Rillor

METHODS NOTE

What an agent costs per finished task

Agent cost per task, worked out from public rate cards: tokens per step, steps per task, cache discounts, tool fees and the extra attempts that failures add.



Rillor

Contents

01 Summary

02 Background

03 Method

04 Findings

05 Implications

06 Limits of this analysis

07 Sources

Summary

An agent's cost per finished task is the cost of one attempt, summed step by step from a dated rate card, divided by the share of attempts that finish the task [1].

- Each step of an agent loop is a new model call that bills the conversation so far as input again [2, 3]. In a 10-step example worked here at $2 input and $10 output per million tokens, input was 90.6% of the cost, and 20 steps used 3.1 times the input tokens of 10 (this note's arithmetic) [4].
- Prompt caching bills a repeated prompt prefix at a fraction of the input price. On rate cards viewed 8 November 2026, cache reads cost 0.025 to 0.1 times base input, and caching cut a 10-step example attempt at $2 input and $10 output per million tokens from $0.639 to $0.192 in this note's arithmetic, 70% less [4, 5, 6, 7].
- Tool definitions and tool results are billed as input on every call that carries them. A model developer counted 58 tools taking about 55,000 tokens before a conversation began [8], and each family's current web search tool costs $10 to $14 per 1,000 searches [4, 5, 6].
- Cost per finished task divides the cost of one attempt by the success rate, assuming independent attempts whose failures are detected. In one open-weight model's March 2026 submission, one-try success rates of 9.79% (banking, re-graded July 2026) to 97.81% across four task domains mean 1.02 to 10.2 attempts per success under that assumption [1, 9].
- Failed attempts tend to cost more than successful ones. In the 2024 SWE-agent study, successful runs had a median cost of $1.21 over 12 steps and unsuccessful runs a mean of $2.52 over 21 steps, and the paper's headline cost averages over successful runs only [10].
- Design multiplies tokens. One model developer measured its agents at about 4 times the tokens of a chat and its multi-agent research system at about 15 times, and a 2026 study counted 26.1 to 44.3 turns per task for coordinated multi-agent designs against 7.2 for one agent [11, 12].
- Token prices fall at fixed performance, by 9x to 900x a year in Epoch AI's 2025 analysis, and list prices also rise: one rate card lists a model's $2 and $10 per million input and output tokens doubling from 1 December 2026 [5, 13].

Background

Why an agent task costs more than a chat reply

A chat reply is one model call. An agent calls the model, runs a tool, adds the result to the conversation and calls the model again. One provider's caching guide lists agentic tool use as a case "where each step typically requires a new API call" [2]. Another provider's guide to conversation state says that even when a call points to the previous response, "all previous input tokens for responses in the chain are billed as input tokens in the API" [3]. Every step pays again for the instructions, the tool definitions and everything read so far.

A model developer reported in June 2025 that its agents "typically use about 4× more tokens than chat interactions", and its multi-agent systems about 15 times more, a measurement of its own research system [11]. A 2026 preprint on eight models working SWE-bench Verified found agentic coding tasks "consuming 1000x more tokens than code reasoning and code chat", with input tokens driving the overall cost more than output tokens [14]. The same developer's guide to building agents puts the trade in one line: "Agentic systems often trade latency and cost for better task performance" [15].

Why cost per task gets left out

In 2024 the authors of "AI Agents That Matter" found "a narrow focus on accuracy without attention to other metrics" in agent benchmarks and argued for "jointly optimizing" cost and accuracy [16]. Cost-of-pass, a framework first posted in 2025, defines "the expected monetary cost of generating a correct solution" [1]. The ARC Prize leaderboard calls the relationship between cost per task and performance "a key measure of efficiency" [17], and the Holistic Agent Leaderboard (HAL) states that "Agents can be 100x more expensive while only being 1% better" [18]. Reporting is uneven: a March 2026 τ-bench leaderboard submission leaves its cost field empty in every domain [9].

This note is for finance and engineering leads who need an estimate of cost per finished task before an agent is built, and who want to know which design choices move it.

Method

What was examined. The billing rules that set the cost of one agent attempt on the public rate cards of three commercial model families, viewed 8 November 2026: input, output and reasoning tokens, cache writes and reads, batch discounts, tool fees and runtime [4, 5, 6]. Beside them, published measurements of tokens, steps, success rates and cost in agents.

Sources. Thirty-four sources, each opened on 8 November 2026 ([4] and [17] again on 13 November 2026): eleven provider documentation and pricing pages, five engineering posts from one model developer, eleven papers and preprints, three leaderboards, one architecture guide and three trend analyses.

Families. Families A, B and C are three commercial model families. Providers appear in the source list only. The worked examples use one model per family listed at $2 per million input tokens and $10 per million output tokens, plus one smaller Family C model [4, 5, 6], so the billing rules meet one token profile. No provider or model is ranked.

How. Each price was read with its tier, cache rules and any dated change. Usage follows the providers' definitions, total input being cache reads plus cache writes plus uncached input [2]. Figures marked "this note's arithmetic" are computed here from the sources' own numbers.

The formula. For one attempt of n steps:

- Cost per attempt = sum over steps of (uncached input × input price + cache writes × write price + cache reads × read price + output × output price) + per-call tool fees + runtime
- Cost per finished task = cost per attempt / p
- Cost per finished task, failures priced apart = c_s + c_f × (1 - p) / p
- Input tokens over n steps = n × (S + U) + d × n × (n - 1) / 2

p is the share of attempts that finish the task, and c_s and c_f the costs of a successful and a failed attempt. S is the fixed prompt sent on every step (instructions and tool definitions), U the task's own input and d the tokens each step adds, its output plus the tool result. Output includes reasoning tokens, which all three families bill as output [5, 19, 20]. Some providers bill tools per call and runtime per hour [4, 5, 6, 21].

The second line is cost-of-pass. Erol and colleagues define it as the expected cost of one attempt divided by the probability of a correct answer, note that the expected number of attempts to the first success is 1/p "assuming independent trials", and state that the cost becomes infinite when p is zero [1]. The third line follows from the same definition when a failed attempt costs a different amount: the successful attempt costs c_s, and (1 - p) / p failures are expected before it (this note's algebra). The fourth line counts the context resent at each step (this note's algebra).

Assumptions. Attempts are independent and share one success rate, every failure is detected and retried, and success is measured per completed task. Finding 4 tests the first against published repeated trials.

Where sources disagree. One preprint's abstract gives 1000x where its body gives 3500x and 1200x against two baselines [14]. A 2026 scaling study calls overheads of 1.6 to 6.2 times token budgets, yet its systems were matched on reasoning tokens and the figures equal its turn ratios, so this note cites turns [12].

Findings

1. Tokens per step and steps per task

Steps. In a study of deployed agents that combined 20 case studies with a survey of 86 practitioners, the authors report that "68% execute at most 10 steps before human intervention" [22]. A model developer's rules for its research agents give 1 agent with 3 to 10 tool calls for simple fact-finding, 2 to 4 subagents with 10 to 15 calls each for direct comparisons, and more than 10 subagents for complex research [11]. In SWE-agent's 2024 study, successful runs finished in a median of 12 steps and unsuccessful ones in a mean of 21 [10].

Tokens per step. Family A's guide says reasoning models "may generate anywhere from a few hundred to tens of thousands of reasoning tokens" depending on difficulty, bills them as output and recommends reserving at least 25,000 tokens for reasoning and output when first trying them [19]. If that cap is hit early, a call can "incur costs for input and reasoning tokens without receiving a visible response" [19]. Family C labels its output prices "including thinking tokens" [5], and Family B reports how many of the billed output tokens were internal reasoning [20]. Epoch AI found reasoning models producing "around 8x more tokens on average" than other models, their responses growing about 5 times a year against 2.2 times, from 11 reasoning models and with a wide interval of 2 to 13 on the yearly growth [23].

Spread between runs. On SWE-bench Verified, "runs on the same task can differ by up to 30x in total tokens", a maximum, while for each model the typical spread between runs is about 2 times [14]. The same study found that "higher token usage does not translate into higher accuracy", and that frontier models asked to predict their own usage "systematically underestimate real token costs", with correlations of at most 0.39 [14].

Per task, across benchmarks. HAL's October 2025 paper ran 21,730 agent rollouts on 9 models and 9 benchmarks for about $40,000 and about 2.5 billion tokens [24], an average of about $1.84 and 115,000 tokens per rollout over very different benchmarks (this note's arithmetic).

Units. Family B's newer tokenizer "produces approximately 30% more tokens for the same text", by an amount that depends on content [4], so a token profile measured on one model version needs a recount on another.

2. Context growth and prompt caching

Growth. Because each step resends the conversation, input over an attempt grows with the square of the step count, as the fourth line of the formula shows. Newer Family B models also keep prior turns' thinking blocks in context and bill them as input [20]. The profile used in this note: S = 12,000 tokens (instructions, about 10 tool definitions at about 950 tokens each [8] and a 286-token tool-use system prompt [4]), U = 3,000, 600 output tokens a step including reasoning, and a 2,500-token tool result a step, the size one rate card gives for an average web page [4].

Steps	Context at the last step	Input tokens over the attempt	Output tokens	Input against 10 steps
10	42,900	289,500	6,000	1.0x
20	73,900	889,000	12,000	3.1x
40	135,900	3,018,000	24,000	10.4x

This note's arithmetic from [4, 8].

At $2 input and $10 output per million tokens, the 10-step attempt costs $0.639 without caching, and input is 90.6% of that (this note's arithmetic). Family A raises its prices above 272,000 input tokens [6], and one smaller Family B model above 100,000 [4]. The profile stays below both.

Caching rules. Prompt caching stores a prompt's prefix so later calls read it at a discount.

Rule	Family A	Family B	Family C
Default	On for supported models [7]	Breakpoints, or automatic caching that moves forward as the conversation grows [2]	Implicit caching on by default for current models [25]
Cache read	0.1x input on most models, 0.05x on two [7]	0.1x standard, 0.05x on two current models including the one priced here, 0.025x on four others [4]	$0.10 against $2 input on the model priced here [5]
Cache write	1.25x for 30 minutes, 2.5x for 12 hours [7]	1.25x for 5 minutes, 2x for 1 hour [4]	No write price listed [5]
Minimum prefix	1,024 tokens [7]	512 tokens on current models [2]	4,096 tokens on current models [25]

Family B's page says caching saves money "after a small number of cache reads", one for the 5-minute duration (1.25x write) and two for the 1-hour duration (2x write), and that the multipliers stack with batch and data residency pricing [4]. Its cache is refreshed at no cost each time it is used [2]. Family C states that it passes on savings "if your request hits caches", gives no assurance that a request hits, and for small additions on the model priced here re-processes the last 2,050 tokens of the previous request at the standard input rate [25].

What breaks a cache. Family B builds cache prefixes in the order tools, system, messages, and "Modifying tool definitions (names, descriptions, parameters) invalidates the entire cache" [2]. Changing the thinking budget between requests invalidates cache breakpoints too [20]. Family A requires the entire rendered prefix to match and advises putting stable instructions and shared reference material first [7], and Family C gives the same advice [25].

The 10-step attempt, cached. With each step reading the previous context from cache and writing only its new tokens, the attempt reads 246,600 tokens and writes 42,900 (this note's arithmetic). At Family B's listed $0.10 a million for reads and $2.50 for 5-minute writes on the model priced here, reads cost $0.025, writes $0.107 and output $0.060, a total of $0.192, 70.0% below the uncached $0.639 (this note's arithmetic from [4]).

Cache reads at scale. In coding-agent traces for one model family, cache-read input tokens made up more of the usage than any other category, and "cheap-per-token cache reads still outweigh expensive-per-token output in aggregate" [14]. Context editing, which clears stale tool calls and results, cut token use by 84% in one developer's 100-turn evaluation [26], and Family A offers server-side compaction [3].

3. Tool definitions, tool results and per-call fees

Definitions. Tool definitions are billed as input on every call that includes them [4]. One developer counted a five-server setup at "58 tools consuming approximately 55K tokens before the conversation even starts", about 950 tokens per tool (this note's arithmetic), and has seen definitions reach 134,000 tokens [8]. Its tool search feature cut one setup from about 77,000 tokens to about 8,700, an 85% reduction [8]. Family B adds a tool-use system prompt of 286 to 804 tokens a request, depending on model and tool choice, plus 325 tokens for its bash tool and 974 for its text editor tool, and its computer and browser toolsets add about 4,500 and 6,600 tokens, that system prompt included [4].

Results. "Every intermediate result must pass through the model", and a 2-hour meeting transcript "could mean processing an additional 50,000 tokens" [27]. One rate card sizes fetched content at about 2,500 tokens for an average web page, 25,000 for a large documentation page and 125,000 for a research paper PDF, and offers a setting to cap it [4]. Letting the agent write code that calls tools, loading only the definitions a task needs and keeping intermediate results out of the context, cut one example from 150,000 tokens to 2,000, a saving of 98.7%, at the price of a sandbox with its own operational overhead [27].

Fees. Some tools carry charges on top of tokens.

Charge	Listed price, viewed 8 November 2026	Source
Web search, Families A and B, current tools	$10 per 1,000 searches or calls, plus the search content as tokens	[4, 6]

Charge	Listed price, viewed 8 November 2026	Source
Search grounding, Family C, current models	5,000 free requests a month, then $14 per 1,000, each search query billed	[5]
File search, Family A	$2.50 per 1,000 calls, storage $0.10 per GB a day after 1 GB free	[6]
Hosted containers, Family A	$0.03 per 20-minute session for 1 GB	[6]
Code execution, Family B	1,550 free hours per organization a month, then $0.05 per container-hour, 5-minute minimum	[4]
Managed agent sessions, Family B	$0.08 per session-hour while running, plus tokens, no batch mode	[4]
One cloud provider's agent runtime	v1: $0.0895 per vCPU-hour and $0.00945 per GB-hour. v2: $0.1276 and $0.0169. CPU at zero during I/O wait	[21]

Family B does not bill a web search that fails with an error [4]. The cloud provider's own example prices the v2 runtime of a 10-minute customer support session, 90% of it waiting on I/O, at $0.006703 [21]. For the 10-step profile, five web searches at $10 per 1,000 add $0.05 to $0.192 of cached tokens, about 26% (this note's arithmetic), before the search content enters the context.

4. Retries and cost per completed task

The division. Cost-of-pass divides the expected cost of one attempt by the success rate, with 1/p attempts expected per success under independent trials [1]. At the one-try success rates of one open-weight model in a March 2026 τ-bench leaderboard submission, evaluated with four trials per task, that means 1.02 attempts per success in telecom (97.81%), 1.18 in retail (84.43%), 1.23 in airline (81.5%) and 10.2 in a banking knowledge domain (9.79%, re-graded on 15 July 2026) (this note's arithmetic from [9]). On the cached 10-step attempt of $0.192, cost per finished task is $0.196, $0.227 and $1.96 at 97.81%, 84.43% and 9.79% (this note's arithmetic).

Pooled success rates. pass^k, introduced with τ-bench in 2024, is the share of tasks solved on all k of k trials [28]. If every task had the same one-try rate, retail's 84.43% would predict 0.8443^4 = 50.8% success on all four trials, against 59.65% measured, and airline's 81.5% would predict 44.1% against 68.0% (this note's arithmetic from [9]). The measured rates are higher than one shared rate predicts, which fits some tasks being solved on every try and others failed on every try (this note's reading). On such a failing task a retry rarely rescues the attempt, so the cap on attempts sets what it spends (this note's reading). In 2024, τ-bench found pass^8 under 25% in retail [28].

Failures cost more. SWE-agent's authors found that agents succeed quickly and fail slowly: successful runs "finish with a median cost of $1.21 and 12 steps", unsuccessful runs at "a mean of $2.52 and 21 steps" [10]. The comparison sets a median against a mean, so the ratio of 2.08 is indicative (this note's arithmetic). The 2026 token study also found models consuming more tokens on a shared failure subset than on a shared success subset [14]. With failed attempts priced at 2.08 times a successful one, the third line of the formula gives $0.266 per finished task at 84.43%, $0.591 at 50% and $3.87 at 9.79% (this note's arithmetic).

What published averages count. SWE-agent's main table reports 12.47% resolved on full SWE-bench at an average cost of $1.59, averaged "over all successfully resolved instances" [10], which leaves out the failed attempts that cost more.

Caps. SWE-agent set "the per-instance budget to $4", and 93.0% of resolved instances were submitted before exhausting it, against 69.0% of all instances [10]. A model developer notes that agents commonly include stopping conditions "such as a maximum number of iterations" [15]. A cap bounds c_f.

Retries and accuracy. In the words of "AI Agents That Matter", "Accuracy alone cannot identify progress because it can be improved by scientifically meaningless methods such as retrying" [16]. Cost-of-pass found that inference-time methods such as majority voting and self-refinement, when performance-oriented with marginal gains, "rarely justify the costs" [1].

Undetected failures. A failure nobody catches adds no retry, so the formula then understates the cost of a correct result. The production study reports that "74% depend primarily on human evaluation" [22].

5. Token multiples for single and multi-agent designs

One developer's multiples. The 4x and 15x figures come from one model developer's research system, measured against chat [11]. Their ratio puts the multi-agent design at about 3.75 times the tokens of a single agent in that system (this note's arithmetic). The developer found three factors explained 95% of the performance variance in its BrowseComp evaluation, with "token usage by itself" explaining 80% and the number of tool calls and the model choice the rest [11]. It reported a 90.2% gain over a single agent on an internal research evaluation, and concluded that multi-agent systems need tasks "where the value of the task is high enough to pay for the increased performance" [11]. Applied to the cached 10-step attempt, 3.75 times gives about $0.72 an attempt if the token mix stayed the same (this note's arithmetic, illustrative only).

Overheads. One cloud provider's architecture guidance says costs rise "because each agent processes redundant context and communication overhead multiplies with agent count", and advises teams to "Validate cost benefits through prototyping before adding orchestration overhead" [29]. A 2026 scaling study matched systems at a mean of 4,800 reasoning tokens per trial and counted turns per task: 7.2 for a single agent, 11.4 for independent agents, 26.1 decentralized, 27.7 centralized and 44.3 hybrid [12]. Against one agent, multi-agent performance ranged from +80.8% on decomposable financial reasoning to -70.0% on sequential planning, and tool-heavy tasks "appear to incur multi-agent overhead" [12]. In 30 software tasks run by the

ChatDev framework, code review took 59.4% of tokens on average [30]. In one author's preliminary experiments, compiling a multi-agent system into one agent with skills "can substantially reduce token usage and latency while maintaining competitive accuracy on reasoning benchmarks" [31].

Simpler designs. On HumanEval in 2024, a warming strategy scored 93.2% at $2.45 for one pass over 164 problems, against 88.0% at $134.50 for LATS [16], about $0.015 and $0.82 a problem at 2024 prices (this note's arithmetic). The authors write that "For substantially similar accuracy, the cost can differ by almost two orders of magnitude" [16]. On GAIA, Efficient Agents kept 96.7% of the performance of OWL, an open-source agent framework, while cutting cost from $0.398 to $0.228, a 28.4% improvement in cost-of-pass, in work its authors label in progress [32].

6. Worked examples from rate cards viewed 8 November 2026

The 10-step profile, 289,500 input and 6,000 output tokens with 246,600 of the input readable from cache, priced on the rate-card rows below.

Rate card row	Input / cache read / output, $ per million	Uncached attempt	Cached attempt
Family B model, 5-minute writes at $2.50	$2 / $0.10 / $10	$0.639	$0.192
Family B model, 1-hour writes at $4	$2 / $0.10 / $10	$0.639	$0.256
Family B model, US-only inference at 1.1x	$2.20 / $0.11 / $11	$0.703	$0.211
Family A model, 30-minute writes at $2.50	$2 / $0.10 / $10	$0.639	$0.192
Family C model, through 30 November 2026	$2 / $0.10 / $10	$0.639	$0.171
Family C model, from 1 December 2026	$4 / $0.20 / $20	$1.278	$0.341
Family C smaller model, through 31 December 2026	$0.75 / $0.075 / $3.75	$0.240	$0.073
Family C smaller model, from 1 January 2027	$1.50 / $0.15 / $7.50	$0.479	$0.146

This note's arithmetic from [4, 5, 6]. Family C's cached figures assume every prefix hits the cache, with new tokens at the input price and no write charge.

Families A and B match because the models priced here list the same four rates [4, 6]. Family C's cached figure is lower because its page lists no cache-write price [5], and it is a floor, with no assurance of a hit [25]. Family B's 1.1x for US-only inference applies to every token category [4], and Family A adds 10% for regional processing on models released on or after 5 March 2026 [6].

Batch. All three families sell batch processing at half price [4, 5, 6]. Family B's batch documentation describes it as suited "to tasks that do not require immediate responses", with most batches finishing in under an hour and results available within 24 hours [33]. Family B's managed agent sessions have no batch mode, because "Sessions are stateful and interactive" [4].

From attempt to task. Add tool fees and runtime from finding 3, then divide by the success rate from finding 4. For the Family B cached attempt with five web searches, $0.242 an attempt becomes $0.287 per finished task at 84.43% success with equal attempt costs (this note's arithmetic).

7. Price trends and what they leave out

Falling prices per token. Epoch AI found the price of reaching a fixed performance level falling 9x to 900x a year depending on the milestone, and 40x a year for an early-2023 flagship's performance on PhD-level science questions [13]. The authors caution that the steep drops came in the year before their March 2025 analysis and may not persist [13]. Stanford's 2025 AI Index reports that inference cost for a system at a 2022 chat model's level fell over 280-fold from November 2022 to October 2024 [34]. Cost-of-pass found the frontier cost of a correct answer on complex quantitative tasks "roughly halved every few months" [1].

Rising tokens per task. Reasoning responses grew about 5 times a year in Epoch AI's sample [23], and Family B's newer tokenizer counts about 30% more tokens for the same text [4]. Cost per task falls only where the price per token falls faster than tokens per task rise.

Rising list prices. Prices also move up on dated schedules. Family C's page lists doublings on 1 December 2026 and 1 January 2027, and from 1 December 2026 routes requests for older models to newer ones, billed at the newer models' rates [5]. Family A's page calls one model's current price promotional until 30 November 2026 [6]. Family B's page states that a rise for one model, scheduled for 1 September 2026, was dropped [4].

More tokens, uneven accuracy. HAL found "higher reasoning effort reducing accuracy in the majority of runs" [24], and the 2026 token study found that accuracy "often peaks at intermediate cost and saturates at higher costs" [14]. A bigger token budget raises the cost of an attempt with no reliable gain in p.

Implications

Measure on your own tasks before quoting a cost. Tokens per step, steps per task and the success rate decide the estimate, and runs of one task differ by up to 30 times in tokens [14]. Log input, cache, output and reasoning tokens beside dollars. "AI Agents That Matter" asks evaluations to report input and output token counts in addition to dollar costs [16].

Report cost over every attempt. Divide total spend, failed attempts included, by tasks finished. An average over successful runs leaves out the attempts that cost about twice as much in SWE-agent's data [10].

Keep the prefix stable. Put tool definitions and instructions first and change them rarely, since a tool definition change invalidates a whole cache and a thinking-budget change breaks cache breakpoints [2, 7, 20, 25]. Load only the tools a step needs [8].

Cap steps and spend. A maximum number of iterations and a per-task budget bound the cost of a failing attempt [10, 15].

Choose the model per task. Cost-of-pass found lightweight models ahead for basic quantitative tasks and large models for knowledge-intensive ones [1].

Separate fixed from variable cost. "AI Agents That Matter" splits the fixed cost of optimizing an agent from the variable cost of each run, which dominates with use. In its example, joint optimization gave "53% lower variable cost with similar accuracy" and broke even after about 1,350 tasks [16].

Date every price and pin the version. A cost estimate holds for one model version on one day's rate card [4, 5, 6]. On the agentic systems Rillor builds (https://rillor.com/si/agentic-systems), model families are chosen per project and versions are pinned, so every run can be replayed.

Limits of this analysis

- **Snapshot.** Prices were viewed on 8 November 2026, and several change on listed dates [5, 6].
- **Profile.** The 10-step profile is built from published sizes, not measured on a running agent [4, 8], and real attempts vary widely in tokens [14].
- **Assumptions.** The formula assumes independent attempts with one success rate for every task, and detected failures. Published repeated trials show success rates that differ between tasks, so a pooled rate understates the attempts a hard task needs (this note's reading of [9]).
- **Cache hits.** The cached figures assume every step reads the previous context from cache [2, 7, 25].
- **Multiples.** The 4x and 15x figures are one developer's measurement of its own system [11], and the 2026 token study gives its dollar costs at prices "at the time of our experiments", with no experiment date, so this note uses its token counts [14].
- **Leaderboards.** The ARC Prize table gives a cost per task for ARC-AGI-1 and ARC-AGI-2 and a total cost for ARC-AGI-3 [17], HAL's home page shows run totals without task counts [18], and the τ-bench submission reports no cost [9].
- **Out of scope.** Engineering effort, staff and review time. No source opened gives a cost of human review per task. The choice between an API and self-hosting is set out in Self-hosted LLM vs API cost: a break-even method (https://rillor.com/insights/self-hosted-llm-vs-api-cost).
- **Scope.** This is a method for engineering cost estimates. It gives no investment, accounting or procurement advice.

Sources

1. Cost-of-Pass: An Economic Framework for Evaluating Language Models. Mehmet Hamza Erol, Batu El, Mirac Suzgun, Mert Yuksekgonul and James Zou, arXiv:2504.13359, version 2, 26 February 2026 (first version 17 April 2025), viewed 8 November 2026. https://arxiv.org/abs/2504.13359

2. Prompt caching, Claude Platform documentation. Anthropic, viewed 8 November 2026. https://platform.claude.com/docs/en/build-with-claude/prompt-caching

3. Conversation state, OpenAI API documentation. OpenAI, viewed 8 November 2026. https://developers.openai.com/api/docs/guides/conversation-state

4. Pricing, Claude Platform documentation. Anthropic, viewed 13 November 2026. https://platform.claude.com/docs/en/about-claude/pricing

5. Gemini Developer API pricing, Google AI Studio documentation. Google, viewed 8 November 2026. https://aistudio.google.com/docs/pricing

6. Pricing, OpenAI API documentation. OpenAI, viewed 8 November 2026. https://developers.openai.com/api/docs/pricing

7. Prompt caching, OpenAI API documentation. OpenAI, viewed 8 November 2026. https://developers.openai.com/api/docs/guides/prompt-caching

8. Introducing advanced tool use on the Claude Developer Platform. Bin Wu, Anthropic Engineering, 24 November 2025, viewed 8 November 2026. https://www.anthropic.com/engineering/advanced-tool-use

9. τ-bench leaderboard submission qwen3.5-397b-a17b-think_sierra_2026-03-02, submission.json. Sierra, 2 March 2026, viewed 8 November 2026. https://sierra-tau-bench-public.s3.us-west-2.amazonaws.com/submissions/qwen3.5-397b-a17b-think_sierra_2026-03-02/submission.json

10. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. John Yang, Carlos E. Jimenez, Alexander Wettig and others, arXiv:2405.15793, version 3, 11 November 2024 (first version 6 May 2024), viewed 8 November 2026. https://arxiv.org/abs/2405.15793

11. How we built our multi-agent research system. Jeremy Hadfield, Barry Zhang, Kenneth Lien and others, Anthropic Engineering, 13 June 2025, viewed 8 November 2026. https://www.anthropic.com/engineering/multi-agent-research-system

12. Towards a Science of Scaling Agent Systems. Yubin Kim, Ken Gu, Chanwoo Park and others, arXiv:2512.08296, version 3, 8 April 2026 (first version 9 December 2025), viewed 8 November 2026. https://arxiv.org/abs/2512.08296

13. LLM inference prices have fallen rapidly but unequally across tasks. Ben Cottier, Ben Snodin, David Owen and Tom Adamczewski, Epoch AI, 12 March 2025, viewed 8 November 2026. https://epoch.ai/data-insights/llm-inference-price-trends

14. How Do AI Agents Spend Your Money? Analyzing and Predicting Token Consumption in Agentic Coding Tasks. Longju Bai, Zhemin Huang, Xingyao Wang and others, arXiv:2604.22750, version 3, 2 October 2026 (first version 24 April 2026), viewed 8 November 2026. https://arxiv.org/abs/2604.22750

15. Building effective agents. Erik S. and Barry Zhang, Anthropic Engineering, 19 December 2024, viewed 8 November 2026. https://www.anthropic.com/engineering/building-effective-agents

16. AI Agents That Matter. Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir and Arvind Narayanan, arXiv:2407.01502, 1 July 2024, viewed 8 November 2026. https://arxiv.org/abs/2407.01502

17. ARC Prize leaderboard. ARC Prize Foundation, rows dated to 10 November 2026, viewed 13 November 2026. https://arcprize.org/leaderboard

18. Holistic Agent Leaderboard (HAL). Princeton University, viewed 8 November 2026. https://hal.cs.princeton.edu/

19. Reasoning models, OpenAI API documentation. OpenAI, viewed 8 November 2026. https://developers.openai.com/api/docs/guides/reasoning

20. Extended thinking, Claude Platform documentation. Anthropic, viewed 8 November 2026. https://platform.claude.com/docs/en/build-with-claude/extended-thinking

21. Amazon Bedrock AgentCore pricing. Amazon Web Services, viewed 8 November 2026. https://aws.amazon.com/bedrock/agentcore/pricing/

22. Measuring Agents in Production. Melissa Z. Pan, Negar Arabzadeh, Riccardo Cogo and others, ICML 2026, arXiv:2512.04123, version 4, 4 June 2026 (first version 2 December 2025), viewed 8 November 2026. https://arxiv.org/abs/2512.04123

23. LLM responses to benchmark questions are getting longer over time. Luke Emberson, Ben Cottier, Josh You, Tom Adamczewski and Jean-Stanislas Denain, Epoch AI, 17 April 2025, viewed 8 November 2026. https://epoch.ai/data-insights/output-length

24. Holistic Agent Leaderboard: The Missing Infrastructure for AI Agent Evaluation. Sayash Kapoor, Benedikt Stroebl, Peter Kirgis and others, arXiv:2510.11977, 13 October 2025, viewed 8 November 2026. https://arxiv.org/abs/2510.11977

25. Context caching, Google AI Studio documentation. Google, viewed 8 November 2026. https://aistudio.google.com/docs/caching

26. Managing context on the Claude Developer Platform. Anthropic, Claude blog, 29 September 2025, viewed 8 November 2026. https://claude.com/blog/context-management

27. Code execution with MCP: building more efficient agents. Adam Jones and Conor Kelly, Anthropic Engineering, 4 November 2025, viewed 8 November 2026. https://www.anthropic.com/engineering/code-execution-with-mcp

28. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. Shunyu Yao, Noah Shinn, Pedram Razavi and Karthik Narasimhan, arXiv:2406.12045, 17 June 2024, viewed 8 November 2026. https://arxiv.org/abs/2406.12045

29. Choosing Between Building a Single-Agent System or Multi-Agent System, Cloud Adoption Framework. Microsoft Learn, 1 December 2025, last updated 10 December 2025, viewed 8 November 2026. https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/ai-agents/single-agent-multiple-agents

30. Tokenomics: Quantifying Where Tokens Are Used in Agentic Software Engineering. Mohamad Salim, Jasmine Latendresse, SayedHassan Khatoonabadi and Emad Shihab, arXiv:2601.14470, 20 January 2026, viewed 8 November 2026. https://arxiv.org/abs/2601.14470

31. When Single-Agent with Skills Replace Multi-Agent Systems and When They Fail. Xiaoxiao Li, arXiv:2601.04748, version 2, 14 January 2026, viewed 8 November 2026. https://arxiv.org/abs/2601.04748

32. Efficient Agents: Building Effective Agents While Reducing Cost. Ningning Wang, Xavier Hu, Pai Liu and others, arXiv:2508.02694, 24 July 2025, viewed 8 November 2026. https://arxiv.org/abs/2508.02694

33. Batch processing, Claude Platform documentation. Anthropic, viewed 8 November 2026. https://platform.claude.com/docs/en/build-with-claude/batch-processing

34. The 2025 AI Index Report. Stanford Institute for Human-Centered Artificial Intelligence (HAI), 2025, viewed 8 November 2026. https://hai.stanford.edu/ai-index/2025-ai-index-report

About this report

Published by Rillor on 14 November 2026. Superintelligence (SI) tools assist research and drafting, and every figure cites its source. Rillor builds and secures SI systems, supplies the data they learn from and delivers the infrastructure they run on. We apply SI to financial and compute markets.

Cite as: Rillor. What an agent costs per finished task. 14 November 2026. https://rillor.com/insights/agent-cost-per-task

Notices

Superintelligence (SI) outputs can be wrong. Important decisions should include human review.

Work with Rillor

Tell us the workflow, the sources it draws on and the decisions it supports, and we scope the agents, tools and tests around them. Contact us at rillor.com/contact?type=agentic-systems or contact@rillor.com. More on agentic systems: rillor.com/si/agentic-systems