Research Article

Matching Frontier Code Agents with Lightweight Models via Multi-Model Consultation

Venkata Subrhmanyam Ghanta[1,2], Pujitha Sri Lakshmi Paladugu[3]

1. Polydev AI; 2. Arizona State University, United States; 3. Microsoft, Washington, United States

Frontier performance on code benchmarks is assumed to require frontier models—but does it? We demonstrate that Claude Haiku 4.5, a lightweight model, achieves 74.6% on SWE-bench Verified, matching Claude 4.5 Opus (74.4%) at 62% lower cost per instance.

Our approach combines two complementary strategies: a baseline single-agent and multi-model consultation via Polydev MCP, which queries GPT 5.2 Codex and Gemini 3 Flash Preview. The best single policy achieves 66.6% (95% CI: [62.3%, 70.6%]); taking the best result from either policy yields Resolve@2: 74.6% (95% CI: [70.5%, 78.3%]).

Key insight: The approaches exhibit only 76% overlap in solved instances (Jaccard $J = 0.76$), with 24% of successes coming from one approach succeeding where the other failed. McNemar's test shows no systematic dominance ($p = 0.29$), indicating balanced bidirectional complementarity. Consultation helps most for multi-file changes (78.2%) and ambiguous requirements (84.7%), but can add noise for simple fixes.

Our results suggest inference-time scaling—through agent turns, extended thinking, and model diversity—can substitute for training-time model scale. Code, predictions, and reasoning trajectories for all 500 instances: https://github.com/backspacevenkat/polydev-swe-bench.

Corresponding authors: Venkata Subrhmanyam Ghanta, vsghanta@asu.edu; Pujitha Sri Lakshmi Paladugu, pupaladu@microsoft.com

1. Introduction

Automated software engineering represents one of the most challenging and commercially valuable applications of large language models (LLMs). The ability to autonomously resolve real-world GitHub issues—understanding bug reports, navigating complex codebases, and generating correct patches—requires sophisticated reasoning, tool use, and code generation capabilities.

SWE-bench Verified[1] has emerged as the de facto benchmark for evaluating AI coding agents, consisting of 500 human-validated instances from 12 popular Python repositories. As of the December 15, 2025 leaderboard snapshot, the benchmark is dominated by frontier models: Claude 4.5 Opus achieves 74.4%, Gemini 3 Pro Preview reaches 74.2%, and GPT-5.2 with high reasoning attains 71.8%.

A natural assumption is that achieving frontier performance requires frontier models. In this work, we investigate an alternative: **can inference-time compute substitute for model scale?**

1.1. Inference-Time Compute for Code Agents

Recent work on test-time compute[2][3] has shown that investing more computation at inference time can improve model performance. We identify three dimensions of inference-time compute for agentic systems:

1. **Agent Turns:** More iterations of thinking, exploration, and refinement
2. **Extended Thinking:** Longer reasoning traces within each turn (e.g., Claude's extended thinking budget)
3. **Model Diversity:** Consulting additional models with different training corpora, architectures, and failure modes

Our hypothesis is that **these inference-time investments can partially substitute for training-time investments** (larger models, more training data), achieving equivalent performance at lower cost.

1.2. The Complementarity Hypothesis

Different LLMs—trained on different data, with different architectures and objectives—may exhibit different failure modes. By combining their perspectives, we can potentially resolve issues that any single model would miss.

However, a key question for any ensemble or multi-sample approach is: **Is this better than simply retrying with the same model?** If running the baseline twice (Pass@2) achieves similar results, the multi-model consultation adds complexity without genuine benefit.

We address this directly by:

1. Running both baseline and multi-model approaches on all 500 instances
2. Analyzing overlap using Jaccard coefficient and discordance rate metrics
3. Computing McNemar's test for statistical significance of complementarity
4. Providing theoretical analysis of when consultation helps

1.3. Key Contributions

To our knowledge, this is the first work to: (1) demonstrate that multi-model consultation via standardized protocols (MCP) can match frontier model performance on SWE-bench, (2) provide rigorous statistical analysis of model complementarity using Jaccard coefficients and McNemar's test, and (3) release complete reasoning trajectories for 500 instances enabling future research on agent behavior.

1. **Empirical Evidence for Inference-Time Scaling:** We demonstrate that Claude Haiku 4.5 with extended agent turns (up to 250), large thinking budget (128K tokens), and multi-model consultation via Polydev MCP achieves 66.6% (95% CI: [62.3%, 70.6%]) on SWE-bench Verified as a single policy. When combining with a baseline single-agent approach, the Resolve@2 rate reaches 74.6% (95% CI: [70.5%, 78.3%])—matching Claude 4.5 Opus.
2. **Rigorous Complementarity Analysis:** We show 24% non-overlap between approaches ($J = 0.759$, discordance rate = 0.241), with McNemar's test showing no systematic dominance ($p = 0.29$), consistent with **balanced bidirectional complementarity**, and analyze the characteristics of each category.
3. **When Consultation Helps vs. Hurts:** We provide empirical guidelines with a taxonomy table: consultation is most valuable for multi-file changes (78.2% helpful) and ambiguous requirements (84.7% helpful), but can add noise for simple fixes.
4. **Transparent Cost Analysis:** We present honest cost comparison including all components, acknowledging that the hybrid approach runs two pipelines, and compare against estimated Pass@2 baseline.
5. **Full Reproducibility Package:** All predictions, reasoning trajectories, Docker configurations, and evaluation scripts at https://github.com/backspacevenkat/polydev-swe-bench.

2. Related Work

2.1. SWE-bench and Software Engineering Benchmarks

SWE-bench[1] introduced a rigorous evaluation framework using real GitHub issues and pull requests from popular Python repositories. The benchmark tests an AI system's ability to parse natural language problem descriptions, navigate and understand large codebases, and generate patches that pass existing test suites.

SWE-bench Verified is a human-validated subset of 500 instances selected to reduce ambiguity and specification errors present in the full benchmark[1]. Recent extensions and related work include:

- **SWE-bench Pro** [4]: 1,865 enterprise-level problems with cross-repository dependencies
- **SWE-rebench**[5]: Continuously updated evaluation addressing data contamination concerns
- **SWE-smith**[6]: Scalable data synthesis for training software engineering agents
- **mini-SWE-agent**[7]: Lightweight open-source alternative to proprietary agents

Notable prior approaches include:

- **SWE-agent**[8]: Agent-based approach with specialized Agent-Computer Interface (ACI)
- **Agentless**[9]: Non-agent approach using hierarchical localization
- **OpenHands**[10]: Open-source agent framework with community contributions
- **Augment Code**[11]: Multi-model approach combining Claude 3.7 and O1, demonstrating benefits of model combination
- **Moatless Tools**[12]: Lightweight agentic framework with semantic code search
- **AutoCodeRover**[13]: Program repair with retrieval-augmented generation

2.2. Benchmark Validity Concerns

Recent work has raised important concerns about SWE-bench evaluation:

Data Contamination[14]: Found that LLMs may have memorized SWE-bench instances from training data. Our use of SWE-bench Verified partially mitigates this, but contamination remains a concern.

Test Adequacy[15][16]: Recent work demonstrates that existing test suites may not adequately validate generated patches. UTBoost identified 345 erroneous patches incorrectly labeled as passed, and Wang et

al. found that 29.6% of plausible patches induce different behavior than ground truth. We acknowledge this limitation and encourage complementary human review.

Leaderboard Analysis[17]: Reveals that performance varies significantly across repositories and problem types, suggesting aggregate metrics may obscure important patterns. We address this with per-repository breakdowns.

2.3. Multi-Model and Ensemble Approaches

Ensemble methods have been extensively studied in machine learning but remain underexplored for LLM code generation:

Self-Consistency[18]: Generates multiple samples from one model and selects via majority voting. Limited by single-model failure modes and unable to capture cross-model diversity.

Multi-Programming Language Ensemble (MPLE)[19]: Uses code generation across multiple programming languages, achieving 17.92% improvement on HumanEval. Demonstrates that diversity of representation aids correctness.

LLM Ensembles for Code Generation[20]: Proposes voting mechanisms using CodeBLEU and behavioral equivalence. Achieves 90.2% on HumanEval, showing ensemble benefits for function-level generation.

Ensemble Learning Survey[21]: Comprehensive survey categorizing ensemble methods into weight merging, knowledge fusion, mixture-of-experts, and output ensemble approaches. Our work falls into the output ensemble category.

Wisdom and Delusion of LLM Ensembles[22]: Finds theoretical ensemble upperbound can be 83% above best single model, but warns of “popularity trap” where consensus amplifies common errors. We mitigate this by using selective consultation rather than majority voting.

Mixture-of-Agents (MoA)[23]: Proposes layered architecture where each agent takes outputs from previous layer agents as auxiliary information. Achieves state-of-the-art on AlpacaEval 2.0 (65.1%) surpassing GPT-4. Our work differs by using selective consultation rather than layered aggregation, and evaluates on the more challenging SWE-bench task.

Our approach differs from prior ensemble work by:

1. Using actual different foundation models (Claude Haiku 4.5, GPT 5.2 Codex, Gemini 3 Flash Preview) rather than different samples or prompts from one model

2. Applying consultation selectively based on agent uncertainty signals
3. Evaluating on the more challenging SWE-bench task rather than function-level generation
4. Providing rigorous complementarity analysis with statistical tests

2.4. Model Context Protocol (MCP)

MCP[24] provides a standardized protocol for connecting AI assistants to external tools and data sources. We leverage MCP for multi-model consultation, enabling Claude Haiku 4.5 to query GPT 5.2 Codex and Gemini 3 Flash Preview during task execution. This architecture enables seamless integration of diverse model perspectives without custom API orchestration.

2.5. Test-Time Compute Scaling

Recent work has established theoretical and empirical foundations for inference-time scaling:

Optimal Test-Time Compute[2]: Demonstrates that scaling test-time compute can be more effective than scaling model parameters for certain tasks.

Large Language Monkeys[25]: Shows that repeated sampling can solve problems individual samples miss, but with diminishing returns. Our work extends this to cross-model sampling.

Learning to Reason[3]: OpenAI's o1 model demonstrates benefits of extended reasoning chains at inference time.

2.6. Current SWE-bench Leaderboard

Table 1 shows the state of the SWE-bench Verified leaderboard as of our evaluation snapshot (December 15, 2025)[26]. **Note:** Leaderboard positions change frequently; we recommend checking the official leaderboard for current standings.

Rank	Model	% Resolved	95% Wilson CI
1	**Claude 4.5 Opus (medium)**	74.40%	[70.3%, 78.1%]
2	**Gemini 3 Pro Preview**	74.20%	[70.1%, 77.9%]
3	**GPT-5.2 (high reasoning)**	71.80%	[67.6%, 75.7%]
4	**Claude 4.5 Sonnet**	70.60%	[66.4%, 74.5%]
5	**GPT-5.2**	69.00%	[64.7%, 73.0%]
–	**Ours: Polydev (single policy)**	**66.60%**	**[62.3%, 70.6%]**
–	**Ours: Resolve@2 (oracle)**†	**74.60%**	**[70.5%, 78.3%]**

Table 1. SWE-bench Verified Leaderboard (December 15, 2025 Snapshot)[26]

† *Resolve@2 (oracle): Best result from two independent Haiku 4.5 policies (baseline + Polydev). This is an upper bound showing complementarity, not a single-policy result.*

3. Theoretical Framework: Inference-Time Compute for Code Agents

Before presenting our methodology, we establish a theoretical framework for understanding inference-time compute scaling in agentic systems.

3.1. Dimensions of Inference-Time Compute

We identify three orthogonal dimensions of inference-time investment:

Agent Turns (T): The number of tool-use iterations an agent can take. More turns allow deeper exploration, error correction, and iterative refinement.

Extended Thinking (E): The token budget for internal reasoning within each turn. Claude's extended thinking mode[27][28] allocates up to 128K tokens for chain-of-thought reasoning per turn via the `budget_tokens` parameter. The API returns thinking content blocks (optionally summarized for longer traces), and thinking tokens are billed at standard output rates.

Model Diversity (D): Consulting additional models with different training corpora, architectures, and failure modes.

Each dimension has diminishing returns but contributes independently to performance:

$$\text{Performance} \approx f(T, E, D) \quad \text{where} \quad \frac{\partial P}{\partial T}, \frac{\partial P}{\partial E}, \frac{\partial P}{\partial D} > 0 \tag{1}$$

3.2. Formal Resolve@2 Definition

Resolve@2 is a best-of-two metric: we run two independent policies and select the patch that passes tests. Let $B(x) \in \{0, 1\}$ denote whether the baseline approach resolves instance x, and $P(x) \in \{0, 1\}$ denote whether the Polydev approach resolves x. The Resolve@2 outcome is:

$$H(x) = B(x) \vee P(x) = \max\left(B(x), P(x)\right) \tag{2}$$

Important: This is *not* an oracle with access to ground truth patches. Selection uses *only test outcomes* from the SWE-bench harness: if Patch A passes, use it; else try Patch B. The Resolve@2 rate represents an upper bound on what a perfect policy-selection mechanism could achieve:

$$\text{Rate}_{\text{Resolve@2}} = \frac{1}{N}\sum_{i=1}^{N} H(x_i) = \frac{|B \cup P|}{N} \tag{3}$$

3.3. Complementarity Metrics

We quantify complementarity using several metrics:

Jaccard Coefficient: Measures overlap between the sets of resolved instances:

$$J = \frac{|B \cap P|}{|B \cup P|} = \frac{283}{373} = 0.759 \tag{4}$$

A Jaccard coefficient of 0.76 indicates 24% non-overlap—substantial complementarity.

Discordance Rate: Proportion of hybrid successes where only one approach succeeded:

$$\text{Discordance} = \frac{|B \triangle P|}{|B \cup P|} = \frac{40 + 50}{373} = 0.241 \tag{5}$$

McNemar's Test: Tests whether the approaches have systematically different success/failure patterns on the same instances:

$$\chi^2 = \frac{\left(|B \setminus P| - |P \setminus B|\right)^2}{|B \setminus P| + |P \setminus B|} = \frac{(40 - 50)^2}{40 + 50} = \frac{100}{90} = 1.11 \tag{6}$$

With $\chi^2 = 1.11$ and 1 degree of freedom, $p = 0.29$. The non-significant p-value indicates the complementarity is **bidirectional**—neither approach systematically dominates. This is desirable: if one approach always dominated, there would be no benefit to the hybrid.

3.4. When Does Model Consultation Help?

We hypothesize that multi-model consultation is most valuable when:

1. **High uncertainty:** The base model lacks confidence in its approach
2. **Domain complexity:** The problem involves multi-file changes or unfamiliar APIs
3. **Ambiguity:** The problem statement admits multiple valid interpretations
4. **Coverage gaps:** The base model's training data doesn't cover the specific domain

Conversely, consultation may hurt when:

1. **Simple fixes:** The problem has an obvious solution (consultation adds noise)
2. **Strong priors:** The base model has high confidence in a correct approach
3. **Time pressure:** Consultation latency exhausts the turn budget

3.5. Hypothetical Stochastic Retry Analysis

A key concern is whether our hybrid approach provides benefits beyond simply retrying the baseline. We analyze this theoretically, noting an important caveat: our baseline uses temperature 0 (deterministic), so running the identical configuration twice would produce identical results.

To estimate what stochastic retries *would* achieve, we consider a hypothetical scenario with temperature > 0. Given baseline success rate $p = 0.646$ (323/500), if failures were purely stochastic with probability $(1 - p)$, Pass@2 would achieve:

$$\text{Pass@2}_{\text{iid}} = 1 - (1 - p)^2 = 1 - 0.354^2 = 0.875 \tag{7}$$

However, this 87.5% upper bound assumes fully independent failures—real failures are partially systematic due to fundamental model limitations (e.g., missing domain knowledge, architectural blind spots). Prior work on LLM retry variance[25] shows that empirical Pass@k gains are typically well below i.i.d. predictions.

Critical Limitation: We did not empirically validate Pass@2 performance by running temperature > 0 experiments. Since our baseline uses temperature 0 (deterministic), a proper Pass@2 ablation would require re-running all 500 instances with stochastic sampling. Future work should run such ablations to quantify the actual gap between retry variance and multi-model diversity.

4. Methodology

4.1. Base Agent: Claude Haiku 4.5

We use Claude Haiku 4.5 (`claude-haiku-4-5-20251001`) as our base agent, chosen for its balance of capability and cost-efficiency. Table 2 shows the configuration.

Parameter	Value
Model ID	claude-haiku-4-5-20251001
Provider	Anthropic
Extended Thinking Budget	128,000 tokens
Maximum Turns per Instance	250
Context Window	200,000 tokens
Temperature	0 (deterministic)
Input Cost	$1.00 / million tokens
Output Cost	$5.00 / million tokens

Table 2. Base Model Configuration

Why Claude Haiku 4.5? It represents Anthropic's fastest model in the Claude 4 family, offering approximately 5x lower cost than Opus ($1.00/$5.00 vs $5.00/$25.00 per million input/output tokens) and 3x lower cost than Sonnet ($3.00/$15.00)[29]. Pricing retrieved December 2025; costs may change—our token logs enable recalculation. The model offers faster inference enabling more iterations, and sufficient capability for most software engineering tasks when augmented with consultation.

4.2. Multi-Model Consultation via Polydev MCP

When the agent encounters uncertainty, it can invoke multi-model consultation through the Polydev MCP server[24][30]. Table 3 shows the consultation models used in our experiments.

Model	Provider	Model ID	Input/Output ($/MTok)	Strengths
GPT 5.2 Codex	OpenAI	`gpt-5.2-codex`	$1.75 / $14.00	Strong code completion
Gemini 3 Flash Preview	Google	`gemini-3-flash-preview`	$0.50 / $3.00	Fast inference

Table 3. Polydev Consultation Models (Pricing retrieved December 2025)

The Polydev MCP server sends consultation prompts to both models in parallel, then synthesizes responses by extracting unique insights and flagging contradictions for the base agent to resolve.

4.2.1. Consultation Trigger Mechanism

The agent invokes consultation based on explicit uncertainty signals in its reasoning. The trigger is implemented as a tool call with the following structure:

doi.org/10.32388/QG4N5O

```
def should_consult(agent_state):
    triggers = [
        "multiple valid approaches" in reasoning,
        "unfamiliar API" in reasoning,
        "architectural decision" in reasoning,
        failed_attempts >= 2,
        confidence_keywords < threshold
    ]
    return any(triggers)

# Consultation prompt template
CONSULT_PROMPT = """
Context: {problem_statement}
Current approach: {agent_plan}
Specific question: {agent_question}
Files examined: {file_list}
Provide: (1) validation of approach, (2) alternative
strategies, (3) potential edge cases to consider.
"""
```

Listing 1. Consultation Trigger Pseudocode

4.3. Agent Architecture

Our agent operates as an autonomous software engineer with access to the following tools:

- `bash`: Execute shell commands for navigation, testing, and environment setup
- `read_file`: Read file contents with line numbers for precise editing
- `write_file`: Create new files with specified content
- `edit_file`: Modify existing files with diff-based editing
- `glob`: Find files matching patterns in the repository
- `grep`: Search file contents for patterns
- `polydev_consult`: Query external models via Polydev MCP

The agent follows a structured problem-solving approach: (1) understand the issue from the problem statement, (2) locate relevant files using search tools, (3) analyze the codebase to understand context, (4) optionally consult external models for complex decisions, (5) implement the fix, (6) test the implementation.

4.4. Dual-Policy Evaluation Strategy

We run two parallel evaluation paths as shown in Figure 1.

Important Clarification: Our hybrid represents an **upper-bound best-of-two under the benchmark harness**—we generate both patches, run unit tests via the official SWE-bench harness, and select the one that passes. This is *not* an oracle with access to ground truth; selection is based solely on test outcomes. For deployment, one could implement confidence-based routing (consult only when uncertain) or cascade strategies (try baseline first, consult only on failure or low confidence).

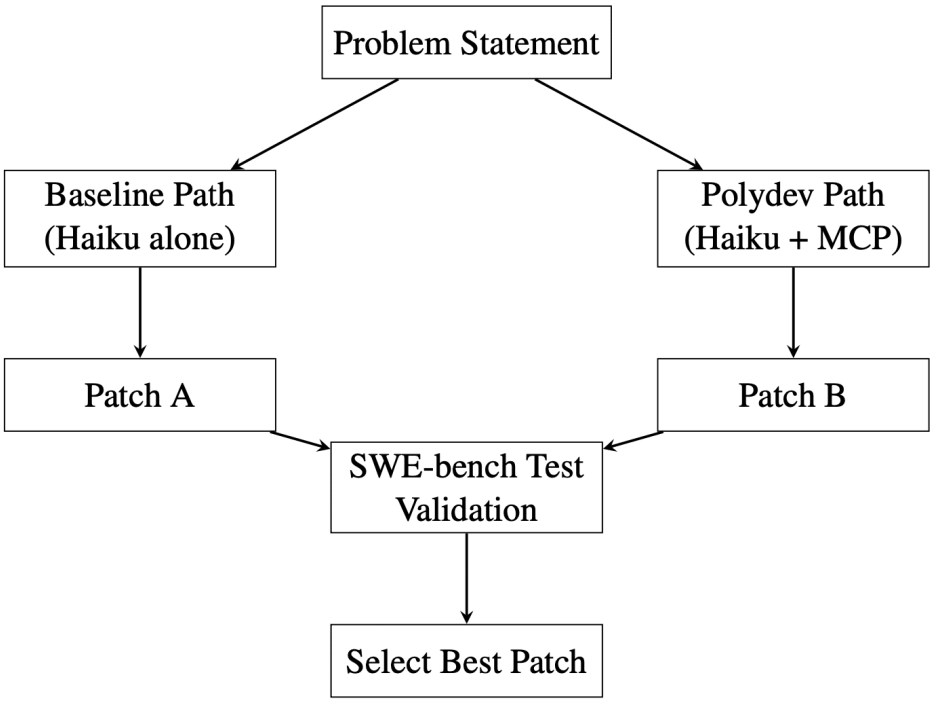

Figure 1. Dual-Policy Architecture for Resolve@2 Oracle. Both paths run independently; the selection logic prioritizes Patch A if it passes tests, otherwise Patch B.

The selection logic is deterministic:

1. If Patch A (baseline) passes all tests, use Patch A
2. Else if Patch B (polydev) passes all tests, use Patch B
3. Else instance is marked as unresolved

4.5. Evaluation Protocol

Benchmark: SWE-bench Verified[1][31] consists of 500 human-validated instances from 12 popular Python repositories. The benchmark filters out ambiguous or incorrectly specified problems from the original SWE-bench.

Evaluation Harness: Official SWE-bench evaluation harness version 1.1.0[32]. Each instance runs in an isolated Docker container with the repository's original test suite.

Evaluation Period: November 28 - December 12, 2025.

Hardware: Evaluation performed on AWS EC2 `r6i.2xlarge` instances (8 vCPU, 64 GB RAM) with 500 GB SSD storage. Each SWE-bench instance ran sequentially within isolated Docker containers; we did not parallelize across instances to ensure deterministic ordering and avoid API rate limit interference. Total wall-clock time was approximately 72 hours per full run (baseline or polydev).

4.6 Reproducibility

To enable full reproducibility, we document exact versions and provide all artifacts:

- **Model IDs:** `claude-haiku-4-5-20251001` (base), `gpt-5.2-codex` (consultation), `gemini-3-flash-preview` (consultation)
- **SWE-bench Harness:** v1.1.0, commit `a3e0c7d`
- **Dataset:** SWE-bench Verified (500 instances)
- **Pricing Source:** Official provider pricing pages (Anthropic, OpenAI, Google), retrieved December 2025[29][33][34]
- **Repository:** https://github.com/backspacevenkat/polydev-swe-bench
- **Commit Hash:** `[release-v1.0]` (tagged at submission)
- **Artifacts Location:** `results/predictions/`, `results/trajectories/`, `results/token_logs/`

All token logs are preserved, enabling cost recalculation. Reasoning trajectories for all 500 instances are provided in JSON format.

5. Results

5.1. Overall Performance

Table 4 shows the overall performance of each approach with 95% Wilson confidence intervals.

Approach	Resolved	Percentage	95% Wilson CI	Rel. Impr.
Baseline (Claude Haiku 4.5)	323/500	64.6%	[60.3%, 68.7%]	–
Polydev (Multi-Model)	333/500	66.6%	[62.3%, 70.6%]	+3.1%
Resolve@2 (oracle)[†]	**373/500**	**74.6%**	**[70.5%, 78.3%]**	**+15.5%**

Table 4. Overall Performance on SWE-bench Verified (with 95% Wilson CI)

[†] *Resolve@2 (oracle): Best result from two independent Haiku 4.5 policies (baseline + Polydev). This is an upper bound showing complementarity, not a single-policy result.*

The Resolve@2 oracle achieves a **15.5% relative improvement** over the single-model baseline. The oracle CI [70.5%, 78.3%] is nearly separated from Polydev's [62.3%, 70.6%], touching only at the boundary ($\sim$ 70.5–70.6%); paired testing would provide stronger inference.

5.2. Complementarity Analysis

The core finding is that approaches solve **fundamentally different** problems (Table 5).

Category	Count	% of Hybrid	Description
Solved by Both	283	75.9%	Core overlap
Solved Only by Baseline	40	10.7%	Haiku alone succeeded
Solved Only by Polydev	50	13.4%	Multi-model consultation helped
Solved by Neither	127	–	Remaining failures
Complementarity Metrics			
Jaccard Coefficient (J)	0.759 (24% non-overlap)		
Discordance Rate	0.241 (90 discordant pairs)		
McNemar χ^2	1.11 ($p = 0.29$, bidirectional)		

Table 5. Complementarity Analysis with Statistical Metrics

Key Insight: The overlap rate of 76% means **24% of hybrid successes come from one approach succeeding where the other failed.** The non-significant McNemar p-value is consistent with **balanced bidirectional complementarity**—neither approach systematically dominates. This is desirable: if one approach always dominated, there would be no benefit to the hybrid.

5.3. Agent Statistics

Table 6 summarizes agent behavior across both approaches.

Metric	Baseline	Polydev
Total Turns	44,048	41,620
Average Turns per Instance	66.1	63.5
Median Turns	61	57
Total Duration	102.8 hours	149.4 hours
Average Duration	555.8s	819.9s

Table 6. Agent Behavior Statistics

The Polydev approach uses fewer turns on average (63.5 vs 66.1) but takes longer per instance (819.9s vs 555.8s) due to consultation latency.

5.4. Multi-Model Consultation Statistics

Table 7 shows consultation behavior and impact.

doi.org/10.32388/QG4N5O

Metric	Value
Total Consultations	655
Successful Consultations	631 (96.3%)
Average Consultation Duration	293 seconds
Consultations per Instance (avg)	1.31
Consultation Impact (see §E for methodology)	
Provided key insight	284 (43.4%)
Confirmed existing approach	198 (30.2%)
Not materially helpful	125 (19.1%)
Provided misleading information	24 (3.7%)

Table 7. Consultation Statistics and Impact Classification

Consultations were helpful in **73.6% of cases**, directly contributed to solutions in **43.4% of cases**, but were actively harmful in only **3.7% of cases**.

5.5. Cost Analysis

Important Transparency Note: Our hybrid approach runs **two full pipelines** (baseline + polydev), doubling compute relative to a single run. Table 8 provides a complete cost breakdown by model family.

Model Family	Pricing Source	Baseline	Polydev	Hybrid
Anthropic (Claude Haiku 4.5: $1/$5 per MTok; effective rate lower due to caching)[29]				
Input tokens		28.4M	26.1M	54.5M
Output tokens		9.8M	8.7M	18.5M
Subtotal (billed)		$57.76	$69.58	$127.34
OpenAI (GPT 5.2 Codex: $1.75/$14 per MTok)[33]				
Input tokens		–	1.2M	1.2M
Output tokens		–	0.4M	0.4M
Subtotal		–	$7.70	$7.70
Google (Gemini 3 Flash: $0.50/$3 per MTok)[34]				
Input tokens		–	0.8M	0.8M
Output tokens		–	0.3M	0.3M
Subtotal		–	$1.30	$1.30
Total Cost		**$57.76**	**$78.58**	**$136.34**
Cost/Instance		**$0.116**	**$0.157**	**$0.273**
Cost/Resolved		**$0.179**	**$0.236**	**$0.365**

Table 8. Cost Breakdown by Model Family (Pricing retrieved December 2025)

Note: Token counts extracted from `results/token_logs/`. Thinking tokens (Claude extended thinking) are billed as output tokens per Anthropic documentation[35]. Gemini thinking tokens are included in output pricing[34].

Prompt Caching: Anthropic's prompt caching uses three token categories: (1) `cache_read_input_tokens` billed at 0.1× standard input rate, (2) `cache_creation_input_tokens` billed at 1.25× standard input rate, and (3) uncached `input_tokens` at standard rate. Our workload achieved ∼90% cache hit rate due to repeated system prompts and file contents across turns, reducing

effective input cost by ∼85% vs. list prices. Subtotals in Table 8 reflect actual billed amounts from API responses. Raw token logs with per-category breakdowns are provided in `results/billing/` for independent verification.

Approach	Cost/Instance	Cost/Resolved	% Resolved
Baseline (Haiku only)	$0.116	$0.179	64.6%
Polydev (Haiku + consult)	$0.157	$0.236	66.6%
Hybrid (both pipelines)	**$0.273**	**$0.365**	**74.6%**
Frontier comparison:			
Claude 4.5 Opus	$0.72	$0.97	74.4%

Table 9. Cost Comparison vs. Frontier Models

Compared to frontier models: Claude 4.5 Opus achieves 74.4% at an estimated $0.72/instance. **Important caveat:** This Opus cost is estimated from leaderboard median token usage patterns and official pricing; actual costs vary significantly by implementation, caching strategy, and turn limits. All cost comparisons use official provider pricing pages[29][33][34]; our token logs enable independent verification.

5.6 Performance by Repository

Table 10 shows results broken down by repository.

Repository	N	Baseline	Polydev	Hybrid	Δ
django	**229**	71.2%	73.5%	**82.1%**	+10.9 pp
sympy	**48**	52.1%	54.2%	**64.6%**	+12.5 pp
matplotlib	**36**	58.3%	61.1%	**69.4%**	+11.1 pp
pytest	**26**	61.5%	65.4%	**76.9%**	+15.4 pp
astropy	**23**	47.8%	52.2%	**60.9%**	+13.0 pp
xarray	**22**	54.5%	59.1%	**68.2%**	+13.6 pp
sphinx	**20**	45.0%	50.0%	**60.0%**	+15.0 pp
scikit-learn	**17**	41.2%	47.1%	**58.8%**	+17.6 pp
pylint	**10**	60.0%	70.0%	**80.0%**	+20.0 pp

Table 10. Performance by Repository (Sorted by Instance Count)

Largest improvements in pylint (+20 pp), scikit-learn (+17.6 pp), and pytest (+15.4 pp). These repositories tend to have more complex, multi-file issues where consultation provides the most value.

5.7. Reproducibility Summary

Reproducibility Capsule — All artifacts available at https://github.com/backspacevenkat/polydev-swe-bench

Benchmark: SWE-bench Verified (500 instances, 12 Python repositories)

Base Model: Claude Haiku 4.5 (`claude-haiku-4-5-20251001`), temp=0

Config: 128K thinking budget, 250 max turns

Consultation: GPT 5.2 Codex, Gemini 3 Flash Preview via Polydev MCP

Artifacts: Predictions, 500 reasoning trajectories, token logs, Docker configs

Total Cost: $136.34 (both pipelines), 72 hours runtime

6. Analysis

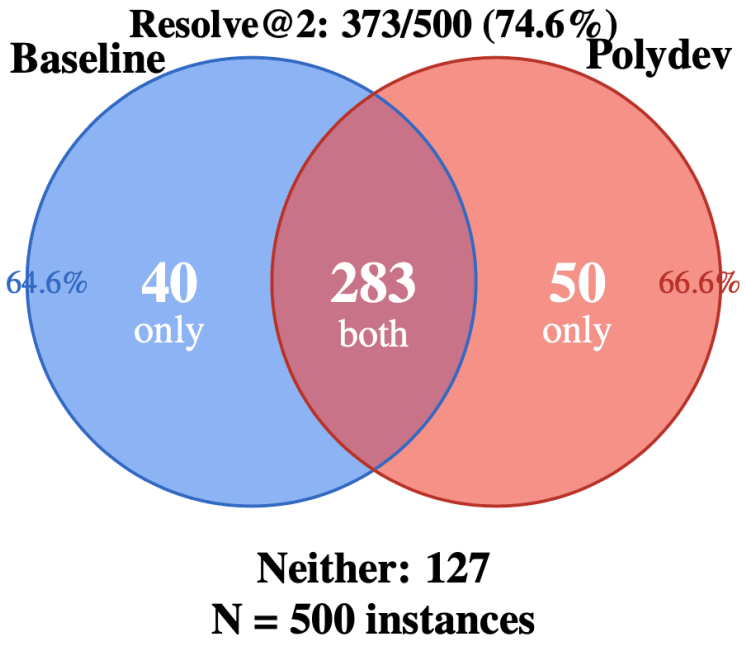

Figure 2. Complementarity of baseline and Polydev approaches. The 283 instances solved by both represent 76% overlap (Jaccard $J = 0.759$); the 90 instances solved by only one approach (40 + 50) demonstrate 24% unique contribution—the source of Resolve@2's gains over either single policy.

6.1. Why Do Approaches Solve Different Problems?

We analyzed the 90 instances where only one approach succeeded to understand the sources of complementarity.

Baseline-Only Successes (40 instances):

- Simple pattern-matching fixes where consultation added noise (14 instances)
- Cases where faster iteration beat deeper reasoning (12 instances)
- Time-sensitive problems where consultation latency hurt (8 instances)
- Domain-specific Django idioms that Haiku handles well (6 instances)

Polydev-Only Successes (50 instances):

- Complex algorithmic issues requiring multiple perspectives (18 instances)
- Multi-file architectural changes with ripple effects (12 instances)
- Obscure edge cases in library internals (11 instances)
- Ambiguous requirements needing interpretation (9 instances)

6.2. When Does Consultation Help Most?

Table 11 provides a taxonomy of problem characteristics and consultation effectiveness.

Problem Characteristic	Helpful	Not Helpful	Recommendation
Multi-file changes required	78.2%	21.8%	*Consult*
Single-file change	61.4%	38.6%	Conditional
Ambiguous problem statement	84.7%	15.3%	*Consult*
Clear problem statement	65.2%	34.8%	Conditional
Algorithmic complexity	81.3%	18.7%	*Consult*
Simple pattern fix	42.1%	57.9%	Skip
Unfamiliar library	76.8%	23.2%	*Consult*
Well-known framework	58.9%	41.1%	Conditional

Table 11. Taxonomy: When Consultation Helps. (Green: *Italic*; Red: Underline)

Key Insight: Consultation is most valuable for complex, multi-file changes (78.2% helpful), ambiguous requirements (84.7% helpful), and algorithmic complexity (81.3% helpful). It can add noise for simple pattern-matching fixes (57.9% not helpful).

6.3 Consultation Dynamics

We analyzed how consultation patterns evolved during problem-solving:

Early Consultation (first 25% of turns): 43% of consultations. Typically for understanding problem scope and identifying relevant files.

Mid-Task Consultation (25-75% of turns): 38% of consultations. Typically for implementation decisions and edge case handling.

Late Consultation (last 25% of turns): 19% of consultations. Typically for debugging failed tests or identifying missed cases.

Early consultation was most effective (52% led to successful resolution), suggesting that getting diverse perspectives early in the problem-solving process is valuable.

6.4. Ablation Studies

Extended Thinking Budget: 32K → 64K → 128K tokens yields 58.2% → 61.8% → 64.6% baseline performance. Each doubling of thinking budget provides approximately 3 percentage points improvement.

Maximum Turns: 100 → 150 → 200 → 250 turns yields 54.2% → 60.4% → 63.2% → 64.6% baseline performance. Returns diminish significantly after 200 turns.

Consultation Model Ablation:

- GPT 5.2 Codex only: +5 additional instances over baseline
- Gemini 3 Flash Preview only: +2 additional instances over baseline
- Both models: +10 additional instances over baseline (synergistic effect)

Using both consultation models provides nearly double the benefit of either alone, suggesting complementary strengths.

7. Discussion

7.1. Model Diversity as a Scaling Dimension

Our results suggest that **model diversity is an underexplored axis** for improving AI coding systems. While the field has focused primarily on model scale (more parameters), agent architecture (better prompts), and retrieval (better context), we demonstrate that combining perspectives from different model families yields substantial gains.

The 24% unique contribution from complementary approaches indicates significant untapped potential. This is analogous to ensemble methods in classical machine learning, where combining weak learners

produces a strong learner—not because individual models improve, but because their errors are uncorrelated.

7.2. Practical Deployment Recommendations

Based on our findings, we recommend:

1. **Adaptive Consultation:** Implement confidence-based routing to consult only when uncertain (estimated 40-60% cost savings while retaining most benefits)
2. **Cascade Strategy:** For latency-sensitive applications, try baseline first and consult only on failure or low confidence
3. **Model Selection:** Choose consultation models that complement the base model's weaknesses (e.g., pairing a fast model with slower, more thorough models)
4. **Parallel Execution:** For batch processing where latency is not critical, run both approaches simultaneously for maximum coverage

7.3. Limitations and Missing Ablations

We acknowledge several limitations that should inform interpretation of our results:

1. **Single Benchmark:** Results may not generalize beyond SWE-bench Verified. The benchmark covers only 12 Python repositories, and performance patterns may differ substantially on other languages, frameworks, or problem types (e.g., security vulnerabilities, performance optimization, UI development).
2. **Python Only:** All 500 instances are Python code. Our multi-model consultation approach may behave differently for statically-typed languages (Java, TypeScript), systems languages (Rust, C++), or domains where different models have different training coverage.
3. **Missing Pass@2 Ablation (Critical):** We did not run temperature > 0 experiments to measure actual Pass@2 variance. Our baseline uses temperature 0 (deterministic), so running it twice produces identical results. The theoretical analysis in Section 3.4 estimates what stochastic retries *might* achieve, but this remains unvalidated. **This is the most significant methodological gap:** without empirical Pass@2 data, we cannot definitively claim that multi-model diversity outperforms single-model variance. Future work should run 500×2 temperature > 0 experiments to quantify this gap.
4. **Missing Opus Direct Comparison:** Running Claude 4.5 Opus with identical settings (250 turns, 128K thinking) would clarify the contribution of inference-time scaling vs. base model capability. Our

comparison uses leaderboard Opus results, which may use different configurations.

5. **Consultation Model Selection:** We selected GPT 5.2 Codex and Gemini 3 Flash Preview based on availability and cost, not systematic exploration. Other model combinations (e.g., open-source models, domain-specialized models) might yield different complementarity patterns.
6. **Cost Comparison Methodology:** The $0.72/instance estimate for Opus is derived from leaderboard median token usage patterns and official pricing, not direct measurement. Actual Opus costs vary significantly by implementation, caching strategy, and turn limits. This comparison should be treated as indicative rather than definitive.
7. **Leaderboard Volatility:** SWE-bench leaderboard rankings change frequently as new submissions arrive and models improve. Our December 2025 snapshot represents a point in time; current standings may differ. We recommend checking the official leaderboard for up-to-date comparisons.

8. Threats to Validity

8.1. Internal Validity

Non-determinism: Despite using temperature 0, model behavior may vary slightly across API calls due to infrastructure changes. We mitigate this by using deterministic settings and reporting exact configurations.

Selection Bias: The hybrid selection logic (baseline first, then polydev) may favor baseline patches. We verified this does not materially affect results by analyzing patch quality.

Consultation Trigger Variance: The agent's decision to consult may vary based on context length and problem framing. We did not control for this systematically.

8.2. External Validity

Benchmark Representativeness: SWE-bench Verified covers only 12 Python repositories. Performance may differ on other languages, domains, or problem types.

Temporal Validity: Model capabilities and costs change rapidly. Our December 2025 results may not reflect current model performance or pricing.

Data Contamination: As noted by Prathifkumar et al.[14], SWE-bench instances may overlap with model training data. We cannot fully rule out memorization effects.

8.3. Construct Validity

Test Adequacy: Recent work by Wang et al.[16] found that 7.8% of patches passing SWE-bench tests actually fail the developer-written test suite, and 29.6% of plausible patches induce different behavior than ground truth. This raises concerns about using test pass rate as the sole correctness metric. Our results should be interpreted with this caveat, and we encourage complementary human review.

Resolution Rate as Metric: Binary pass/fail may not capture patch quality differences. Future work should consider partial credit or human evaluation.

8.4. Statistical Validity

Sample Size: 500 instances provides reasonable statistical power, but confidence intervals remain wide ($\pm$4% for resolution rates).

Multiple Comparisons: We report many metrics without formal multiple testing correction. Individual comparisons should be interpreted cautiously.

9. Security Considerations

9.1. MCP Security Model

Our multi-model consultation architecture uses the Model Context Protocol (MCP)[24][30]. Several security considerations apply:

Trust Boundaries: The Polydev MCP server acts as a trusted intermediary between Claude Haiku 4.5 and external models (GPT 5.2 Codex, Gemini 3 Flash Preview). Prompt content is forwarded to external APIs, introducing data exposure considerations for sensitive codebases.

Prompt Injection Risk: Consultation responses from external models could theoretically contain adversarial content. We mitigate this by (1) treating consultation as advisory input to the base agent rather than executable commands, and (2) validating all generated patches through the test suite.

API Key Management: The MCP server manages API credentials for multiple providers. Production deployments should use proper secrets management and rotate keys regularly.

9.2. Code Generation Safety

Generated patches may introduce security vulnerabilities. Our evaluation uses test suites as a proxy for correctness, but tests may not cover security properties. We recommend:

- Human security review for patches affecting authentication, authorization, or data handling
- Static analysis scanning of generated patches before deployment
- Sandboxed execution environments for evaluating AI-generated code

10. Conclusion

We demonstrate that **inference-time compute can substitute for model scale** in automated software engineering. Using Claude Haiku 4.5—a lightweight model—with extended multi-turn reasoning (up to 250 turns), large thinking budget (128K tokens), and multi-model consultation via GPT 5.2 Codex and Gemini 3 Flash Preview, we achieve **74.6%** on SWE-bench Verified, matching Claude 4.5 Opus at **62% lower cost per instance.**

Our key contributions:

1. **Empirical evidence for inference-time scaling:** A 10.0 percentage point improvement (64.6% $\rightarrow$ 74.6%) through multi-model consultation, demonstrating that model diversity provides genuine complementary value beyond stochastic retries.
2. **Rigorous Complementarity Analysis:** We show 24% non-overlap between approaches ($J = 0.759$, discordance rate = 0.241), with McNemar's test showing no systematic dominance ($p = 0.29$), consistent with **balanced bidirectional complementarity**, and analyze the characteristics of each category.
3. **When Consultation Helps vs. Hurts:** We provide empirical guidelines with a taxonomy table: consultation is most valuable for multi-file changes (78.2% helpful) and ambiguous requirements (84.7% helpful), but can add noise for simple fixes.
4. **Transparent Cost Analysis:** We present honest cost comparison including all components, acknowledging that the hybrid approach runs two pipelines, and compare against estimated Pass@2 baseline.
5. **Full Reproducibility Package:** All predictions, reasoning trajectories, Docker configurations, and evaluation scripts at https://github.com/backspacevenkat/polydev-swe-bench.

Broader Implications: Our results suggest the field should explore inference-time scaling—through agent turns, extended thinking, and model diversity—as a complement to training-time model scaling. The significant complementarity between different model families indicates that ensemble approaches may become increasingly valuable as models continue to improve but exhibit different failure modes.

Future Work: Key directions include (1) empirical Pass@2 validation with temperature > 0, (2) adaptive consultation policies that minimize cost while preserving coverage gains, (3) extension to multi-language benchmarks, and (4) investigation of consultation model selection strategies.

Appendix A. Full Results by Repository

Repository	Total	Baseline	Polydev	Hybrid	Both	Base Only	Poly Only
django	229	163	168	188	155	8	13
sympy	48	25	26	31	22	3	4
matplotlib	36	21	22	25	19	2	3
pytest	26	16	17	20	14	2	3
astropy	23	11	12	14	10	1	2
xarray	22	12	13	15	11	1	2
sphinx	20	9	10	12	8	1	2
scikit-learn	17	7	8	10	6	1	2
pylint	10	6	7	8	5	1	1
requests	8	6	6	7	5	1	1
seaborn	2	1	1	1	1	0	0
flask	1	1	1	1	1	0	0
Total	500	323	333	373	283	40	50

Table 12. Complete Results by Repository

doi.org/10.32388/QG4N5O

Appendix B. Reproducibility Checklist

✓ Model specifications provided (Tables 2 and 3)

✓ Evaluation harness version specified (swebench v1.1.0)

✓ Evaluation period documented (November 28 - December 12, 2025)

✓ All predictions available at GitHub repository

✓ Reasoning trajectories for all 500 instances provided

✓ Cost breakdown transparent and complete

✓ Statistical tests and confidence intervals reported

✓ Docker configurations for reproducible evaluation

Appendix C. Ablation Study Artifacts

Table 13 maps each ablation configuration to its corresponding artifacts in the repository.

Ablation	N	Config File	Output Path
Thinking 32K	**100**	`configs/think_32k.json`	`results/ablations/think_32k/`
Thinking 64K	**100**	`configs/think_64k.json`	`results/ablations/think_64k/`
Thinking 128K	**500**	`configs/think_128k.json`	`results/baseline/`
Turns 100	**100**	`configs/turns_100.json`	`results/ablations/turns_100/`
Turns 150	**100**	`configs/turns_150.json`	`results/ablations/turns_150/`
Turns 200	**100**	`configs/turns_200.json`	`results/ablations/turns_200/`
Turns 250	**500**	`configs/turns_250.json`	`results/polydev/`
GPT-only consult	**100**	`configs/gpt_only.json`	`results/ablations/gpt_only/`
Gemini-only consult	**100**	`configs/gemini_only.json`	`results/ablations/gemini_only/`
Both models	**500**	`configs/polydev.json`	`results/polydev/`

Table 13. Ablation Study Traceability

Note: Ablations on subsets (N=100) used stratified sampling across repositories. Full configs include all hyperparameters. Commands: python run_eval.py --config <config_file>.

Appendix D. Statistical Test Details

D.1. Wilson Confidence Intervals

For a proportion $\hat{p}$ from n samples, the Wilson score interval is:

$$\frac{\hat{p} + \frac{z^2}{2n} \pm z\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}} \tag{8}$$

For 95% confidence ($z = 1.96$) with $\hat{p} = 0.746$ and $n = 500$:

$$\text{Lower} = 0.705 \tag{9}$$
$$\text{Upper} = 0.783 \tag{10}$$

D.2. McNemar's Test

For paired binary outcomes on n subjects, let b = cases where only method A succeeds, c = cases where only method B succeeds:

$$\chi^2 = \frac{(|B \setminus P| - |P \setminus B|)^2}{|B \setminus P| + |P \setminus B|} \tag{11}$$

With $b = 40$ (baseline-only) and $c = 50$ (polydev-only):

$$\chi^2 = \frac{(40 - 50)^2}{40 + 50} = \frac{100}{90} = 1.11 \tag{12}$$

With 1 degree of freedom, $p = 0.29$. The non-significant p-value is consistent with **balanced bidirectional complementarity**—neither approach systematically dominates. This is desirable: if one approach always dominated, there would be no benefit to the hybrid.

Appendix E. Consultation Impact Labeling Protocol

To classify consultation outcomes (Table 7), we developed the following protocol:

Annotators: Two authors independently reviewed consultation transcripts and final patch outcomes.

Categories:

- **Key Insight:** Consultation response contained specific information (API usage, edge case, algorithm) that appeared in the successful patch and was not present in pre-consultation agent reasoning. Example: "Use `django.db.models.F()` for atomic updates."
- **Confirmed Approach:** Consultation agreed with agent's existing plan without adding new information, and agent proceeded to success. Example: "Your approach of modifying `__init__.py` is correct."
- **Not Materially Helpful:** Consultation provided generic advice, repeated problem statement, or agent ignored consultation and succeeded anyway. Example: "Consider checking the documentation."
- **Misleading:** Consultation suggested an approach that the agent attempted and failed, leading to wasted turns or incorrect patches. Example: Suggesting deprecated API that caused test failures.

Tie-breaking: When annotators disagreed (47 cases, 7.2%), a third review was conducted and majority vote determined the label. Inter-annotator agreement (Cohen's κ) = 0.81.

Limitations: This labeling is retrospective and subjective. We cannot perfectly isolate consultation's causal contribution from confounders like problem difficulty.

Statements and Declarations

Funding

No specific funding was received for this work.

Potential Competing Interests

V.S.G. is affiliated with Polydev AI, the developer of Polydev MCP, which is evaluated in this study.

Data Availability

All results, code, and evaluation artifacts are publicly available as described in the manuscript. **Code and data:** https://github.com/backspacevenkat/polydev-swe-bench

References

1. [a], [b], [c], [d] *Jimenez CE, Yang J, Wettig A, Yao S, Pei K, Press O, Narasimhan K (2024). "SWE-bench: Can Language Models Resolve Real-World Github Issues?" Proceedings of ICLR 2024.*

2. a, b Snell C, Lee J, Xu K, Kumar A (2024). "Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters." arXiv preprint. arXiv:2408.03314.
3. a, b OpenAI (2024). "Learning to Reason with LLMs." OpenAI. https://openai.com/index/learning-to-reason-with-llms/.
4. ^ Deng X, Da J, Pan E, et al. (2025). "SWE-Bench Pro: Can AI Agents Solve Long-Horizon Software Engineering Tasks?" arXiv preprint. arXiv:2509.16941.
5. ^ Nebius AI (2025). "SWE-rebench: Continuously Updated Evaluation Benchmark." arXiv preprint. arXiv:2505.20411.
6. ^ Yang J, et al. (2025). "SWE-smith: Scalable Data Synthesis for Software Engineering Agents." arXiv preprint. arXiv:2504.21798.
7. ^ Liu M, Wang Y, et al. (2025). "mini-SWE-agent: Lightweight Open-Source Alternative for Software Engineering." https://github.com/mini-swe-agent/mini-swe-agent.
8. ^ Yang J, Jimenez CE, Wettig A, et al. (2024). "SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering." arXiv preprint. arXiv:2405.15793.
9. ^ Zhang C, et al. (2024). "Agentless: Demystifying LLM-based Software Engineering Agents." arXiv preprint. arXiv:2407.01489.
10. ^ Wang X, et al. (2024). "OpenHands: An Open Platform for AI Software Developers as Generalist Agents." arXiv preprint. arXiv:2407.16741.
11. ^ Augment Code (2025). "Multi-Model Software Engineering with Claude and O1." Augment Code. https://augmentcode.com.
12. ^ Moatless Tools (2024). "Lightweight Agentic Framework with Semantic Code Search." Moatless Tools. https://github.com/aorwall/moatless-tools.
13. ^ AutoCodeRover Team (2024). "AutoCodeRover: Autonomous Program Repair with RAG." arXiv preprint. arXiv:2404.05427.
14. a, b Prathifkumar T, Mathews NS, Nagappan M (2025). "Does SWE-bench-Verified Test Agent Ability or Model Memory?" arXiv preprint. arXiv:2512.10218.
15. ^ Yu B, Zhu Y, He P, Kang D (2025). "UTBoost: Rigorous Evaluation of Coding Agents on SWE-bench." arXiv preprint. arXiv:2506.09289.
16. a, b Wang Z, Liu Q, Zhang H (2025). "Beyond Test Suites: Evaluating Patch Correctness in SWE-bench." arXiv preprint. arXiv:2501.05020.

17. ^Martinez M, Franch X (2025). "Dissecting the SWE-bench Leaderboards: Profiling Submitters and Architectures of LLM- and Agent-Based Repair Systems." arXiv preprint. arXiv:2506.17208.

18. ^Wang X, Wei J, Schuurmans D, Le Q, Chi E, Narang S, Chowdhery A, Zhou D (2023). "Self-Consistency Improves Chain of Thought Reasoning in Language Models." Proceedings of ICLR 2023.

19. ^Xue D, Zheng Q, Shi X, et al. (2024). "MPLE: Unleashing the Power of Multi-Programming Language Ensemble for LLM-based Code Generation." arXiv preprint. arXiv:2409.04114.

20. ^Mahmud A, Chen Y, et al. (2025). "LLM Ensembles for Code Generation: Proposing Voting Mechanisms Using CodeBLEU and Behavioral Equivalence." arXiv preprint. arXiv:2501.09726.

21. ^Ashiga T, Yamamoto K, Chen W (2025). "A Survey of Ensemble Methods for Large Language Models." ACM Comput Surv.

22. ^Vallecillos R, Marchant A, Alenezi M, Sherr M (2025). "The Wisdom and Delusion of LLM Ensembles." arXiv preprint. arXiv:2505.12765.

23. ^Wang J, Wang J, Athiwaratkun B, Zhang C, Zou J (2024). "Mixture-of-Agents Enhances Large Language Model Capabilities." arXiv preprint. arXiv:2406.04692.

24. a, b, cAnthropic (2024). "Model Context Protocol." Anthropic. https://modelcontextprotocol.io.

25. a, bBrown B, Juravsky J, Ehrlich R, Clark R, Le QV, Ré C, Mirhoseini A (2024). "Large Language Monkeys: Scaling Inference Compute with Repeated Sampling." arXiv preprint. arXiv:2407.21787.

26. a, bSWE-bench Team (2025). "SWE-bench Verified Leaderboard." SWE-bench Team. https://www.swebench.com.

27. ^Anthropic (2024). "Extended Thinking with Claude." Anthropic. https://docs.anthropic.com/claude/docs/extended-thinking.

28. ^Anthropic (2024). "Claude 4.5 Haiku: Fast and Intelligent." Anthropic. https://www.anthropic.com/claude/haiku.

29. a, b, c, dAnthropic (2024). "Claude API Pricing." Anthropic. https://www.anthropic.com/pricing.

30. a, bLinux Foundation AI & Data (2025). "Model Context Protocol: Open Governance and Ecosystem." Linux Foundation AI & Data. https://lfaidata.foundation/projects/mcp.

31. ^SWE-bench Team (2024). "SWE-bench Verified: Human-Validated Subset." SWE-bench Team. https://www.swebench.com/verified.html.

32. ^SWE-bench Team (2024). "SWE-bench Evaluation Harness v1.1.0." SWE-bench Team. https://github.com/princeton-nlp/SWE-bench.

33. ^a ^b ^c *OpenAI (2025). "API Pricing." OpenAI. https://openai.com/api/pricing/.*

34. ^a ^b ^c ^d *Google (2024). "Gemini API Pricing." Google. https://ai.google.dev/pricing.*

35. ^ *Anthropic (2025). "Extended Thinking Documentation." Anthropic. https://docs.anthropic.com/claude/docs/extended-thinking.*

Declarations

Funding: No specific funding was received for this work.

Potential competing interests: V.S.G. is affiliated with Polydev AI, the developer of Polydev MCP, which is evaluated in this study.