research question has been addressed by the documented analysis.

5.1 Answer to research question

- TODO: Answer the research question based on the results.

5.2 Limitations

- TODO: Describe methodological and practical limitations.

5.3 Outlook

- TODO: Identify appropriate future work.

Bibliography

[1] DataCurve, "DeepSWE v1.1 Leaderboard." Accessed: Sep. 20, 2026. [Online]. Available: https://deepswe.datacurve.ai/

[2] MiniMax, "MiniMax API Token Plan." Accessed: Sep. 20, 2026. [Online]. Available: https://platform.minimax.io/subscribe/token-plan?tab=api-enterprise

[3] Anthropic, "Introducing Claude Haiku 4.5." Accessed: Sep. 21, 2026. [Online]. Available: https://www.anthropic.com/news/claude-haiku-4-5

[4] W. Huang, C. Lee, L. Tng, and S. Ge, "DeepSWE: Measuring Frontier Coding Agents on Original, Long-Horizon Engineering Tasks," *arXiv preprint arXiv:2607.07946*, 2026, Accessed: Sep. 15, 2026. [Online]. Available: https://arxiv.org/abs/2607.07946

[5] OpenAI, "Codex model catalog for release 0.151.0." Accessed: Sep. 15, 2026. [Online]. Available: https://github.com/openai/codex/blob/rust-v0.151.0/codex-rs/models-manager/models.json

[6] OpenAI, "GPT-5.6 Luna Model." Accessed: Sep. 15, 2026. [Online]. Available: https://developers.openai.com/api/docs/models/gpt-5.6-luna

[7] Pi contributors, “OpenAI Completions adapter in Pi 0.84.4.” Accessed: Sep. 20, 2026. [Online]. Available: https://github.com/earendil-works/pi/blob/v0.84.4/packages/ai/src/api/openai-completions.ts

[8] OpenAI, “Codex fallback model metadata for release 0.151.0.” Accessed: Sep. 15, 2026. [Online]. Available: https://github.com/openai/codex/blob/rust-v0.151.0/codex-rs/models-manager/src/model_info.rs

[9] FZR-forks, “CLIProxyAPI benchmark fork with OpenAI Codex OAuth and protocol translation.” Accessed: Sep. 21, 2026. [Online]. Available: https://github.com/FZR-forks/CLIProxyAPI/tree/2e7c81219be99590727ce34b21b94eac125a5111

[10] Fireworks AI, “Rate limits.” Accessed: Sep. 20, 2026. [Online]. Available: https://docs.fireworks.ai/serverless/rate-limits

[11] DataCurve, “Pier.” Accessed: Sep. 15, 2026. [Online]. Available: https://github.com/datacurve-ai/pier

[12] FZR-forks, “Pier: PA1 experimental fork.” Accessed: Sep. 21, 2026. [Online]. Available: https://github.com/FZR-forks/pier/commit/ac868ad54893db98143f6e9a2d8c783faeb64a61

[13] TheFerragamo, “v2 (opencode2): max_tokens never sent for @ai-sdk/openai-compatible models, limit.output ignored, thinking turns truncate at provider default.” Accessed: Sep. 15, 2026. [Online]. Available: https://github.com/anomalyco/opencode/issues/47398

[14] MAXOUXAX, “openai-compatibility: Codex shows a truncated assistant message when one chunk holds content and reasoning_content (deepseek-v4-flash).” Accessed: Sep. 14, 2026. [Online]. Available: https://github.com/router-for-me/CLIProxyAPI/issues/5659

[15] L. Pater, “fix(openai): process reasoning deltas before content in responses stream.” Accessed: Sep. 14, 2026. [Online]. Available: https://github.com/router-for-me/CLIProxyAPI/commit/c8ecb4f3c972664aae802e21c6a6743d5f6bd80a

[16] FZR-forks, “Relax OpenCode V2 provider URL restrictions.” Accessed: Sep. 21, 2026. [Online]. Available: https://github.com/FZR-forks/pier/pull/15