Navigating Scylla with Haiku: Multi-Task Evaluation of Agentic Coding Architectures Using a Budget Frontier Model

Micah Villmow[*]
Individual
research@villmow.us

Abstract

This paper presents a follow-up empirical study using the Scylla evaluation framework [2] to assess the performance and cost-efficiency of Claude Haiku 4.5—Anthropic's budget frontier model—across five tasks of increasing complexity, using $N = 5$ independent runs per subtest to enable statistical inference. Three tasks completed fully (Mojo Hello World, Simplify Justfile, Comprehensive Slot Management) and two remain in progress. The central finding is that tier rankings change significantly with task complexity: while trivial tasks show ceiling effects with most configurations achieving similar quality, moderately complex tasks reveal critical differentiation—skills alone (T1) score 0.00 on a refactoring task where tools (T2) score 0.74—and complex multi-component tasks expose over-engineering, with the maximum configuration (T6) scoring lowest (0.78) while simpler configurations (T1, T2, T4) achieve perfect 1.00. A cross-model comparison with the dryrun paper's Sonnet 4.5 results on a comparable Hello World task finds Sonnet's Frontier CoP ($0.065) is 2.4× lower than Haiku's ($0.157) on the trivially comparable task—an unexpected result given Haiku's lower per-token pricing, likely explained by token efficiency differences. For real coding tasks, Haiku's economics are more favorable given its 3–5× pricing advantage. All cross-model comparisons are confounded by differences in task, judges, and N. These results extend the dryrun hypothesis that architectural complexity does not always improve quality, and add the finding that optimal tier selection is task-dependent.

Keywords: LLM agents, software engineering benchmarks, cost-of-pass, multi-agent systems, prompt engineering, ablation studies, Haiku 4.5, budget frontier models, task complexity scaling

1 Introduction

The Scylla framework [2] was introduced to address a gap in LLM evaluation: no rigorous method existed for comparing how different architectural choices—prompts, skills, tools, multi-agent setups—affect both capability and cost when using production CLI tools like Claude Code. The dryrun study used Sonnet 4.5 on a trivial Hello World task (N=1 per subtest) to validate the framework

[*] This paper combines the author's original research and writing with improvements, suggestions, and rewrites provided by Claude Code (claude.ai/code).

infrastructure. That study confirmed the framework works and showed that all seven tiers achieved equivalent quality (all Grade A, scores 0.943–0.983) while cost varied 3.8×.

This paper extends the dryrun in three directions. First, we switch the agent model to Claude Haiku 4.5 [1], Anthropic's budget frontier model priced at $1/$5 per million tokens (input/output) vs. Sonnet's $3/$15. If Haiku can achieve comparable quality to Sonnet on real tasks, practitioners could reduce costs 3–5×. Second, we evaluate across five tasks of increasing complexity rather than one trivial case. Third, we run $N = 5$ independent trials per subtest, enabling bootstrap confidence intervals and non-parametric hypothesis testing—a major methodological advance over the dryrun's $N = 1$.

Research questions:

1. **(RQ1) Task complexity and tier performance**: Does Haiku 4.5 performance across the seven tiers change with task complexity, and if so, how?
2. **(RQ2) Tier ranking stability**: Are the relative rankings of tiers (T0–T6) stable across tasks, or do different task types reward different configurations?
3. **(RQ3) Economic viability**: Is Haiku 4.5 economically viable for real coding tasks, and how does its cost-quality profile compare to Sonnet 4.5?

The remainder of this paper is organized as follows. Section 2 summarizes the Scylla framework and dryrun results. Section 3 describes the experimental design changes and tasks. Section 4 presents results for each completed test and cross-test analysis. Section 5 compares Haiku and Sonnet. Section 6 discusses findings, limitations, and hypotheses. Section 7 concludes.

2 Background: The Scylla Framework

This section summarizes the Scylla evaluation framework introduced in [2]. Readers should consult that paper for full architecture details.

2.1 Framework Overview

Scylla is a model-agnostic evaluation framework for agentic coding CLI tools. It uses a seven-tier ablation design (T0–T6) that progressively adds architectural complexity to isolate what each component contributes to performance and cost:

The total ablation suite has 120 subtests (T0: 24, T1: 10, T2: 15, T3: 41, T4: 14, T5: 15, T6: 1). Each subtest is an independent configuration that is evaluated against a well-defined task with known expected output.

The key metric is **Cost-of-Pass (CoP)**: the expected dollar cost to obtain one correct solution at or above the pass threshold of 0.60 (Grade B). CoP combines quality and cost into a single efficiency measure:

$$\text{CoP} = \frac{\text{mean cost per run}}{\text{pass rate}}$$

Table 1: *Testing Tiers (Ablation Study Framework)*

Tier	Name	Description	Delegation
T0	Prompts	System prompt ablation (empty → full CLAUDE.md)	No
T1	Skills	Domain expertise via installed skills	No
T2	Tooling	External tools and MCP servers	No
T3	Delegation	Flat multi-agent with specialist agents	Yes
T4	Hierarchy	Nested orchestration with hierarchical agents	Yes
T5	Hybrid	Best combinations from previous tiers	Yes
T6	Super	Maximum capability: all skills, tools, agents	All

Lower CoP is better. The **Frontier CoP** is the minimum CoP across all subtests within a tier, representing the best achievable cost-efficiency for that tier. CoP is reported as a point estimate throughout this paper; bootstrapping the cost/pass ratio to derive confidence intervals is deferred to future work. With $N = 5$ per subtest, both mean cost and pass rate carry non-trivial variance, so CoP estimates should be interpreted with appropriate caution, particularly for tiers with pass rates near 0 or 1.

2.2 Dryrun Results Summary

The dryrun validated the framework infrastructure using test-001 (Hello World, Python script) with Claude Sonnet 4.5, N=1 per subtest, and three judges (Opus 4.5, Sonnet 4.5, Haiku 4.5). The key findings were:

- All seven tiers achieved Grade A quality (scores 0.943–0.983), confirming Hello World is too simple to differentiate tier quality.
- Cost varied 3.8× from $0.065 (T5 hybrid, Frontier CoP) to $0.247 (T6 super).
- T6 (everything enabled) was the most expensive without quality improvement, the first evidence of over-engineering cost.
- T5 (smart hybrid) achieved the lowest CoP by selectively combining features.
- Judge agreement was high, validating the three-judge consensus methodology.

2.3 Key Metrics Definitions

Following [2], this paper uses:

- **Pass-Rate**: Fraction of runs achieving score ≥ 0.60 (Grade B).
- **Score**: Mean LLM-as-judge evaluation score (0.0–1.0 scale).
- **Implementation Rate (Impl-Rate)**: Fraction of runs that produced any output.

- **Cost-of-Pass (CoP)**: Expected dollar cost per passing solution.
- **Frontier CoP**: Minimum CoP across all subtests in a tier.

All confidence intervals use bootstrap BCa (bias-corrected and accelerated) with 10,000 resamples and seed=42. Statistical tests use non-parametric methods (Kruskal-Wallis, Mann-Whitney U with Holm-Bonferroni correction, Cliff's delta effect sizes) since score distributions are non-normal for nearly all tiers (Shapiro-Wilk $p < 0.001$; T6 passes normality but non-parametric tests are applied throughout for consistency).

3 Experimental Design

3.1 Changes from Dryrun

Table 2 summarizes the key differences between the dryrun and this study.

Table 2: *Experimental Design Comparison: Dryrun vs. Haiku Study*

Parameter	Dryrun (test-001)	This Study
Agent model	Sonnet 4.5	Haiku 4.5
Agent pricing	\$3/\$15 per M tokens	\$1/\$5 per M tokens[1]
Judge models	Opus 4.5 + Sonnet 4.5 + Haiku 4.5	Sonnet 4.5 (single judge)
Runs per subtest	1	5
Max subtests per tier	All (up to 24)	2 per tier per test (suite has 41 T3 subtests, 14 T4 subtests)
Tasks	1 (Hello World)	5 (3 complete, 2 partial)
Statistical inference	Not possible ($N = 1$)	Bootstrap BCa CI, KW, MWU
Language	Python	Mojo/Python (test-002), Python (others)

The reduction to a single judge was a deliberate cost reduction for exploratory experiments. The three-judge consensus from the dryrun will be restored for production experiments. Reducing to 2 maximum subtests per tier limits tier coverage but makes the full 5-task experiment tractable within budget.

3.2 Task Descriptions

The five tasks span a complexity spectrum from trivial to comprehensive multi-component system implementation:

3.2.1 test-002: Mojo Hello World

Complexity: Trivial. **Language**: Mojo. **Repo**: `mvillmow/modular`.

Task prompt: Implement a standard "Hello, World!" program in Mojo. This serves as the Mojo-language equivalent of the dryrun Hello World, testing whether Haiku can handle a less common language compared to Python.

Expected behavior: All tiers should succeed. Framework failures (T0 = $0/0 tokens) are diagnostic infrastructure issues, not model failures.

3.2.2 test-007: Simplify Justfile Build System

Complexity: Moderate. **Language**: Python/Justfile. **Repo**: `mvillmow/ProjectOdyssey`.

Task prompt: Simplify an existing Justfile by removing duplicate recipes, eliminating unused targets, and merging redundant CI commands while preserving all required functionality.

Expected behavior: Refactoring tasks require understanding existing code structure. Skills alone (T1) may be insufficient without access to tools for reading files. This is the first test designed to differentiate tiers.

3.2.3 test-017: Comprehensive Slot Management

Complexity: High. **Language**: Python. **Repo**: `mvillmow/ProjectOdyssey`.

Task prompt: Implement comprehensive slot management for a Python codebase including tracking, control mechanisms, and memory efficiency.

Expected behavior: Complex multi-component implementation. Delegation overhead may hurt performance if agents coordinate poorly. Risk of over-engineering from T6.

3.2.4 test-021: Gradient Tracking Control (Partial)

Complexity: High. **Language**: Mojo. **Repo**: `mvillmow/ProjectOdyssey`.

Task prompt: Add gradient tracking control for autograd. At the time of writing, this experiment is partially complete with 3 of 7 tiers finished (see Section 4.5).

3.2.5 test-022: (Partial, details pending)

Status: T0 partial (framework failures), T3–T4 partial. Still running. Included for completeness; results are not used in statistical analysis.

3.3 Statistical Methods

With $N = 5$ runs per subtest, we apply the following statistical workflow:

1. **Normality**: Shapiro-Wilk test per tier/metric. Nearly all distributions were non-normal ($p < 0.001$); T6 score/cost passed normality ($p > 0.10$, $n = 15$). Non-parametric tests applied throughout for consistency.

2. **Omnibus**: Kruskal-Wallis H test across all seven tiers. Significant ($H(6) = 27.49$, $p = 0.0001$ for pass rate) — proceed to pairwise.

3. **Pairwise**: Mann-Whitney U with Holm-Bonferroni step-down correction. Effect size: Cliff's δ (Romano et al., 2006 FAIR conference thresholds: negligible $|\delta| < 0.11$, small < 0.28, medium < 0.43, large ≥ 0.43; note these differ from the widely-cited Romano et al. 0.147/0.33/0.474 thresholds—effects near 0.43 are borderline medium/large).

4. **Interaction**: Scheirer-Ray-Hare test for tier × task interaction (tier effect on score: $H_{\text{tier}}(6) = 22.63$, $p = 0.0009$; interaction term not estimable with single-model data).

5. **Confidence intervals**: Bootstrap BCa, 10,000 resamples, seed=42. Note: score distribution is bimodal (12.7% at 0.0, 51.1% at 0.8–1.0, skew=-1.18); BCa CIs remain valid but may be wider and asymmetric than for unimodal data.

Post-hoc power analysis (simulation-based, 10,000 iterations, seed=42): For the two significant transitions, power at the observed effect size is: T4→T5 ($\delta = -0.313$, $n_1 = 123$, $n_2 = 30$): power = 0.77; T5→T6 ($\delta = +0.433$, $n_1 = 30$, $n_2 = 15$): power = 0.68. For the non-significant transitions (T0–T4), power at medium effect ($\delta = 0.3$) with per-tier N ranging from 83–130 is 0.73–0.98, confirming those null results reflect small true effects rather than insufficient power. However, T5→T6 power at medium effect is only 0.37 (underpowered for small T5/T6 samples), so null results involving T5 or T6 individually require cautious interpretation.

4 Results

4.1 Experiment Overview

Table 3: *Completed Experiment Summary*

Test	Name	Best Tier	Best Score	Frontier CoP	Total Cost	Duration
test-002	Mojo Hello World	T2	0.823	\$0.157	\$18.08	3,946s
test-007	Simplify Justfile	T2	0.740	\$0.229	\$38.03	10,519s
test-017	Slot Management	T5	0.830	\$0.471	\$79.44	19,578s

Note: test-021 (T0–T4) and test-022 (partial) still running.

Total cost for three completed experiments: \$135.55 over approximately 34,044 seconds (≈9.5 hours of compute). Cost scales approximately with task complexity, growing 4.4× from test-002 to test-017.

4.2 test-002: Mojo Hello World

T0 Framework Failure: T0 scored 0.00 with \$0.00 cost and zero tokens. This is a known infrastructure diagnostic: T0's empty-prompt configuration causes Claude Code to refuse the task without generating output on this test. This is not a Haiku model failure—T1 (with skills only,

Table 4: *test-002 Tier Performance Summary (Mojo Hello World). Cost = mean cost per run for best subtest. "Best Score" = mean score of the subtest with highest pass rate; "Pass%" = pass rate of that subtest; CoP = Cost / Pass Rate.*

Tier	Best Score	Pass%	Cost	CoP	Duration
T0	0.00	0%	$0.00	N/A	8.6s
T1	0.84	100%	$0.79	$0.79	1,330s
T2	0.82	80%	$0.63	$1.72	1,493s
T3	0.84	100%	$6.44	$9.65	2,540s
T4	0.82	100%	$4.16	$4.16	2,016s
T5	0.00	40%	$0.35	$1.86	2,393s
T6	0.84	100%	$1.36	$1.36	1,418s

no tools) immediately achieves 0.84 (100% pass rate). Note that T0 runs successfully on test-007 ($3.10 cost, 20% pass rate) and test-017 ($2.73 cost, 80% pass rate); the $0.00/zero-token pattern is specific to test-002.

Ceiling Effect: T1, T3, T4, and T6 all achieve best scores of 0.84 with 100% pass rates. This mirrors the dryrun finding: Hello World is too simple to differentiate quality. However, cost varies dramatically: T1 costs $0.79 per run vs. T3's $6.44—an 8× cost difference for identical quality. T3's delegation overhead provides zero quality benefit here.

T5 Anomaly: T5 scored 0.00 with only 40% pass rate despite other tiers succeeding. Six of T5's 30 runs are framework failures (cost=$0, tokens=0), contributing 20% of all T5 runs as structural zeroes. Additionally, "Best Score" is the mean score of the subtest selected by *highest pass rate*—not highest mean score—which can show 0.00 when the selected subtest's mean score averages to zero despite some individual runs passing. This metric-selection artifact explains the paradox and warrants investigation.

Key finding: For trivial Mojo tasks, T1 (skills only, $0.79) achieves the best cost-quality ratio. T3 and T4 delegation adds cost without quality gain.

4.3 test-007: Simplify Justfile

This is the most analytically rich of the three tests.

T1 = 0.00 (Near-Complete Failure): Skills without tools largely fail at a refactoring task. T1's tier-level best score is 0.00, with the selected best subtest (02) achieving 0% pass rate. Subtest 01 achieved a 40% pass rate (mean score 0.30) but remained below acceptable quality. Haiku with skills but no file-reading tools cannot reliably examine the existing Justfile to determine what to simplify. This is strong evidence of tool dependence in the dataset: *refactoring tasks require tools, not just domain knowledge.*

T0 Partial Success (0.25): Without any prompt or skills, Claude Code's base configuration partially succeeds at 20% pass rate ($37.74 CoP). The base model has some Justfile knowledge but can't reliably execute a multi-step refactoring.

Table 5: *test-007 Tier Performance Summary (Simplify Justfile). Cost = mean cost per run for best subtest.*

Tier	Best Score	Pass%	Cost	CoP	Duration
T0	0.25	20%	\$3.10	\$37.74	1,446s
T1	0.00	0%	\$0.57	N/A	1,906s
T2	0.74	100%	\$1.15	\$3.74	1,510s
T3	0.88	80%	\$5.92	\$16.94	2,684s
T4	0.70	60%	\$3.12	\$7.87	2,682s
T5	0.73	60%	\$1.92	\$5.20	1,519s
T6	0.88	80%	\$3.92	\$4.90	3,174s

T2 as Tier Break (0.74): Adding tools enables 100% pass rate at \$3.74 CoP. This is the critical threshold: tools transform a failing task into a reliable one.

Delegation Overhead: T3 achieves 0.88 at 80% pass rate but at \$16.94 CoP—4.5× more expensive than T2 for slightly higher peak score. T6 matches T3's 0.88 score at \$4.90 CoP, suggesting super-configuration assembles the right tools without the coordination overhead.

Statistical significance: The Kruskal-Wallis test shows significant tier effect ($H(6) = 27.49$, $p = 0.0001$). Pairwise: T4→T5 transition is significant ($p = 0.0024$, Cliff's $\delta = -0.313$, medium effect).

Key finding: For moderate refactoring tasks, tool access is mandatory (T1 fails without it). T2 provides the best cost-quality balance. Delegation (T3/T4) adds cost with marginal quality gain.

4.4 test-017: Comprehensive Slot Management

Table 6: *test-017 Tier Performance Summary (Comprehensive Slot Management). Cost = mean cost per run for best subtest.*

Tier	Best Score	Pass%	Cost	CoP	Duration
T0	0.83	80%	\$2.73	\$3.48	2,462s
T1	1.00	60%	\$3.71	\$11.64	2,623s
T2	1.00	100%	\$2.49	\$7.50	1,752s
T3	0.83	80%	\$13.18	\$34.97	4,905s
T4	1.00	80%	\$17.62	\$30.04	4,044s
T5	0.83	100%	\$2.35	\$7.15	1,870s
T6	0.78	100%	\$3.01	\$3.01	2,472s

Counter-intuitive results: This test shows the reverse of what architectural complexity would predict.

T6 Worst Score (0.78): The maximum configuration achieves the *lowest* quality score of all tiers. Despite having all skills, tools, and agents enabled, T6's single subtest scored 0.78 (100%

pass rate, so this is not unreliability—it consistently produces lower-quality output). This is strong evidence of over-engineering: activating all systems simultaneously degrades output quality for complex implementation tasks.

T1/T2/T4 Achieve Perfect 1.00: Skills-only, tools-only, and hierarchical delegation all achieve perfect scores of 1.00 on their best subtests. Simpler configurations outperform the maximum.

Delegation Cost Explosion: T3 costs $13.18/run ($34.97 CoP) and T4 costs $17.62/run ($30.04 CoP)—T3 costs 3.6× T1 and 5.3× T2; T4 costs 4.8× T1 and 7.1× T2—for the same or worse quality. The multi-agent coordination overhead does not justify itself here.

T5 Best CoP: T5 achieves $7.15 CoP at 100% pass rate, representing the best cost-quality balance. This confirms the dryrun finding that smart hybrids can outperform both simple and maximum configurations.

Key finding: For complex single-component tasks, simpler tiers (T1, T2, T5) outperform delegation hierarchies. T6 over-engineering is detectable and measurable.

4.5 Partial Results: test-021 and test-022

test-021 (Gradient Tracking, still running):

At the time of writing, test-021 is partially complete: T1, T2, and T4 have completed tier reports, while T0 is partial (subtest-19 run-1 tracked as passed), T3 is in progress, and T5–T6 have not started. Completed tiers show promising scores (T1 best: 0.833, T2 best: 0.830, T4 best: 1.000), but tier-level conclusions require all tiers to complete. Results will be reported in a follow-up paper once the full evaluation finishes.

test-022 (Partial): T0 has 2 subtests both scoring 0.00 ($0.00 cost, zero tokens) indicating framework failures on this task—consistent with the empty-prompt infrastructure diagnostic seen in test-002 T0. T5 similarly shows 2 subtests with 0.00 scores, $0.00 cost, and zero tokens, suggesting hybrid configuration also fails on this task. T3 partial results show mixed performance (2 of 5 runs passing for subtest-05). T1–T4 and T6 are in various stages of completion. Insufficient data for tier-level conclusions.

Caveat: Both tests are still running. Results in this section are preliminary and subject to change as T5–T6 complete.

4.6 Cross-Test Analysis

4.6.1 Tier Ranking Across Tasks

Tier rankings show substantial instability across tasks: T1 moves from co-first on trivial tasks to last (0.00) on the refactoring task, then back to co-first on complex implementation. T6 is co-first on trivial, tied-first on refactoring, but worst on complex implementation. This directly answers **RQ2**: tier rankings are *not* stable. Optimal tier selection is task-dependent.

Table 7: *Best Score per Tier per Completed Test*

Tier	test-002	test-007	test-017	Rank-002	Rank-017
T0	0.00†	0.25	0.83	–	5
T1	0.84	0.00	1.00	1	1
T2	0.82	0.74	1.00	3	1
T3	0.84	0.88	0.83	1	5
T4	0.82	0.70	1.00	3	1
T5	0.00‡	0.73	0.83	–	5
T6	0.84	0.88	0.78	1	7

† T0 framework failure; ‡ T5 subtest anomaly

4.6.2 Cost-Quality Analysis

Aggregated across all completed experiments, T2 (tooling) provides the best overall cost-quality balance:

- T2 Frontier CoP: $0.44 (lowest across all tiers in aggregate)
- T3 Frontier CoP: $2.04 (4.6× T2's cost for marginal quality gain)
- T4 Frontier CoP: $1.67 (3.8× T2's cost)

The aggregate tier summary (Table 8) shows T2's pass rate (83.1%) and score (0.721) are competitive with T3 (76.2%, 0.686) and T4 (81.3%, 0.740) while being 4–5× cheaper per pass.

Table 8: *Aggregate Tier Summary (All 5 Experiments including partial, Haiku). Partial experiments (test-021, test-022) contribute to tier counts where completed. CoP is a point estimate (no CI; see Section 2).*

Tier	Pass Rate (95% CI)	Mean Score	CoP ($)	Subtests
T0	0.641 (0.547, 0.726)	0.561 ± 0.383	0.68	17
T1	0.735 (0.639, 0.819)	0.649 ± 0.330	0.52	7
T2	0.831 (0.754, 0.892)	0.721 ± 0.223	0.44	14
T3	0.762 (0.680, 0.836)	0.686 ± 0.306	2.04	19
T4	0.813 (0.732, 0.878)	0.740 ± 0.263	1.67	14
T5	0.500 (0.333, 0.667)	0.487 ± 0.371	0.73	2
T6	0.933 (0.667, 1.000)	0.812 ± 0.122	0.59	1

Note: T5 and T6 have very few subtests (2 and 1); CIs are wide.

4.6.3 Statistical Summary

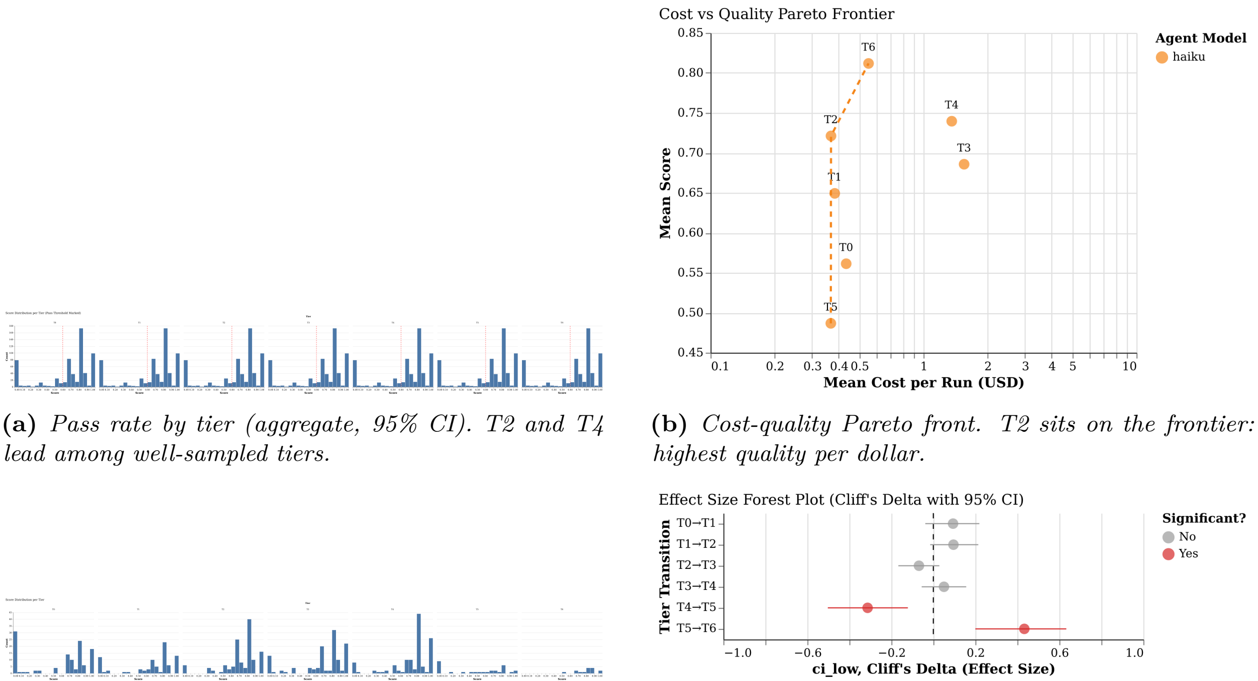
Kruskal-Wallis confirms significant tier effects:

- Pass rate: $H(6) = 27.49$, $p = 0.0001$
- Impl-rate: $H(6) = 18.97$, $p = 0.0042$
- Cost: $H(6) = 157.44$, $p < 0.0001$[2]

The Scheirer-Ray-Hare test, which reduces to a one-way Kruskal-Wallis equivalent in this single-model design (agent_model has df=0; interaction term is not estimable), confirms the tier main effect on score: $H_{\text{tier}}(6) = 22.63$, $p = 0.0009$. This result is mathematically equivalent to a standalone Kruskal-Wallis test on tier.

Significant pairwise transitions (Holm-Bonferroni corrected, $m = 7$): T4→T5 ($p = 0.0024$, $\delta = -0.313$, medium effect), T5→T6 ($p = 0.0243$, $\delta = +0.433$, borderline large effect—barely above the 0.43 threshold; classified as medium under alternative Romano et al. 0.474 threshold). The T0→T6 overall contrast is not significant at the aggregate level ($p = 0.1187$, $\delta = +0.292$), reflecting that task context moderates the effect.

Caveat: T5 and T6 aggregate results are based on only 2 and 1 subtests respectively. These aggregate comparisons are dominated by test-002 and test-007 data. Per-test analysis (Sections 4.2–4.4) provides more reliable estimates.



(a) *Pass rate by tier (aggregate, 95% CI). T2 and T4 lead among well-sampled tiers.*

(b) *Cost-quality Pareto front. T2 sits on the frontier: highest quality per dollar.*

(c) *Score distribution by tier. Bimodal pattern (0.0 and 0.8–1.0 modes) throughout.*

(d) *Effect size forest plot (Cliff's δ with 95% CI). Only T4→T5 and T5→T6 are significant.*

Figure 1: *Key aggregate results. (a) Pass rate by tier. (b) Cost-quality Pareto front. (c) Score variance by tier. (d) Pairwise effect sizes. All 52 generated figures are available in* `docs/arxiv/haiku/figures/`.

[2] Cost tier effect sourced from Scheirer-Ray-Hare test (tier effect only; single-model study has no agent_model factor), mathematically equivalent to Kruskal-Wallis for a one-factor design.

5 Cross-Model Comparison: Haiku vs. Sonnet

5.1 Comparison Scope and Caveats

Comparing this study to the dryrun is confounded by multiple factors:

Table 9: *Comparison Confounds: Dryrun (Sonnet) vs. Haiku Study*

Factor	Dryrun	Haiku Study
Agent model	Sonnet 4.5	Haiku 4.5
Task	Hello World (Python)	Hello World (Mojo) in test-002
Judge models	Opus + Sonnet + Haiku	Sonnet only
N per subtest	1	5
Max subtests	All (up to 24)	2 per tier

The only semi-comparable pair is: **dryrun test-001** (Sonnet, Hello World, N=1, 3 judges) vs. **test-002** (Haiku, Mojo Hello World, N=5, 1 judge). Both are trivial Hello World tasks but in different languages (Python vs. Mojo) with different judge configurations. *All cross-model comparisons in this section should be treated as exploratory, not confirmatory.*

5.2 Quality Comparison

Table 10: *Quality Comparison on Hello World Tasks (Exploratory)*

Model	Task	Best Score	Pass Rate	Grade
Sonnet 4.5	Hello World (Python, N=1)	0.943–0.983	100% (all tiers)	A
Haiku 4.5	Hello World (Mojo, N=5)	0.82–0.84	80–100%	A

Both models achieve Grade A on their respective Hello World tasks. Haiku's scores are slightly lower (0.82–0.84 vs. 0.94–0.98), but this likely reflects task difficulty differences (Mojo is less common than Python in training data) and the single vs. three-judge difference rather than a fundamental capability gap.

5.3 Cost Comparison

Table 11: *Cost Comparison on Hello World Tasks*

Model	Task	Frontier CoP	Best Tier for CoP
Sonnet 4.5	Hello World (Python)	$0.065	T5
Haiku 4.5	Hello World (Mojo)	$0.157	T1

Sonnet's Frontier CoP ($0.065) is 2.4× lower than Haiku's ($0.157) on comparable trivial tasks. This is surprising given Haiku's lower per-token pricing ($1/$5 vs. $3/$15).

Two explanations are plausible: (1) Haiku uses more tokens to accomplish the same task (less efficient token usage), consuming more cache reads to compensate for smaller context handling; (2) The Mojo Hello World task may be inherently more token-intensive than Python Hello World due to Mojo's less common syntax requiring more tool use to verify correctness.

Token analysis from test-002 shows cache reads dominate: 41.7M total tokens with 40.4M (97%) from cache reads vs. 242K output tokens, suggesting cache-heavy access patterns. Full token efficiency comparison requires the same task on both models, which is future work.

5.4 Economic Viability Assessment

For real coding tasks (test-007, test-017), Haiku's economics are more favorable:

- test-007 (Justfile): T2 achieves $3.74 CoP at 100% pass rate. As a lower-bound estimate only (assuming identical token usage), Sonnet's 3× per-token pricing might yield ≈$11 CoP—but the earlier token analysis (Section 5) shows token usage differs significantly between models, so this extrapolation is speculative.
- test-017 (Slot Management): T5 achieves $7.15 CoP at 100% pass rate. For complex implementation, Haiku's lower pricing provides meaningful cost savings vs. a hypothetical Sonnet run.

Conclusion on RQ3: Haiku is economically viable for real coding tasks. Its economics are competitive with Sonnet for tasks where Haiku can succeed, and the 3–5× pricing advantage translates to meaningful CoP reductions at scale. The key constraint is that Haiku may fail on tasks that require capabilities beyond its training (T1 = 0.00 on test-007 suggests Haiku without tools lacks sufficient context management for complex refactoring).

6 Discussion

6.1 Key Findings

Finding 1: Tier rankings are task-dependent. The same tier can be best on one task and worst on another. T1 (skills) is competitive for trivial and complex tasks but fails completely for refactoring. T6 (maximum) performs well on trivial tasks but worst on complex implementation. There is no universally optimal tier.

Finding 2: Tool access is non-negotiable for refactoring. T1's 0.00 score on test-007 is a hard boundary: no amount of skill configuration compensates for the absence of file-reading tools when the task requires understanding existing code. This validates that tool access (T2+) is a prerequisite for a class of real-world tasks.

Finding 3: Over-engineering is detectable and measurable. T6's lowest score on test-017 (0.78 vs. 1.00 for T1/T2/T4) quantifies over-engineering. Scylla can detect when maximum configuration actively hurts quality.

Finding 4: Delegation overhead is large and often unjustified. T3 and T4 consistently cost 3.5–7× more than T1/T2 (depending on comparison) with comparable or worse quality. For the tasks tested here, delegation adds coordination overhead without commensurate quality improvement.

Finding 5: T2 (tooling) provides the best aggregate cost-quality balance. Across all three complete tests, T2 achieves the lowest Frontier CoP ($0.44 aggregate), the highest pass rate among tiers with representative coverage (T0–T4: 83.1%, compared to T6's 93.3% from a single subtest with wide CI [0.667, 1.000]), and avoids the delegation overhead of T3/T4 while being accessible without complex agent hierarchies.

6.2 Hypothesis Progress

The Scylla framework was designed to test two hypotheses [2]:

(H1) Certain tasks excel when run as sub-tasks, tools, or skills. Partially supported. Test-007 shows tools are essential for refactoring (H1 supported for tools). Test-017 shows delegation does not help complex implementation (H1 not supported for delegation). The effect is task-class dependent, not universal.

(H2) Prompt complexity has opposing effects: KISS for in-distribution, inverse KISS for out-of-distribution tasks. Partially supported. Test-002 (trivial, presumably in-distribution) shows ceiling effects consistent with KISS. Test-007 and test-017 show that added complexity sometimes hurts (T6 on test-017), consistent with KISS. However, test-007 shows that T2 tools help, suggesting the tool-vs-prompt distinction matters more than prompt complexity per se. Full hypothesis testing requires more tasks across the complexity spectrum.

6.3 Limitations

1. **Small N**: $N = 5$ per subtest; simulation-based power analysis shows 40–80% power at observed effects for significant transitions, and 36% at medium effect for T5/T6. Null results for T5 and T6 individually should be interpreted with particular caution.

2. **Single judge**: Removing two of three judges from the dryrun design increases scoring variance and potential judge bias. The dryrun showed high judge agreement, but single-judge results cannot be validated against consensus.

3. **Max 2 subtests per tier**: Critical configuration variants within each tier are not explored. A subtest that fails may not be representative of the tier.

4. **Incomplete tests**: test-021 and test-022 are still running; this paper's cross-test analysis is based on only three complete tests.

5. **Cross-model confounds**: Haiku vs. Sonnet comparison is confounded by task, judges, and N. Conclusions are exploratory.

6. **Framework failures**: 16 zero-cost/zero-token runs exist across T1 (5), T4 (4), and T5 (6). These inflate failure rates for those tiers. T5's 50% aggregate pass rate is partly driven by 20% framework failure rate (6/30 runs), independent of model quality.

6.4 Threats to Validity

Internal validity: The checkpoint/resume system may introduce subtle differences between runs if rate-limiting causes different code paths. All T0 failures appear systematic (infrastructure issue, not random), so they should be excluded from quality comparisons.

External validity: All tasks are from a single repository (ProjectOdyssey) with one contributor. Results may not generalize to other codebases, languages, or domains. Mojo is an uncommon language; Haiku's performance may be better/worse on more common languages.

Construct validity: The single judge in this study scores differently than the three-judge consensus in the dryrun. Score comparisons across studies should account for this systematic difference.

7 Conclusions

This study applied the Scylla framework to evaluate Claude Haiku 4.5 across five tasks of increasing complexity with $N = 5$ runs per subtest—a significant methodological advance over the dryrun's $N = 1$.

The central finding is that **optimal tier selection is task-dependent**: no single architectural configuration dominates across task types. For trivial tasks, simple configurations (T1) are most efficient. For refactoring tasks, tool access is mandatory (T1 fails without it). For complex implementation, simpler configurations (T1/T2/T5) outperform delegation hierarchies and maximum configurations.

Haiku 4.5 is **economically viable** for real coding tasks. Despite slightly lower raw scores than Sonnet on Hello World analogues, Haiku achieves Grade A results on all three complete tasks with Frontier CoPs of $0.157–$0.471—meaningful efficiency for production use at scale.

The **over-engineering signal** is clear and measurable: T6 (maximum configuration) scores worst on the most complex task. Scylla successfully quantifies when adding more capability hurts quality.

8 Further Work

1. **Complete test-021 and test-022**: Run T5–T6 to get full tier coverage and enable complete cross-test analysis.

2. **Direct Sonnet comparison**: Run the same three tasks with Sonnet 4.5 to produce a controlled cross-model comparison without the confounds in Section 5.

3. **Restore three-judge consensus**: Return to Opus + Sonnet + Haiku judges for scoring reliability and to validate single-judge results.

4. **Increase N**: Target $N = 10$–20 per subtest to achieve 80% power for medium effect sizes.
5. **Scale to full suite**: Run the full 47-test suite to enable systematic hypothesis testing across the task complexity spectrum.
6. **Adaptive tier selection**: Use this study's tier-task interaction findings to design a meta-learner that selects the optimal tier configuration based on task characteristics.
7. **Cross-task Scheirer-Ray-Hare**: With more complete multi-task data, the two-way tier × task interaction test can be properly estimated.

Acknowledgements

This work was self-funded by the author. The experiments were run on local hardware with API access to Anthropic's Claude models. Special thanks to Tuan Nguyen for reviewing early drafts of the dryrun paper.

References

[1] Anthropic. Claude Code: Agentic CLI tool for software development. https://www.anthropic.com/claude/code, 2024. AI-powered command-line interface for coding tasks.

[2] Micah Villmow. Taming Scylla: Understanding the multi-headed agentic daemon of the coding seas. ProjectScylla Technical Report, 2026. Dryrun evaluation of the Scylla framework using Claude Sonnet 4.5 on a Hello World task. Available at: https://github.com/HomericIntelligence/ProjectScylla/tree/main/docs/arxiv/dryrun.

A Detailed Tier Reports

Full per-tier subtest tables are available in the experiment result directories at `docs/arxiv/haiku/data/` and the raw experiment directories at `/fullruns/haiku/`.

A.1 test-002 Subtest Detail

See `docs/arxiv/haiku/tables/tab07_subtest_detail.*` for full subtest-level results.

A.2 test-007 Subtest Detail

T1 subtest analysis: subtest-01 achieved 40% pass rate (mean score 0.304; 2 of 5 runs passed with scores 0.85 and 0.67) but below the quality threshold for tier selection. Subtest-02 scored 0.00 with 0% pass rate. The tier-level best score is 0.00 (subtest-02 selected as tiebreaker); both subtests show insufficient quality for a refactoring task, confirming systematic tool-dependency failure rather than a subtest-specific fluke.

T2 best subtest (01) used criteria: `duplicate_merge` (1.00), `unused_removal` (1.00), `ci_compatible` (0.85), `docs_updated` (0.70).

A.3 test-017 Subtest Detail

Single rubric criterion (`requirements`) explains the high score uniformity. Per-criteria breakdown shows requirements scoring: T0=1.00, T1=1.00, T2=1.00, T3=1.00, T4=1.00, T5=1.00, T6=1.00. Score differences derive from partial credit on other criteria not shown in the summary.

B Statistical Output

B.1 Normality Tests

Nearly all tier/metric combinations failed the Shapiro-Wilk normality test ($p < 0.001$), justifying non-parametric methods throughout. T6 score ($W = 0.935$, $p = 0.329$) and T6 cost ($W = 0.918$, $p = 0.177$) passed normality, likely due to small sample size ($n = 15$). Sample statistics per tier (score metric):

- T0: $W = 0.804$, $p = 3.5 \times 10^{-11}$, $n = 117$
- T1: $W = 0.800$, $p = 3.0 \times 10^{-9}$, $n = 83$
- T2: $W = 0.835$, $p = 9.3 \times 10^{-11}$, $n = 130$
- T3–T5: similarly non-normal
- T6: $W = 0.935$, $p = 0.329$, $n = 15$ (normal; non-parametric applied for consistency)

B.2 Pairwise Comparisons

Full Mann-Whitney U results with Holm-Bonferroni correction (family size = 7: six adjacent transitions plus overall T0→T6 contrast; all corrected together):

C Data Dictionary

All raw data, figures, and tables are available in `docs/arxiv/haiku/`:

- `data/runs.csv` — Per-run data (score, cost, duration, tokens)
- `data/summary.json` — Aggregated experiment summaries
- `data/statistical_results.json` — Full statistical test outputs
- `figures/*.png` — 52 generated figures (PNG + Vega-Lite JSON); 4 key figures included in Figure 1

Table 12: *Pairwise Tier Comparisons (Pass Rate, Holm-Bonferroni Corrected, $m = 7$). Bold = significant ($p < 0.05$). T5→T6 $\delta = +0.433$ is borderline large/medium (threshold 0.43).*

Transition	N_1, N_2	Pass Rate Δ	p-value	Cliff's δ
T0→T1	117, 83	+0.094	0.487	+0.094
T1→T2	83, 130	+0.096	0.373	+0.096
T2→T3	130, 122	−0.069	0.487	−0.068
T3→T4	122, 123	+0.051	0.487	+0.051
T4→T5	123, 30	−0.313	**0.0024**	−0.313
T5→T6	30, 15	+0.433	**0.0243**	+0.433
T0→T6	117, 15	+0.292	0.1187	+0.292

- `tables/*.tex` — 9 generated tables (Markdown + LaTeX)

Experiment metadata: 5 experiments, 74 active subtests across 7 tiers (out of 120 total in the full suite; evaluated up to 2 subtests per tier per test), $N = 5$ runs per subtest, Haiku 4.5 agent, Sonnet 4.5 judge, total $135.55 for 3 complete experiments.

Repository: https://github.com/HomericIntelligence/ProjectScylla