The Coding Agent Economy

A 12-Month Observation of Coding Agent Economics

Thibault Jaigu*

May 2026

Abstract

We analyze twelve months of production LLM gateway data covering nine coding agents (Claude Code [1], Roo Code [2], Cline [3], Kilo Code [4], OpenCode [5], Zed [6], Cursor [7], GitHub Copilot [8], and Codex CLI [9]) from May 2025 through April 2026. We find that: (1) average cost per active user (those with at least two active days per month) is $92/month, rising to $108/month for Claude Code with P95 users spending $291/month; (2) Claude models power 92% of all coding agent spend, up from 68% twelve months prior, representing a near-complete consolidation of model choice; and (3) prompt caching has transformed cost economics, with platform-wide cache hit rates rising from 52% to 86%, effectively compressing per-call costs even as context windows grew 68%. These findings have implications for model providers, agent developers, and infrastructure operators navigating the emerging economics of autonomous coding workflows.

1 Introduction

AI coding agents have evolved from simple autocomplete systems into autonomous software engineering tools capable of planning, executing, and iterating on complex multi-step tasks [15, 16]. This evolution carries significant economic implications. Where a single code completion required one API call, an agentic coding session may involve hundreds of calls as the agent reads files, plans changes, writes code, runs tests, and iterates on failures.

Despite the rapid growth of this category, empirical data on the economics of coding agents remains limited. Prior work has examined neural code completion productivity [17] and agent architectures for software engineering [15], but no longitudinal study has focused specifically on the cost structures, model preferences, and usage intensities of AI coding tools in production.

We present what we believe to be the first twelve-month longitudinal analysis of coding agent economics. Our data covers nine agents spanning the full spectrum from first-party tools (Claude Code [1], Codex CLI [9]) to open-source IDE extensions (Roo Code [2], Cline [3], Kilo Code [4], OpenCode [5]) to integrated editors (Zed [6], Cursor [7]). By observing these agents through a shared gateway, we can make direct comparisons of cost, efficiency, and model preference that are not available from any single provider's telemetry.

The paper is organized as follows. Sections 2 and 3 present the core economic findings: cost trends and caching dynamics. Section 4 covers model preferences. Section 5 examines usage intensity. Section 6 analyzes provider latency. Section 7 explores the cache-cost correlation. Section 8 synthesizes the findings and Section 9 concludes. The data sources, scope, and limitations are documented in the appendix.

*Requesty. Correspondence: thibault@requesty.ai

2 Cost per User Over Time

2.1 Platform Aggregate Trend

Figure 1 shows the weighted average cost per user per month across all coding agents combined.

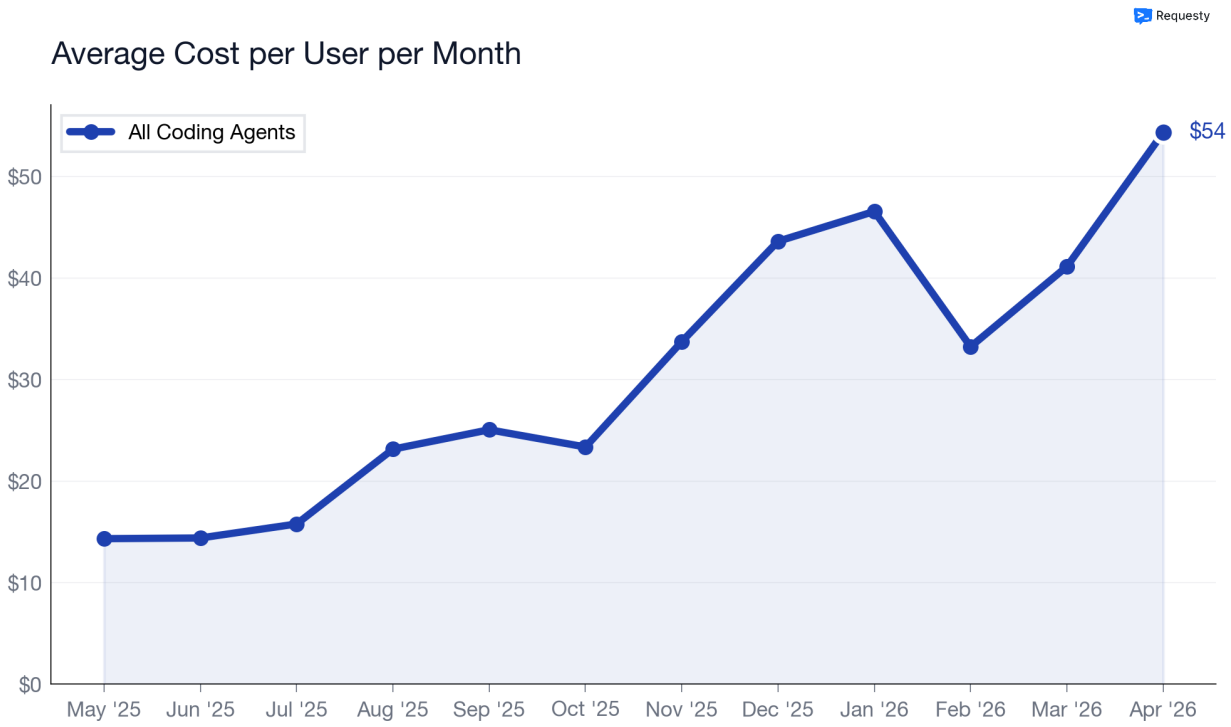

Figure 1. Average cost per user per month across all coding agents (May 2025 to April 2026). The weighted average has risen from $14 to over $54, a 3.8x increase in twelve months.

We observe a clear upward trend. The overall 3.8x increase reflects several converging factors:

- **Reasoning models.** The introduction of Claude Sonnet 4 [12], o3 [13], and Gemini 2.5 Pro [14] with extended thinking increased per-token costs while enabling more capable agentic behavior.
- **Growing context windows.** Average input tokens per call rose from approximately 50,000 (May 2025) to 84,000 (April 2026), a 68% increase reflecting agents passing more project context.
- **Longer sessions.** Claude Code [1] users averaged 1,549 API calls per month by April 2026, consistent with extended multi-step agentic workflows.
- **Composition shift.** The growing share of higher-spending agents (Claude Code, OpenCode) in the platform mix pulled the weighted average upward.

2.2 Cost by Agent

Figure 2 disaggregates the cost trend by agent, revealing substantial variation.

By April 2026, Claude Code users spend the most at $78/month on average, followed by Kilo Code [4] ($65), OpenCode ($58), and Roo Code ($46). Cline [3] users spend the least among major agents at $25/month.

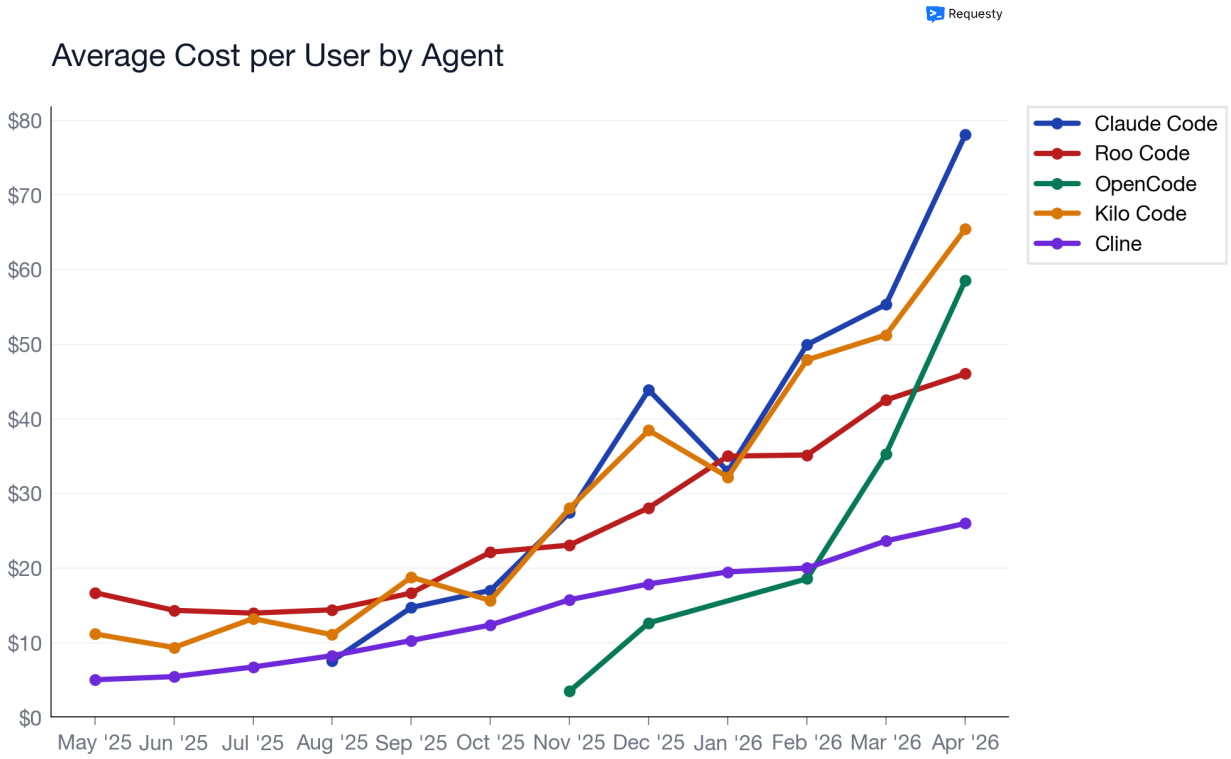

Figure 2. Average cost per user by agent over time. Claude Code [1] leads at $78/month in April 2026, followed by Kilo Code [4] ($65), OpenCode [5] ($58), and Roo Code [2] ($46).

The variation across agents reflects different usage patterns. Claude Code's high average is consistent with its intensive agentic loop (1,549 calls/month), while Cline's lower cost aligns with fewer but longer individual calls (214 calls/month at higher per-call cost).

2.3 Active User Economics

The averages above include all users, many of whom represent one-time trial accounts that made a single day of requests. Filtering to *active users* (those with at least two distinct active days in a given month) removes this trial-user noise and reveals what regular users actually spend.

Table 1. Cost per user by agent, April 2026: all users vs. active users (≥2 active days). Active-user means are 1.4–2.0× higher than all-user means.

Agent	All Mean	Active Mean	Active Median	Active P95
Claude Code	$78	$108	$23	$291
Roo Code	$46	$79	$25	$333
Kilo Code	$65	$107	$5	$268
Cline	$25	$50	$17	$205
OpenCode	$58	$104	$15	$473

Filtering to active users raises the weighted average across all agents from $54 to approximately

$92/month. Claude Code active users average $108/month, with a P95 of $291. The median-to-mean gap remains large across all agents (e.g. $23 median vs. $108 mean for Claude Code), confirming that a small number of power users drive the majority of spend.

The P95 figures reveal the ceiling for heavy users. At $291/month for Claude Code and $473/month for OpenCode, these represent developers running extended agentic sessions daily. Roo Code P95 users spend $333/month, similar to Claude Code despite lower average usage intensity, suggesting a subset of Roo Code users who have adopted comparably deep workflows.

3 The Caching Revolution

Prompt caching [11] has been the single most impactful economic shift in the coding agent space over this twelve-month period.

3.1 Platform Aggregate Cache Rate

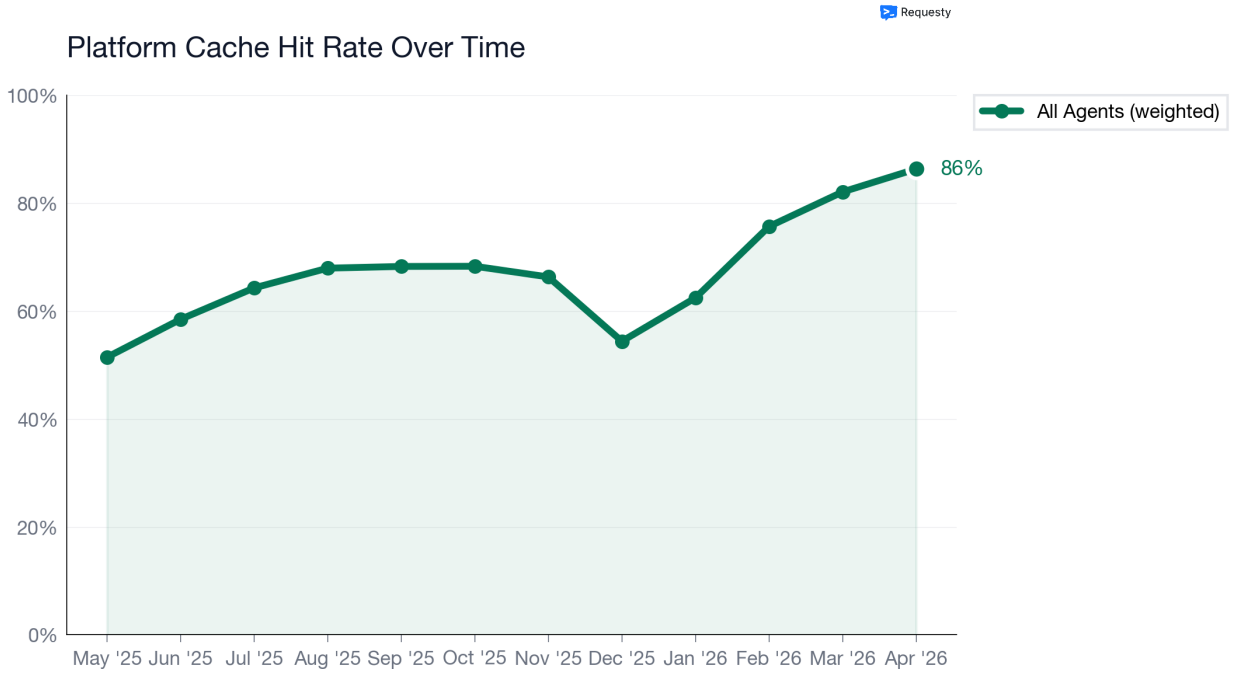

Figure 3. Platform-wide cache hit rate over time (weighted by input tokens). Cache hit rates rose from 52% to 86%, with rapid acceleration through early 2026 as agents optimized their context management.

The cache hit rate measures the fraction of input tokens served from prompt cache rather than reprocessed [11]. An 86% cache hit rate means that the effective cost of input tokens is roughly 7x lower than list price, fundamentally altering the economics of large-context agentic workflows.

3.2 Cache Efficiency by Agent

Claude Code achieves a 92% cache hit rate, meaning only 8% of its input tokens require fresh processing. This is extraordinary given that its average prompt is 84,000 tokens. The implication is that Claude Code's architecture is specifically optimized to maintain consistent context prefixes across sequential calls, maximizing cache reuse.

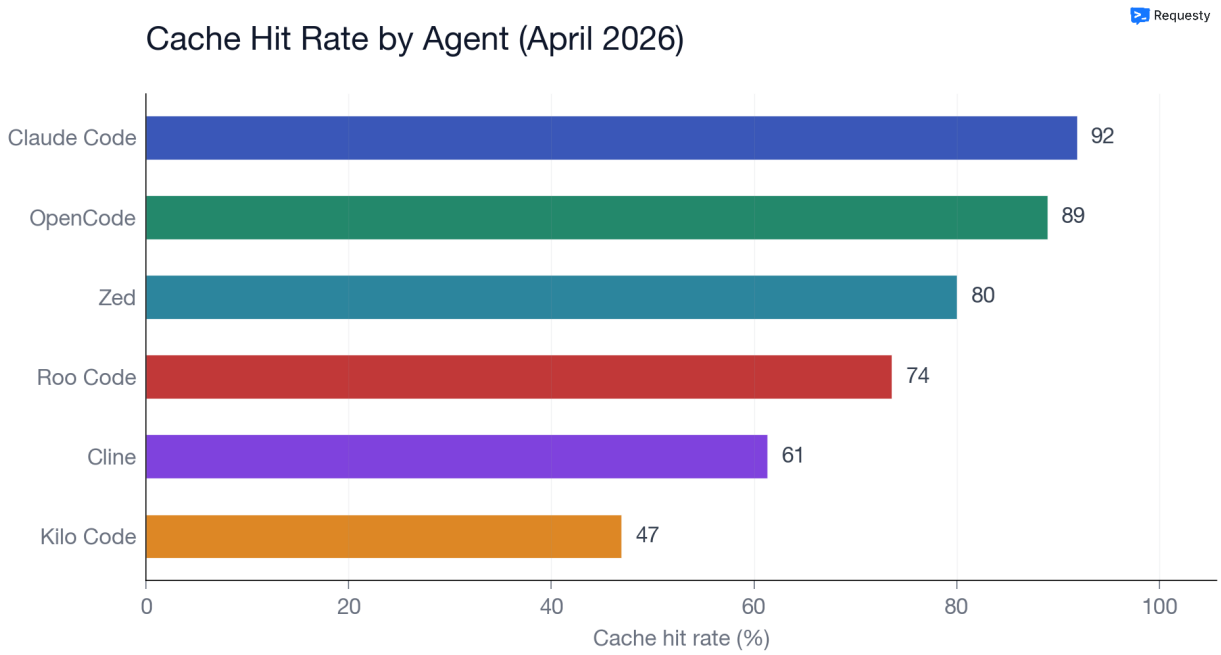

Figure 4. Cache hit rate by agent (April 2026). Claude Code [1] leads at 92%, while Kilo Code [4] sits at 46%. The spread suggests that caching efficiency is largely determined by agent architecture rather than model capability.

At the other end of the spectrum, Kilo Code caches 46% of input tokens, with smaller average context windows (62K vs 84K for Claude Code). This lower cache rate likely reflects different prompt construction patterns that reduce prefix reuse across sequential calls.

The economic impact is direct: Claude Code users can afford to make 1,549 API calls per month at a reasonable average spend ($78) precisely because 92% of every call's input tokens cost approximately 90% less than list price. Without caching, the same usage pattern would cost roughly 7x more.

4 Model Provider Preferences

4.1 The Claude Consolidation

The Claude model family (Sonnet, Opus, Haiku) has achieved near-total dominance of the coding agent category. In May 2025, Claude powered 68% of coding agent spend through our gateway. By December 2025, this had risen to 95%, and it has stabilized around 92% through April 2026.

This consolidation occurred organically across agents whose users freely choose their backend model. It reflects a revealed preference: when developers can pick any model for coding tasks, they overwhelmingly choose Claude.

4.2 Model Share by Agent

Several patterns emerge from the per-agent breakdown:

- **Claude Code is nearly 100% locked to Claude models.** This is expected given it is Anthropic's own product [1].

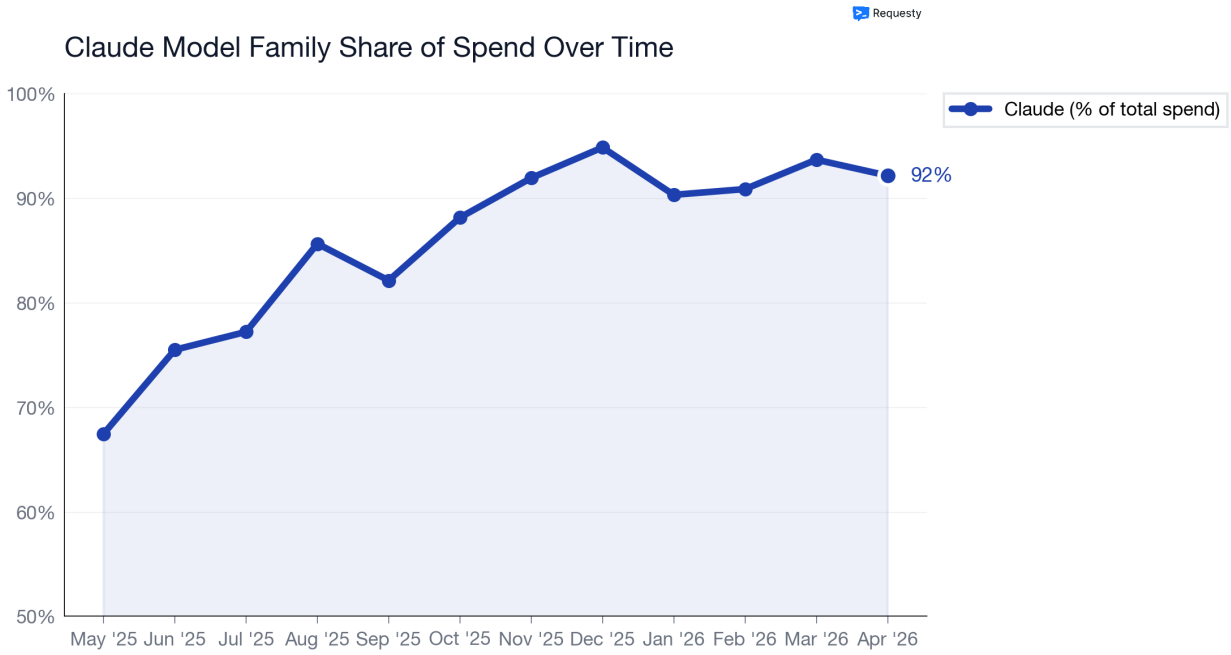

Figure 5. Claude model family share of total spend over time. Claude's share rose from 68% in May 2025 to a plateau of approximately 92% by late 2025, where it has remained.

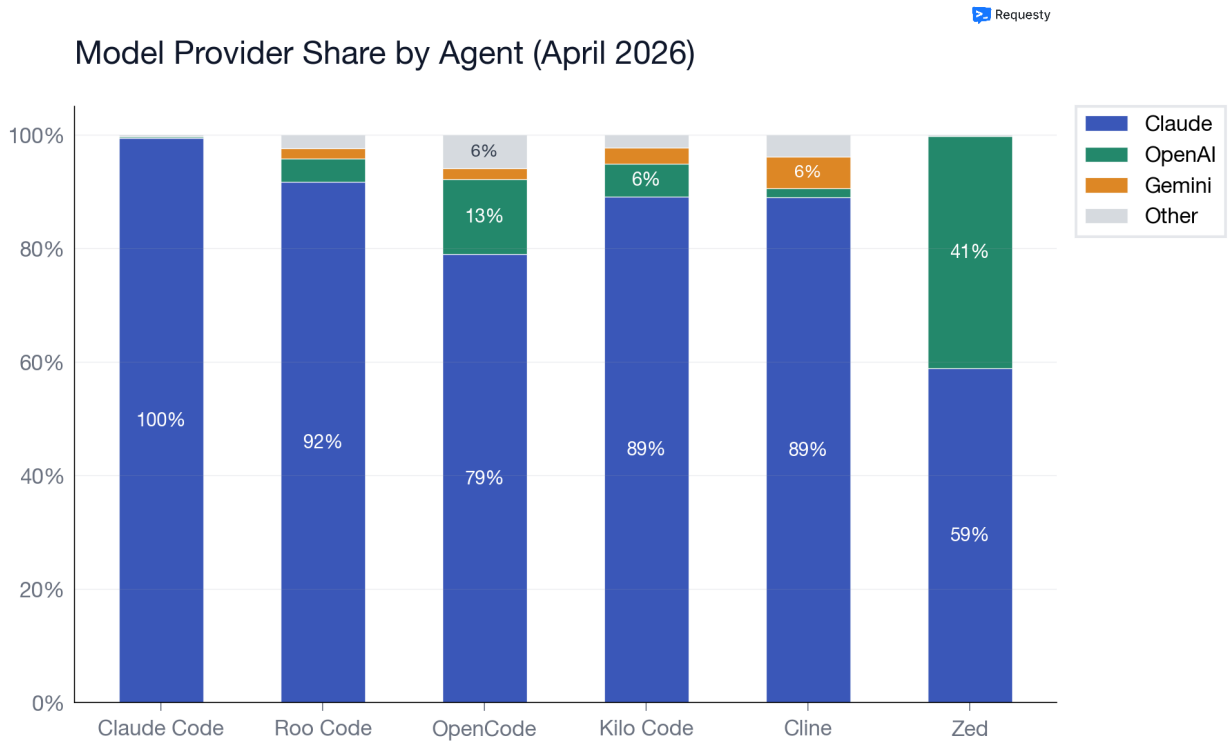

Figure 6. Model provider share by agent (April 2026). Claude dominates all agents from 79% (OpenCode [5]) to 100% (Claude Code [1]). Zed [6] is the most model-diverse at 59% Claude / 41% OpenAI.

- **Zed is the most model-diverse agent** at 59% Claude and 41% OpenAI [6]. This makes Zed a useful bellwether for multi-model strategies in integrated editors.
- **OpenCode has the highest non-Claude adoption among open-source agents** at 13% OpenAI and 5% Other [5], suggesting its users actively experiment with alternative models.
- **Gemini's share is minimal across all agents** (0 to 6%), despite competitive pricing from Google [14]. This suggests that price is not the primary decision factor for coding agent users.

The model consolidation story extends backward in time. In May 2025, Cline users split roughly 64/34 between Claude and Gemini. Aider [10] users were 20% Claude and 76% Gemini. By April 2026, both agents shifted to approximately 89% and 78% Claude respectively. Across every agent we track, Claude's share has grown.

5 Usage Intensity

5.1 API Calls per User

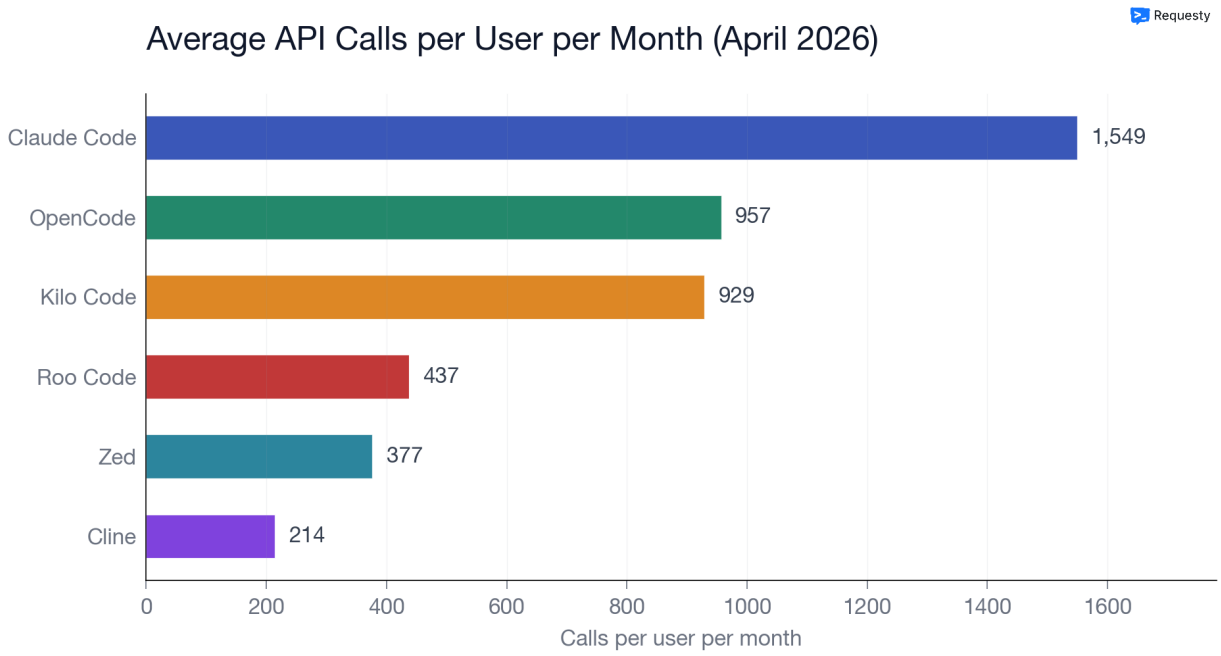

Figure 7. Average API calls per user per month (April 2026). Claude Code [1] users make 3.5x more API calls than Roo Code [2] users and 7x more than Cline [3] users, consistent with a highly iterative agentic loop pattern.

Claude Code's 1,549 calls per user per month (approximately 50 per day) reflects its "agentic loop" architecture: reading files, planning, executing changes, running tests, and iterating, each step generating one or more API calls. This is 3.5x more calls than Roo Code (437/month) and 7.2x more than Cline (214/month).

Notably, this high call volume does not translate to proportionally higher costs because of Claude Code's exceptional caching efficiency. At $0.050 per call (the lowest among major agents), Claude Code is able to sustain intensive usage at a fraction of what the same volume would cost with a less cache-optimized architecture.

5.2 Cost per Call

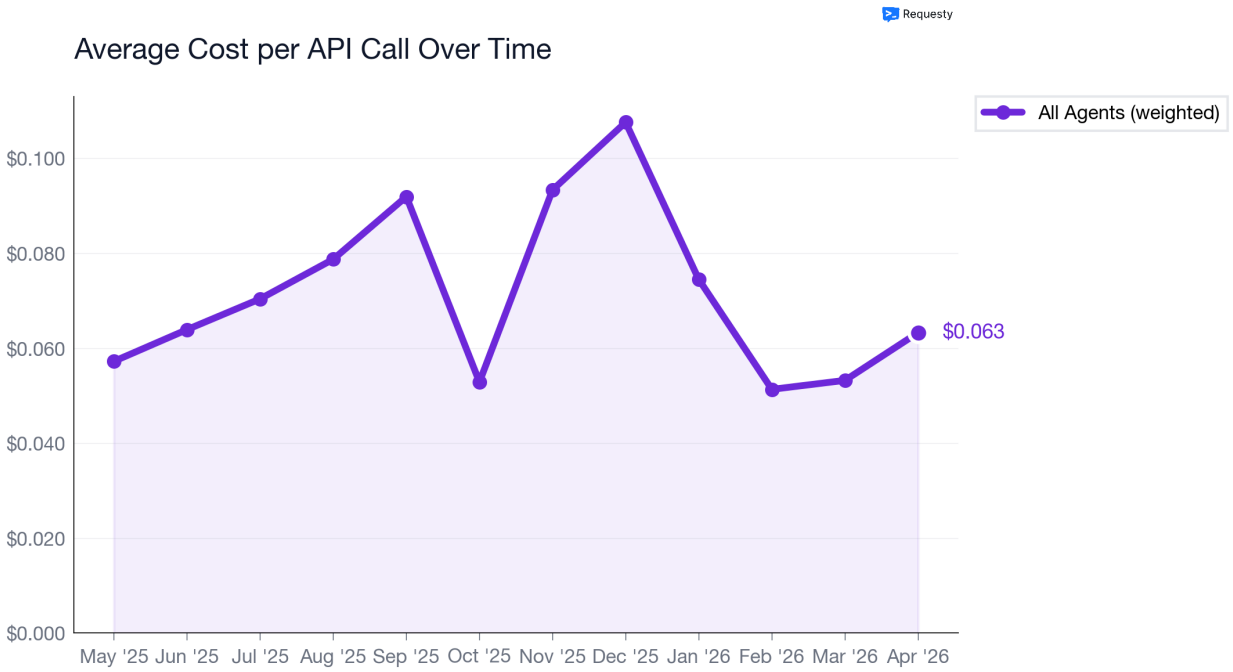

Figure 8. Average cost per API call over time (all agents weighted). Per-call costs have remained relatively stable between $0.05 and $0.10 despite growing context sizes, thanks to the offsetting effect of improved caching.

The stability of per-call costs in the face of growing context windows is a key finding. Average input tokens per call grew 68% over the period, yet average cost per call remained bounded. **Caching has almost exactly offset the cost growth from larger contexts.** This suggests that the per-user cost increase documented in Section 2 is primarily driven by increased call volume (more agentic steps per session) rather than more expensive individual calls.

6 Provider Latency and Routing

Coding agents that route through Requesty can reach the same model via multiple upstream providers: Anthropic direct, AWS Bedrock, and Google Vertex. This creates a natural experiment for comparing provider performance on identical workloads.

6.1 Claude Code Latency by Provider

For Claude Code in April 2026, we observe:

- **Anthropic direct:** 4.6s median provider latency, 2.2s median TTFT, 98.5% success rate
- **AWS Bedrock:** 4.9s median provider latency, 2.3s median TTFT, 96.0% success rate
- **Google Vertex:** 4.0s median provider latency, 1.7s median TTFT, 92.8% success rate

Vertex offers the fastest time to first token (1.7s vs 2.2s for Anthropic) but has the lowest success rate (92.8% vs 98.5%). This speed-reliability trade-off creates meaningful routing decisions

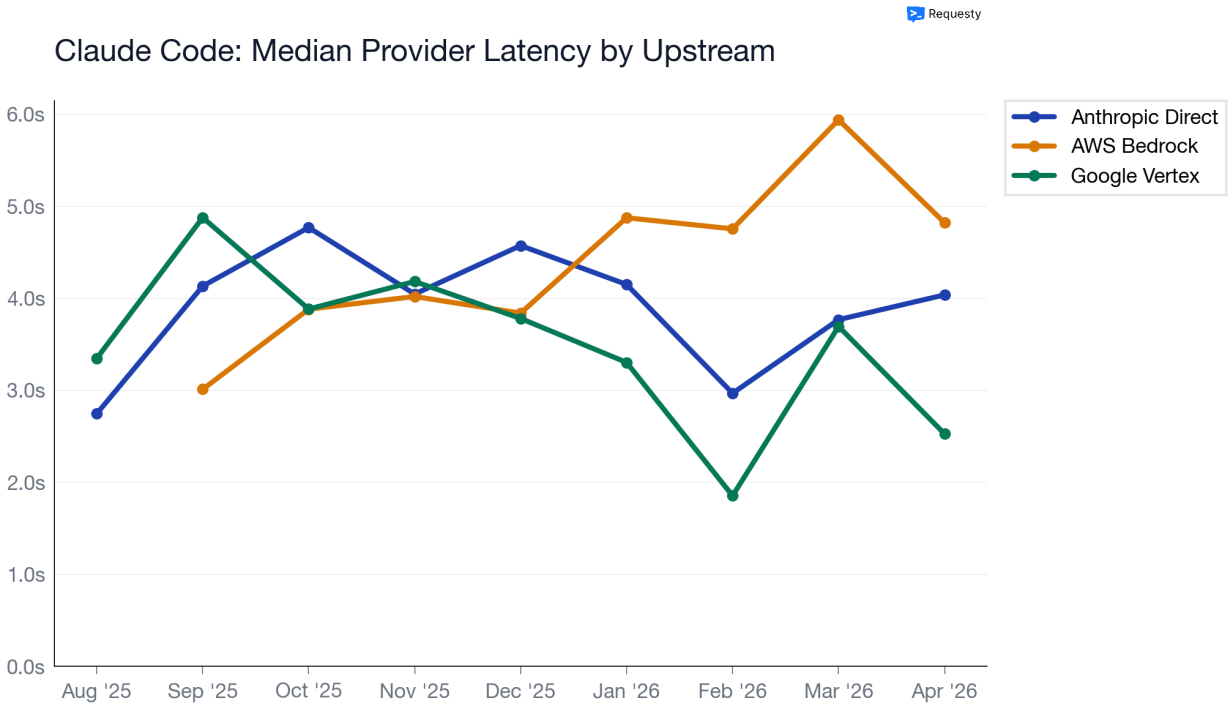

Figure 9. Claude Code: weighted median provider latency by upstream (Aug 2025 to Apr 2026). Anthropic direct and Bedrock track closely in the 5 to 7 second range, while Vertex has historically been faster but more volatile.

for latency-sensitive workloads, particularly given that a typical Claude Code session involves 50 to 200 API calls (Section 5).

6.2 Time to First Token by Model

The TTFT differences between Sonnet and Opus are substantial. Opus takes 2.0 to 2.5s for first token across providers, while Sonnet ranges from 1.4s to 2.1s. For interactive coding workflows where responsiveness matters, this latency gap influences model selection beyond raw capability.

Bedrock adoption has grown from negligible to $2.9K GMV in three months, and cache hit rates on Bedrock (94.6%) now exceed Anthropic direct (92.5%) for Claude Code. This suggests that enterprise users routing through their cloud provider achieve comparable or better caching performance.

7 Cache-Cost Correlation

The relationship between caching efficiency and per-call cost is strong and directional. **Every agent with a cache hit rate above 80% has a per-call cost below $0.10.** This is not simply because cheaper calls happen to be cached; rather, caching reduces the effective input cost by 90% for cached tokens, making the total call cost a fraction of what it would be at list price.

The cache evolution data reveals that caching improvement is not automatic. It requires deliberate architectural investment in context management. Claude Code's trajectory from 40% to 92% over nine months reflects sustained engineering effort to maximize prefix reuse. Kilo Code's plateau at 46% reflects different prompt construction patterns with smaller context windows.

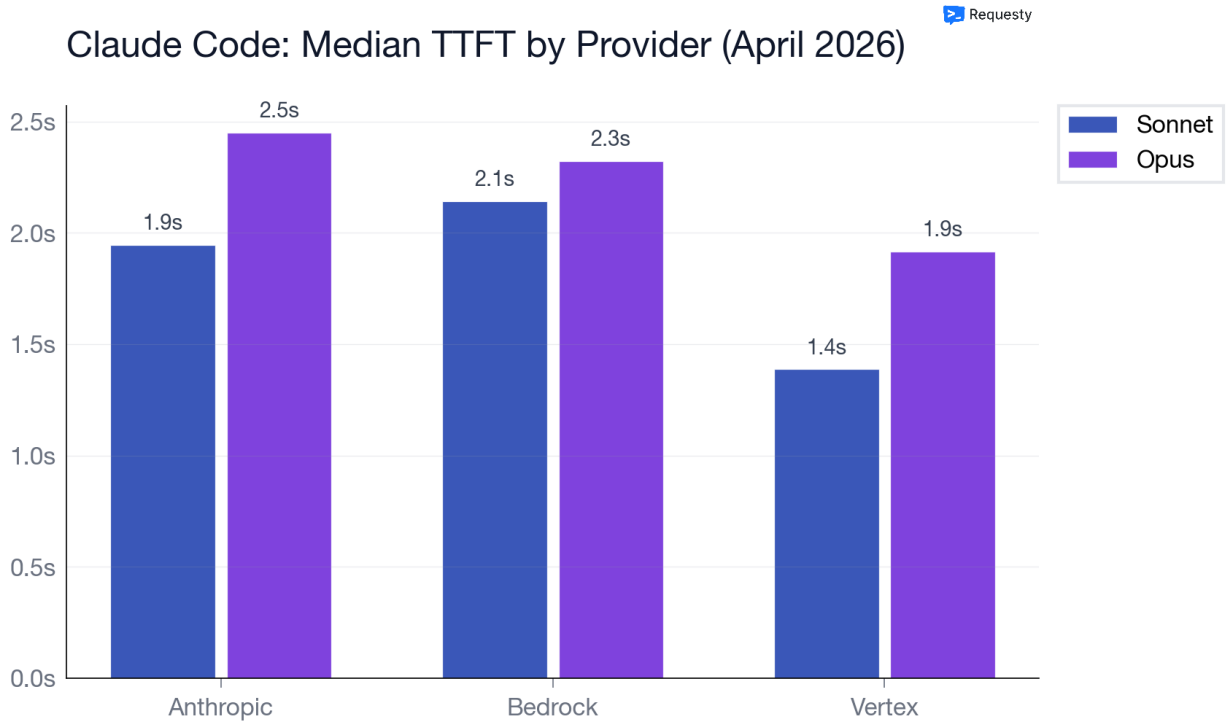

Figure 10. Claude Code: median TTFT by provider and model family (April 2026). Opus is consistently slower to produce the first token across all providers, reflecting its larger model size and reasoning overhead.

The economic implications are significant. At 92% cache hit, Claude Code pays approximately $0.30 per million input tokens effective rate (vs. $3.00 list price for Sonnet). At 46%, Kilo Code pays approximately $1.62 per million input tokens. This 5.4x cost difference compounds across every call in every session, and is a key factor in the per-user cost divergence documented in Section 2.

8 Discussion

8.1 The Caching Moat

Our data reveals that **caching architecture is the primary determinant of coding agent economics**. The difference between 92% cache hit rate (Claude Code) and 46% (Kilo Code, healthy users) translates to roughly 5.4x difference in effective input token cost. Agents that invest in maintaining consistent context prefixes across sequential calls achieve a compounding advantage: lower per-call costs enable more frequent calls, which in turn enable more sophisticated multi-step workflows, which justify higher per-user prices.

This suggests that prompt caching [11] is not merely an optimization but a structural moat. Agents built around caching can offer capabilities that would be cost-prohibitive without it.

8.2 Why Claude Won the Coding Market

The near-total consolidation to Claude models (92% of spend) across tools whose users freely choose their backend model is striking. We hypothesize several contributing factors:

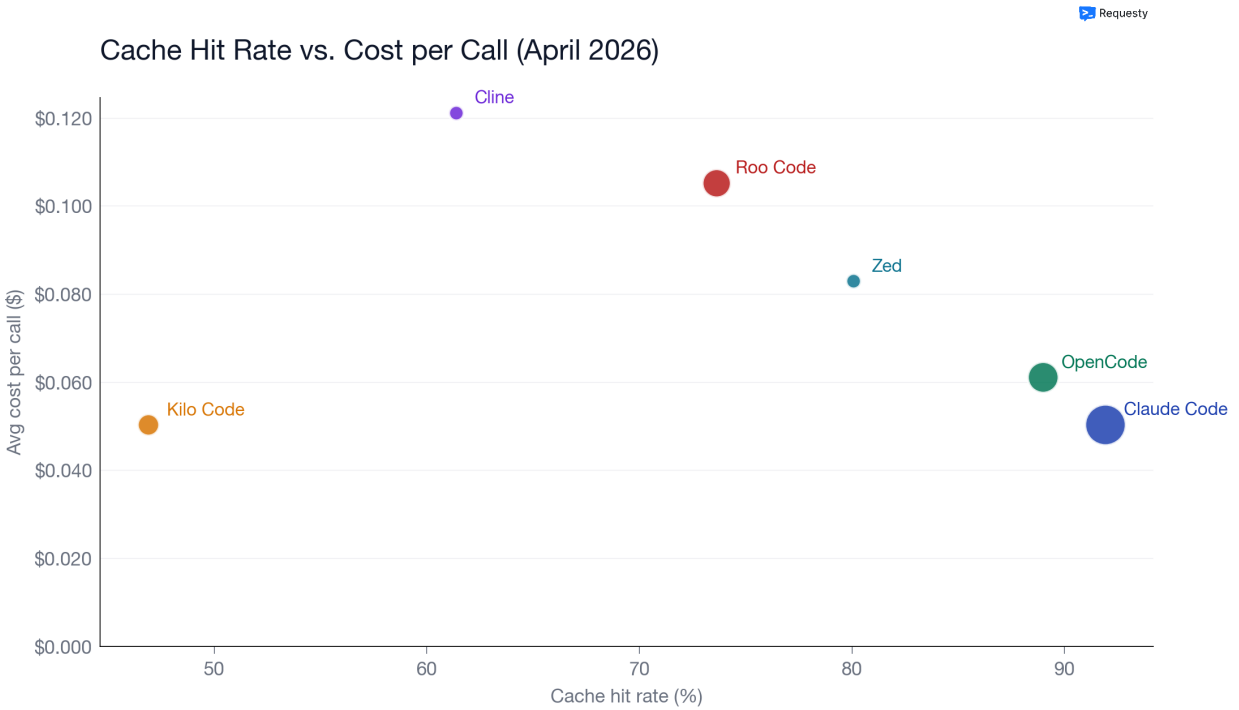

Figure 11. Cache hit rate vs. average cost per call by agent (April 2026). Agents with higher cache efficiency achieve lower per-call costs. Claude Code achieves both the highest cache rate (92%) and the lowest per-call cost ($0.050).

1. **Prompt caching alignment.** Claude's caching implementation appears particularly well-suited to the agentic coding pattern of repeated large contexts with small deltas [11].
2. **Tool use capabilities.** Claude's structured tool use enables the read-plan-edit-test loop that defines modern coding agents [1, 15].
3. **Context window size.** Claude's 200K token window accommodates large projects without requiring context management heuristics [12].
4. **Network effects.** As open-source agents optimize their prompts and workflows for Claude, switching costs increase for users of those agents.

Notably, price does not appear to be the primary factor. Gemini models are often cheaper per token [14], yet their share has declined from 34% (May 2025 via Cline) to under 6% across all agents.

8.3 The Cost Paradox

Per-user costs have nearly quadrupled while per-call costs remained stable. This apparent paradox resolves cleanly: users are making more calls, not more expensive calls. The 3.8x increase in per-user spend closely tracks the growth in average calls per user over the same period, while caching improvements have kept individual call costs flat despite 68% larger contexts.

This has implications for pricing strategy. Per-token pricing penalizes the agentic pattern (many calls, large contexts) even when caching makes the marginal cost low. Agents that can pass caching savings through to users may gain adoption advantages.

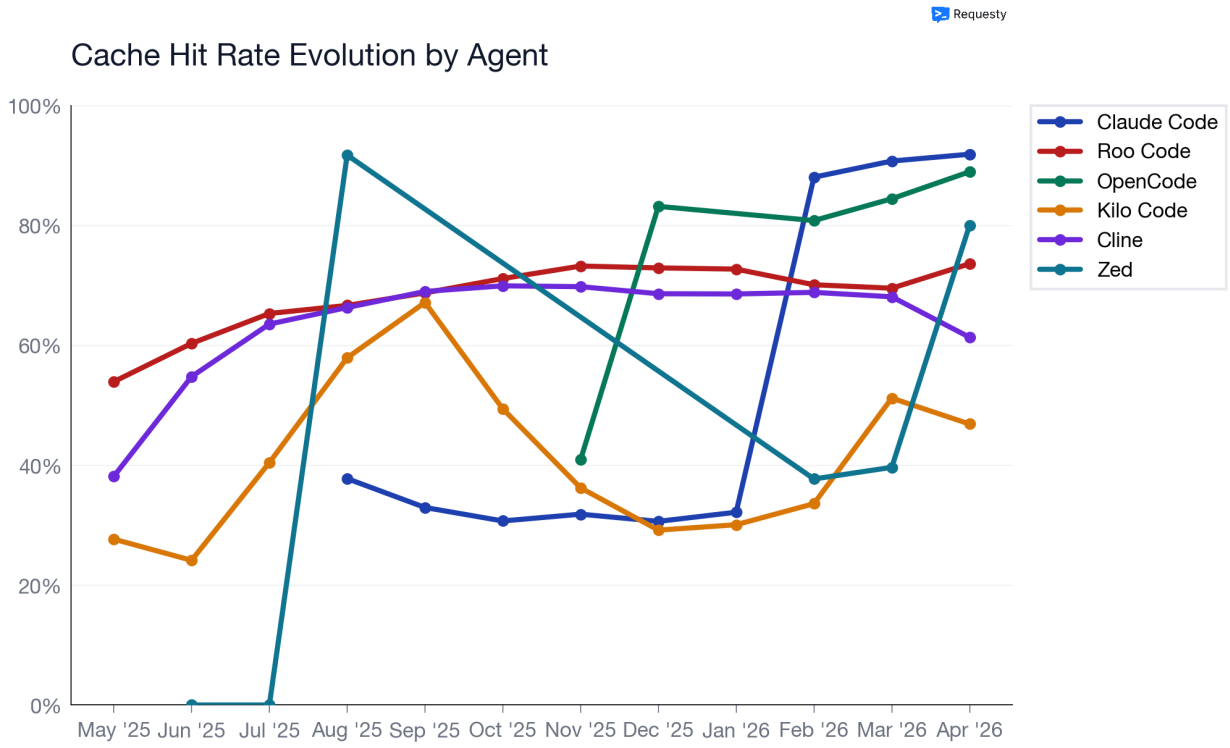

Figure 12. Cache hit rate evolution by agent over time. Claude Code's cache rate rose from 40% (Aug 2025) to 92% (Apr 2026). Kilo Code sits at 46%, reflecting different prompt construction patterns.

8.4 Provider Diversification

The growth of Bedrock as an alternative to Anthropic direct for Claude models (from \$0 to \$2.9K/month GMV for Claude Code alone) points toward a maturing market where enterprises prefer to route through their existing cloud provider relationships. With cache hit rates on Bedrock (94.6%) now exceeding Anthropic direct (92.5%), the performance penalty for indirect routing has largely disappeared.

9 Conclusion

The coding agent market has undergone rapid structural evolution over twelve months. Average per-user costs have nearly quadrupled, yet per-call costs have remained stable due to caching improvements. The per-user increase is driven primarily by longer, more intensive agentic sessions. Model choice has converged overwhelmingly toward Claude, not through lock-in but through revealed preference across open-source tools. Caching has emerged as the defining economic lever, creating up to 5.4x cost advantages for agents that optimize context reuse.

Provider latency analysis reveals that Bedrock and Vertex offer competitive alternatives to Anthropic direct, with Vertex leading on TTFT and Bedrock matching on cache efficiency.

These findings suggest that the coding agent market is still in an early expansion phase where usage intensity continues to grow as agents become more capable. The interplay between rising usage intensity, improving cache efficiency, and provider diversification will determine whether per-user costs stabilize or continue their upward trajectory.

A Data and Methodology

A.1 Data Source

Our dataset comprises anonymized, aggregated monthly metrics from the Requesty production gateway. Requesty is an LLM routing platform that sits between coding agents and upstream model providers (Anthropic, OpenAI, Google, DeepSeek, and others), recording per-request metadata including cost, token counts, latency, and model selection.

A.2 Scope

The dataset covers twelve complete calendar months from May 1, 2025 through April 30, 2026. Nine coding agents are identified via request headers and origin metadata. Not all agents were present for the full period; Claude Code [1] first appeared in August 2025, OpenCode [5] in November 2025, and Codex CLI [9] in March 2026.

A.3 Unit of Analysis

The unit of analysis throughout this paper is a *user*, identified by a unique API routing key. We report only averages, medians, percentiles, and ratios. We do not report absolute user counts, total token volumes, or total spend.

A.4 Agent Identification

Agents were identified using two methods depending on data availability. From May 2025 through January 2026, identification relied on request origin identifiers and referral metadata. From February 2026 onward, structured client identifiers in request headers supplemented these fields for improved coverage.

A.5 Limitations

Our sample reflects users who route through Requesty and is subject to selection bias. Several agents in our dataset (notably Cursor [7] and GitHub Copilot [8]) primarily connect directly to model providers rather than through third-party gateways, so their representation here is minimal and not indicative of their actual market position. Additionally, Anthropic reports zero reasoning tokens for Claude models even when extended thinking is enabled, so reasoning token comparisons across providers should be interpreted with caution.

Contributions

This research was conducted by **Thibault Jaigu** (Requesty), who designed the analysis, wrote the ClickHouse queries, and authored the paper. Data was sourced from the Requesty production gateway. The interactive web version, including all charts and the dedicated research page, was built by the Requesty team.

Correspondence: thibault@requesty.ai

References

[1] Anthropic. *Claude Code: Agentic Coding Tool*. 2026. https://code.claude.com

[2] Roo Code. *Roo Code: AI Coding Agent for VS Code*. 2025. https://github.com/RooVetGit/Roo-Code

[3] Cline Bot Inc. *Cline: Autonomous Coding Agent SDK, IDE Extension, and CLI*. 2026. https://github.com/cline/cline

[4] Kilo Code, Inc. *Kilo Code: Open Source AI Coding Agent for VS Code, JetBrains, and CLI*. 2026. https://kilo.ai

[5] OpenCode. *OpenCode: Terminal-based AI Coding Agent*. 2025. https://github.com/opencode-ai/opencode

[6] Zed Industries. *Zed: A High-Performance Code Editor with AI Integration*. 2026. https://zed.dev

[7] Anysphere. *Cursor: The AI Code Editor*. 2026. https://cursor.com

[8] GitHub. *GitHub Copilot*. 2026. https://github.com/features/copilot

[9] OpenAI. *Codex CLI: Lightweight Coding Agent*. 2026. https://github.com/openai/codex

[10] Gauthier, P. *Aider: AI Pair Programming in Your Terminal*. 2025. https://github.com/Aider-AI/aider

[11] Anthropic. *Prompt Caching with Claude*. 2025. https://claude.com/blog/prompt-caching

[12] Anthropic. *Claude Sonnet 4.6*. 2026. https://claude.com/product/overview

[13] OpenAI. *Introducing o3 and o4-mini*. 2025. https://openai.com/index/introducing-o3-and-o4-mini/

[14] Google DeepMind. *Gemini 2.5 Pro*. 2025. https://deepmind.google/technologies/gemini/

[15] Yang, J. et al. "SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering." *arXiv preprint arXiv:2405.15793*, 2024.

[16] Cognition. *Devin: AI Software Engineer*. 2024. https://www.cognition.ai/blog/introducing-devin

[17] Ziegler, A. et al. "Productivity Assessment of Neural Code Completion." *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, 2022.