

Do Tools Matter, or Just Their Descriptions?

Ananya Tomar

Manipal University Jaipur
ananyatomar2308@gmail.com

Ayush Nangia

Independent Researcher

Abstract

Tool-augmented language models are increasingly applied to financial document reasoning, a setting that demands numerical precision over long financial filings. Earlier comparisons either mix up the model with its scaffolding or report tool-use gains without separating code, strategy, and environment. We isolate these factors in a controlled same-model ablation: six configurations share an identical GPT-5 backbone and differ only in the tool surface (callable tools plus their descriptions), with one control that additionally forces sub-model delegation. On DocMath-Eval under oracle context, the fully equipped configuration beats the restricted-tool baseline by +3.6 points ($p=0.0028$). Where this gain comes from matters more than its size: the system tools that give that configuration its name are used on under 0.2% of samples, and a description-only control, which describes the same tools but never runs them, performs almost identically (within 1 point on every split). The benefit comes from the tool descriptions, not from executing the tools. On long-document splits, the equipped model uses fewer iterations, has answers available earlier when runs are cut short, and varies its tool mix with document length. Comprehension errors dominate the hardest failures: the bottleneck is document comprehension, not the tool interface.

1 Introduction

Consider a percentage-change question over a long financial filing (Figure 1). GPT-5 inside a recursive language model (Zhang et al., 2025) with only a Python REPL reaches the correct answer in six iterations, re-extracting source values through a sub-model and recomputing, while the same GPT-5 given two coded tools, *parsetable* for extraction and *verifyanswer* for self-checking, reaches it in two. Both reach the same correct answer, but along very different paths. What,

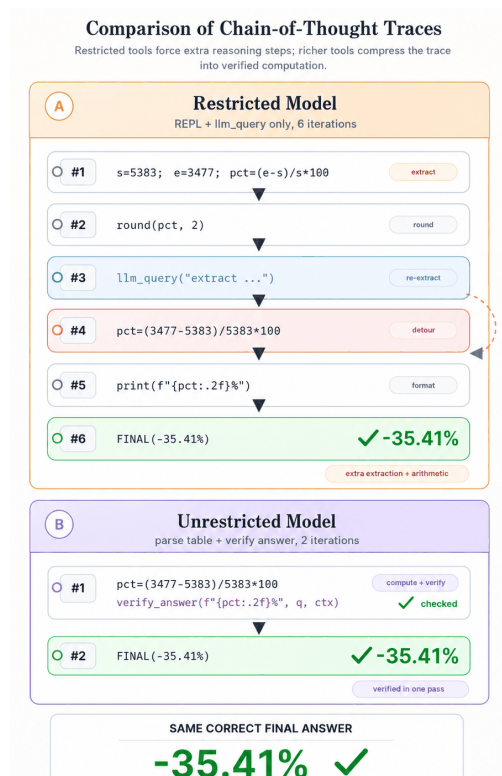


Figure 1: Same GPT-5 root, same DocMath-Eval question, same correct answer, matching the gold exactly.

then, do the tools actually contribute? Tool-augmented LMs are now the standard for financial document QA. Agentic systems (SWE-agent (Yang et al., 2024), OpenHands (Wang et al., 2025)), commercial CLIs (Codex CLI (OpenAI, 2025), Claude Code (Anthropic, 2025), Gemini CLI (Google, 2025)), and interpreter-backed reasoning (Gao et al., 2023; Chen et al., 2023) all report gains from tool use, and finance-specific fine-tuned models narrow the same accuracy gap without tools (Qian et al., 2025; Zhu et al., 2025). But these evaluations bundle the model, the harness, and the tools: none isolates whether the benefit comes from executing tools or from what the model is told about them. Algorithmic Prompt-

ing (Zhou et al., 2022) ran the natural-language control on grade-school arithmetic, and no comparable breakdown exists for long-context financial documents. We hold the model, the harness, and the evaluation fixed and vary only the tool surface, from no added tools to full systems access, plus a description-only control whose tools are described but never executed. The fully equipped configuration beats the restricted baseline by a small margin, yet the description-only control performs within 1 point of it, so the benefit comes from the tool descriptions, not from executing the tools. Tools change how the model computes more than whether it is right, and on the hardest problems the tool advantage vanishes entirely.

Contributions.

1. A controlled ablation that holds the model, harness, and evaluation fixed and varies only the tool surface.
2. A description-only control, where tools are described but never executed.
3. Evidence that tools mainly change how the model works, not how often it is right: fewer steps on long documents, answers ready sooner, and the benefit comes from the tool descriptions alone.
4. A cross-framework comparison with production coding agents, and evidence that tool benefits vanish on an agent-selected hard subset.

2 Related Work

Tool use and strategy knowledge. Toolformer (Schick et al., 2023), ReAct (Yao et al., 2023), and ART (Paranjape et al., 2023) show that language models can learn to call external tools, and that mixing tool actions with reasoning improves multi-step problem solving. PAL (Gao et al., 2023) and Program of Thoughts (Chen et al., 2023) hand arithmetic to an interpreter and report gains over text-only reasoning. Neither line tests whether the same procedures delivered as natural-language descriptions are enough. Algorithmic Prompting (Zhou et al., 2022) runs exactly that control on grade-school arithmetic and finds that natural-language examples match tool-augmented performance, and Sclar et al. (2024) show that LLM outcomes are highly sensitive to the wording of the prompt. Whether this extends to long-context financial documents is untested.

Iterative reasoning and inference-time compute. Structured iterative reasoning improves language model accuracy: chain-of-thought (Wei et al., 2022), self-consistency (Wang et al., 2023), Self-Refine (Madaan et al., 2023), and Reflexion (Shinn et al., 2023) each report gains over single-shot prompting, though Huang et al. (2024) find that models cannot reliably self-correct without external feedback, and Snell et al. (2025) describe the tradeoff between inference-time compute and accuracy.

Coding agents and agentless approaches. Coding agents such as SWE-agent (Yang et al., 2024), AutoCodeRover (Zhang et al., 2024), and OpenHands (Wang et al., 2025) reach strong performance on software engineering through iterative execution and self-checking, yet agentless pipelines match them (Xia et al., 2025), so execution is not always the deciding factor.

Recursive language models. Zhang et al. (2025) introduced recursive language models, where a root LM writes Python in a sandboxed REPL and delegates subtasks to sub-LMs. The framework makes the tool surface explicit and swappable, which is why our ablation builds on it.

Financial QA benchmarks and finance-adapted models. Financial QA is typically evaluated on FinQA (Chen et al., 2021) for table arithmetic, TAT-QA (Zhu et al., 2021) for hybrid tabular-textual reasoning, DocFinQA (Reddy et al., 2024) for long SEC filings, and DocMath-Eval (Zhao et al., 2024) for long-document math. Another line of work targets the same accuracy gap through domain-specific fine-tuning rather than tool use (Qian et al., 2025; Zhu et al., 2025). For evaluation, LLM-as-a-judge protocols (Zheng et al., 2023) are widely used after rule-based normalization.

3 Method

3.1 Task Formulation

We formulate financial QA as a tuple $\langle c, q, y \rangle$, where c is the oracle document, q the question, and y the gold numerical answer. The document c differs by benchmark: a 10-K filing for DocFinQA, a financial-statement table for FinQA, a tabular-textual passage for TAT-QA, and a long financial document for DocMath-Eval. Every system receives the same c , so accuracy differences cannot

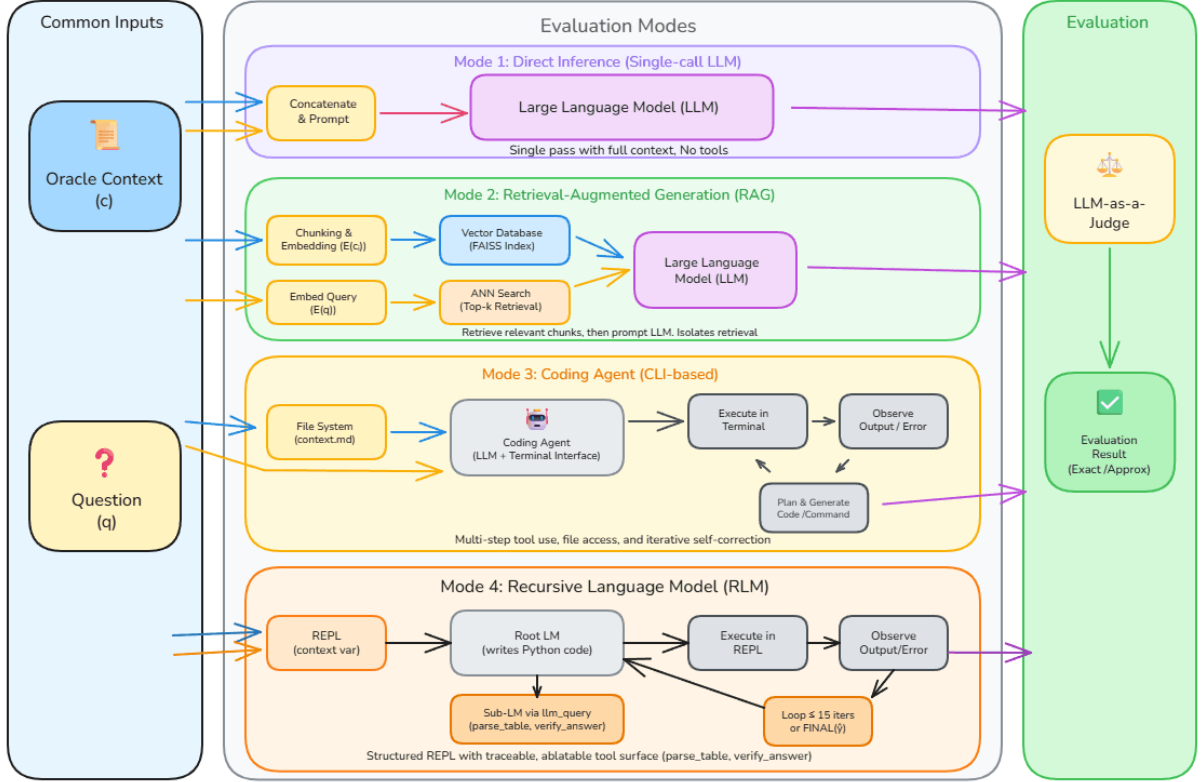


Figure 2: Schematic comparison of the four inference modes under oracle-context conditions.

come from retrieval. q is a natural-language question taken word for word from the source benchmark, and y is a scalar numerical ground truth (occasionally a short span on TAT-QA). A system is a function $f_{\mathcal{T}}$ that maps (c, q) to $\hat{y}$, defined by its tool surface $\mathcal{T}$. Figure 2 illustrates the four evaluation modes.

Tool surface $\mathcal{T}$. The tools the model can call while solving a question, together with what the prompt says about them. $\mathcal{T}$ may contain (a) general-purpose tools (a Python REPL, sub-LM delegation via `llmquery`, `bash`, file I/O), (b) coded reasoning tools (`parsetable`, `verifyanswer`), or (c) no tools ($\mathcal{T} = \emptyset$ for direct inference). We build $\mathcal{T}$ in six ways within the RLM framework, and coding-agent CLIs have their own hidden $\mathcal{T}_{\text{CLI}}$, which we observe through tool traces. We write tool names without underscores for readability; the released code uses `parse_table`, `verify_answer`, and `llm_query`.

3.2 Systems and Inference Modes

We evaluate three types of system, each a different way of reasoning over c .

Baselines: Direct inference and RAG. These are non-agentic controls. In **direct inference**, the full c and q are passed to the LLM in a single call, no tools, no iteration. In **RAG**, c is chunked into 512-token segments, embedded with BAAI/bge-large-en-v1.5¹, indexed with FAISS, and the top-20 chunks by cosine similarity are passed alongside q . These baselines establish what a model can achieve without agentic execution, and whether retrieval helps or hurts when the full context is already available.

Coding agents. Coding agents are LLM-powered systems that solve tasks through an interactive terminal loop: the agent reads the context, writes code or a shell command, executes it, observes the output, and iterates. For financial QA, the oracle context is written to `context.md` in a sandboxed workspace, and the agent is prompted to read this file and return a numerical answer. We evaluate five agents: Codex CLI, Claude Code, Gemini CLI, Vibe CLI, and Trae Agent.

¹<https://huggingface.co/BAAI/bge-large-en-v1.5>

Recursive language models (RLM). In the RLM framework (Zhang et al., 2025), a root LM writes Python code executed in a sandboxed REPL, observing output and revising for up to 15 iterations before emitting $\text{FINAL}(\hat{y})$, with a sub-LM available throughout via `llmquery`. The oracle context is loaded directly into the REPL as the `context` variable. Unlike coding agents, the RLM environment is structured: we control exactly which tools are available, making it a platform for ablation.

3.3 Ablation Design

What drives the gain when tool-augmented models beat a baseline? We break the possible sources down into three components and vary them one at a time. Restricted is itself a capable tool surface (REPL + `llmquery`), so the ablation measures the extra effect of specialized reasoning and system abilities over that baseline, not tools versus no tools. All six configurations share the GPT-5 backbone and the same prompt scaffold, and Table 1 lists the exact capabilities of each.

Skill knowledge (the strategy) is a natural-language description of a reasoning procedure injected into the system prompt, for example “*when the context contains tables with messy formatting, use `llmquery` to ask the sub-LM to extract the table into clean CSV, then parse it with `pandas`*”. The model learns *what* to do but writes the code itself.

Coded tools (the implementation) supply the same two procedures as callable functions, `parsetable` and `verifyanswer`, plus a small notes workspace (`save_note`) for carrying intermediate values across iterations.

Systems access (the environment) adds `run_bash()`, file I/O and `pip_install()`, matching the core abilities of coding agents.

Restricted (REPL + `llmquery` only) is the control. *Skill-Prompted* adds the two strategy descriptions, which isolates the value of knowing the strategy. *Enhanced* supplies the same procedures as code, which separates implementation from strategy. *Unrestricted* adds systems access, which tests whether agent-style execution adds anything once reasoning tools are present.

Description-only keeps *Unrestricted*’s exact prompt, including the names and docstrings of `parsetable`, `verifyanswer`, `run_bash()`, file I/O, and `pip_install()`, but the runtime blocks every added tool call and returns a disabled-tool error, so the model must write any such code it-

self. Against *Unrestricted* it separates description from execution, and against *Restricted* it measures the total effect of the prompt text. *ForceSub* has *Restricted*’s capabilities but must call `llmquery` at least once before answering, which isolates delegation policy from tool surface.

4 Experiments

This section describes the benchmark setup and the evaluation protocol. We run the full RLM ablation on DocMath-Eval, replicate the restricted configuration on FinQA and TAT-QA, and score every output with a fixed two-stage protocol: rule-based normalization, then an LLM judge.

4.1 Benchmarks

Primary ablation benchmark: DocMath-Eval.

All RLM ablations use DocMath-Eval (800 examples), chosen because its 2×2 design crosses question complexity with document length: `SimpShort` (200), `SimpLong` (100), `CompShort` (200), `CompLong` (300), which lets us test whether tool benefits vary with difficulty and context length, and because all answers are purely numerical.

4.2 Evaluation Protocol

Model outputs often embed numerical answers in verbose explanations, so string equality is insufficient. We evaluate with a two-stage protocol: (1) rule-based numeric normalization (sign, unit, scale), applied deterministically when unambiguous, and (2) an LLM judge for remaining cases. The judge $g_\phi(q, y, \hat{y})$ returns one of three labels, $\{\text{EXACT}, \text{APPROXIMATE}, \text{INCORRECT}\}$. It uses Qwen/QwQ-32B², receives $(q, y, \hat{y})$ blind to system identity, and follows the prompt in Figure 8. The exact judge version per run is recorded in the released metadata, alongside the pinned model identifiers (openai/gpt-5 root, openai/gpt-5-mini sub, temperature=0).

- **EXACT:** normalized $\hat{y}$ equals normalized y after unit/sign normalization.
- **APPROXIMATE:** logically equivalent phrasing/sign/units, or rounding.
- **INCORRECT:** none of the above, or the final value cannot be identified.

Accuracy is the percentage labeled EXACT or APPROXIMATE. To check the judge’s reliability, we randomly picked 100 outputs across all systems and splits. A human author labeled each one

²<https://huggingface.co/Qwen/QwQ-32B>

Config	Skill	Coded tools	Systems	Deleg.	Tests
Restricted	–	–	–	Auto	Baseline
Skill-Prompted	NL	–	–	Auto	Strategy knowledge
Enhanced	implicit	parsetable, verifyanswer	–	Auto	Implementation
Unrestricted	implicit	parsetable, verifyanswer	bash, file I/O	Auto	Systems access
Description-only	implicit	parsetable, verifyanswer (disabled)	bash, file I/O (disabled)	Auto	Surface vs. execution
ForceSub	–	–	–	Forced	Delegation autonomy

Table 1: Ablation configurations. *Skill*: strategy text. *Coded tools*: parsetable, verifyanswer. *Systems*: terminal access. *Delegation*: auto/forced. Description-only: tools disabled.

by hand, and the judge agreed with the human on 96 of the 100 (Appendix E). Because even a few judge mistakes are the same size as our measured effects, we also score the four primary configurations (Restricted, Skill-Prompted, Enhanced, Unrestricted) by exact match alone (Table 6). Under exact-match scoring, with no judge at all, Unrestricted still beats Restricted on all four splits. Taken together, Unrestricted wins 42 of the 54 pairs where the two systems disagree, a net gain of +3.8 points ($p < 0.001$), so the headline result does not depend on the judge.

Curated hard subset ($n=200$). To check whether tool benefits persist on the hardest end of the difficulty distribution, we curate a 200-question subset of DocMath-Eval consisting of items that *multiple coding agents in our suite previously answered incorrectly*. We pool the misses of Codex CLI, Claude Code, and Gemini CLI on the four DocMath-Eval splits, keep questions missed by at least two of the three agents, and sample from them in proportion to the original split sizes. Because the RLM shares its model family (GPT-5) with the agents used for selection, this subset is made of questions that GPT-family models tend to miss, so the “36% unsolved across all systems” figure should be read with this in mind.

Tool traces. We set up the RLM runtime to log every tool call (name, arguments, return value, iteration number). The coding agents already write the same information to structured JSON logs. In total we have 2,385 traced calls across both system types on the same 200-question subset.

5 Results and Analysis

5.1 Same-Model Ablation

We vary only the tool surface $\mathcal{T}$ across six GPT-5 configurations on DocMath-Eval ($n=800$, Ta-

ble 2). We test each configuration against Restricted with McNemar’s test (Table 8; bootstrap CIs in Table 10). ForceSub is not a tool-surface configuration: it tests whether always delegating before answering helps or hurts.

Finding 1. Across all splits, the equipped configuration is more accurate than the restricted-tool baseline. Combining the paired per-question outcomes behind Table 9 over all four splits, Unrestricted wins 59 of the 89 pairs on which the two configurations disagree and loses 30, a net $+29/800 = +3.6$ points (exact two-sided binomial, $p=0.0028$). The contrast is stable across three independent runs of each configuration ($+3.6/ +4.2/ +3.1$ points, Table 3). Averaged over splits, accuracy rises in a clean ladder (Table 2, Overall): Restricted 68.6, Enhanced 70.5, Skill-Prompted 71.3, Description-only 71.6, Unrestricted 72.3 (run 1 of each; the three-run Description-only mean is 71.7). Skill-Prompted has no added callable reasoning or systems tools, only strategy text in the prompt, but it performs as well as the tool-equipped configurations: the contrasts against Restricted are +2.7 (Skill-Prompted), +1.9 (Enhanced), +3.0 (Description-only), and +3.6 (Unrestricted) points, and Enhanced is -0.8 points below Skill-Prompted. The two contrasts we test directly both reach significance (Unrestricted $p=0.0028$, Description-only $p=0.048$).

Finding 2. No individual split supports the overall gain on its own. Per split the gains are +3.5, +3.0, +2.0, and +5.0 points on SS/SL/CS/CL, with McNemar p -values of 0.092, 0.648, 0.388, and 0.036. Only CompLong comes close to significance, and it does not survive multiple-comparison correction: across the 16 configuration-vs-baseline comparisons (4 tool configurations $\times$ 4 splits), the corrected thresh-

Configuration	SS <i>n</i> =200	SL <i>n</i> =100	CS <i>n</i> =200	CL <i>n</i> =300	Overall
Restricted	84.5 [79.5, 89.5]	65.0 [56.0, 74.0]	87.5 [82.5, 92.0]	46.7 [41.0, 52.3]	68.6
Skill-Prompted	84.5	67.0	90.5	51.0	71.3
Enhanced	84.5	67.0	89.0	50.0	70.5
Unrestricted	88.0 [83.5, 92.5]	68.0 [59.0, 77.0]	89.5 [85.0, 93.5]	51.7* [46.0, 57.3]	72.3
Description-only [‡]	87.5	67.0	89.0	51.0	71.6
ForceSub [†]	86.5	63.0	88.5	50.7	70.6

Table 2: DocMath-Eval ablation (%). * $p=0.036$ vs. Restricted, uncorrected. † ForceSub: always delegates. ‡ Description-only: execution disabled. Run 1 of three.

Configuration / run	SS	SL	CS	CL	Overall
Restricted, run 1	84.5	65.0	87.5	46.7	68.6
Restricted, run 2	84.0	66.0	86.0	45.7	67.9
Restricted, run 3	85.5	64.5	88.0	47.3	69.2
Restricted, mean±SD	84.7±0.8	65.2±0.8	87.2±1.0	46.6±0.8	68.6±0.6
Unrestricted, run 1	88.0	68.0	89.5	51.7	72.3
Unrestricted, run 2	88.5	66.5	90.0	51.0	72.1
Unrestricted, run 3	87.5	68.5	89.0	52.3	72.3
Unrestricted, mean±SD	88.0±0.5	67.7±1.0	89.5±0.5	51.7±0.7	72.2±0.1
Description-only, run 1	87.5	67.0	89.0	51.0	71.6
Description-only, run 2	87.0	67.5	88.5	51.7	71.7
Description-only, run 3	88.0	66.5	89.5	50.7	71.7
Description-only, mean±SD	87.5±0.5	67.0±0.5	89.0±0.5	51.1±0.5	71.7±0.1

Table 3: Three runs on DocMath-Eval. Contrasts across runs: Unrestricted–Restricted +3.6/ +4.2/ +3.1 (95% CI [+0.6, +6.7]), Description-only–Unrestricted −0.6/ −0.4/ −0.6, mean −0.5 (95% CI [−3.9, +2.8]).

old is $0.05/16=0.0031$, well below 0.036. Most splits are too small to notice an effect this size: with only 13, 19, 12, and 45 disagreeing pairs per split, the smallest difference these tests could catch is 4.5, 11.0, 4.0, and 5.0 points, and SimpLong ($n=100$, needing 11 points) cannot support a split-level claim on its own. We therefore use the test over all splits as primary and report all per-split values, including CompLong’s 0.036, in Table 8. This null should not be read as “the tools are useless”. `parse_table` and `verify_answer` are thin wrappers around `llm_query`, which Restricted already has, so Enhanced versus Skill-Prompted only tests whether packaging the same procedure as code instead of text changes anything. It says nothing about whether the underlying capability itself helps.

Finding 3. The overall gain does not need to execute any tools. Two configurations carry the same strategy content without running anything: Skill-Prompted (strategy text, no added tools) and Description-only (Unrestricted’s tool descriptions, but every added call disabled). Both do as well as the fully equipped configuration. Description-only stays within 1 point of it on every split (87.5, 67.0, 89.0, 51.0), beats Restricted by +3.0 points overall (wins 80 of the 136 questions where the

two disagree, $p=0.048$), and tracks Unrestricted within a fraction of a point across three runs (Table 3). Every number points the same way: executing the tools is not what produces the gain. Among the 61% of Description-only questions where the model never even tried a disabled tool, accuracy is 71.9% (351 of 488), +3.3 points over Restricted’s 68.6%, the cleanest estimate of what the prompt text alone does (these questions may be easier on average). The systems tools that give Unrestricted its name were used only once in 800 questions (Table 11), so Enhanced and Unrestricted behave identically, yet differ by 1.8 points, about half the size of the +3.6-point headline. The systems-tools condition is therefore a control, not a treatment: the oracle document is already loaded as a variable, so `bash`, `file I/O`, and `pip` have nothing to do. The three-run CI for the Unrestricted contrast, [+0.6, +6.7], excludes zero but is too wide to pin the size.

Finding 4. Coded tools do not measurably beat strategy descriptions. Skill-Prompted matches or beats Enhanced on every split ($p \geq 0.250$ on all four McNemar tests), and it also does not separate from Restricted (+0.0, +2.0, +3.0, +4.3 points per split). Given the minimum detectable effects above, this is consistent with both possibilities: the

coded tools do nothing, or they help by less than we can detect. We do not claim they do nothing: the wrappers add convenience rather than new capability, so finding no difference here is partly expected. This says nothing about real tools: a deterministic parser or calculator could behave differently.

Side observations. *ForceSub* performs on par with Restricted on every split: it is +2.0 points better on SS, -2.0 worse on SL, +1.0 better on CS, and +4.0 better on CL, with none of these differences significant. Forcing one delegation neither helps nor hurts. *Conditioning on tool adoption* within Unrestricted, samples using both parsetable and verifyanswer reach 83.6% vs. 57.4% for neither (Table 12), correlational since the model chooses when to use tools. *Delegation* correlates with lower accuracy in both configurations on CompLong, and only on SimpShort does the sign differ (11% vs. 64% of samples delegate on SS, Table 14).

5.2 How Tools Reshape Reasoning Behavior

We log every RLM run with iteration counts and per-iteration code blocks. For the budget- k analysis, we cut each run short at k iterations and score the model’s last submitted value (or mark incorrect if none).

Finding 1. Unrestricted uses 22% fewer root-model iterations on CompLong (3.2 vs. 4.1 iterations, run 1 of three, Table 4), 19% fewer on SimpLong, and 11% fewer on SimpShort, but 20% more on CompShort (2.9 vs. 2.4). The bootstrap 95% confidence intervals for iterations per sample are separate on CompLong (4.1 [3.8, 4.4] vs. 3.2 [3.0, 3.4]), barely touch on SimpLong (3.6 [3.3, 3.9] vs. 2.9 [2.5, 3.3]), and overlap on both short splits, iteration savings are a long-document phenomenon. Wall-clock time improves only on CompLong, where Unrestricted takes 67.4s per sample against 125.6s (-46%), while on SS, SL and CS it is higher for Unrestricted (Table 4), because each iteration triggers more sub-LM calls.

Finding 2. Answers are available earlier when runs are cut short. Runs were produced under a 15-iteration budget and cut short afterward, so this measures which answer each configuration has available at k iterations, not whether the model would stop there on its own. At $k=2$, Unrestricted leads Restricted by +13.7 points on CompLong (23.0% vs. 9.3%), +16.0 on Simp-

System	Metric	SS	SL	CS	CL
RLM Restr.	Iters/sample	2.6	3.6	2.4	4.1
RLM Unr.	Iters/sample	2.3	2.9	2.9	3.2
RLM Restr.	Time/sample (s)	40.7	53.3	30.4	125.6
RLM Unr.	Time/sample (s)	49.7	59.0	60.1	67.4
Claude Code	Time/sample (s)	34.4	33.5	32.1	66.1
Claude Code	Turns/sample	4.1	3.9	3.8	5.5
Codex CLI	Time/sample (s)	27.5	54.6	29.0	-

Table 4: Efficiency on DocMath-Eval. Unrestricted: fewer iterations on long splits (3.2 vs. 4.1 on CompLong, -22%), more on CS (+20%). Wall-clock lower only on CompLong (67.4 vs. 125.6s, -46%).

Short, and +18.0 on SimpLong, but trails by 19.0 on CompShort (37.0% vs. 56.0%). The early-availability advantage is real but split-dependent, and we have no explanation for the CompShort reversal, the largest single behavioral effect in our data.

Finding 3. The model varies its tool mix with document length. parsetable is used on 78% of short-document samples but only 29% of long-document ones, while verifyanswer stays stable (88% vs. 83%). The model may be choosing its tools deliberately, or long contexts may simply not fit in one sub-LM call, and we cannot tell which. Both configurations perform the same two procedures with different tools: Restricted uses regex (32% of CompShort code blocks, down 70% in Unrestricted) plus raw llmquery, while Unrestricted uses pandas with parsetable and verifyanswer.

5.3 Cross-Framework Context and Difficulty Limits

For context, we compare three system families on four benchmarks (Table 5): five CLI agents, two direct baselines (Gemini Pro/Flash), and one RAG baseline. Because backbones differ, these comparisons are reference points, not causal evidence.

Finding 1. Best agent beats best direct baseline, but the same-model comparison does not. Codex CLI exceeds the best direct baseline on six of seven splits (+2.0 to +10.0 points, with TAT-QA the exception at -1.4), but holding the backbone fixed (Gemini Pro Direct vs. Gemini CLI), the direct baseline meets or exceeds the agent on 4 of 7 splits, so much of the gap between the two system types comes from the backbone, while RAG trails the best direct baseline on six of seven splits.

Finding 2. The RLM’s relative advantage is

System	Backbone	Harness	DocFinQA	FinQA	TAT-QA	DocMath-Eval			
						CS	CL	SS	SL
Codex CLI	GPT-5	Agent (CLI)	69.20	85.38 [†]	91.34 [†]	90.50	55.67	89.50	66.00
Claude Code	Claude	Agent (CLI)	63.34 [†]	85.35	95.19	85.50	53.00	86.50	68.00
Gemini CLI	Gemini Pro	Agent (CLI)	49.20	83.09	90.02	83.50	48.33	81.50	64.00
Vibe CLI	Gemini Flash	Agent (CLI)	–	81.17	92.36	84.50	43.33	87.50	60.00
Trae Agent	GPT-5	Agent (CLI)	–	–	–	82.50	53.00	89.00	60.00
RLM Res. [‡]	GPT-5/-mini	RLM	–	78.03	91.10	87.50	46.67	84.50	65.00
RLM Unr.	GPT-5/-mini	RLM	–	–	–	89.50	51.67	88.00	68.00
Gemini Pro	Gemini Pro	Direct	57.38	79.77	92.00	85.50	45.67	84.00	60.00
Gemini Flash	Gemini Flash	Direct	59.33	78.81	92.78	81.50	42.67	87.00	64.00
Gemini Flash	Gemini Flash	RAG	41.32	77.86	92.42	82.00	36.33	85.50	64.00

Table 5: Accuracy (%). Different models per row, no best value. [‡] RLM Restricted on FinQA ($n=1147$), TAT-QA ($n=1663$). [†] Codex FinQA/TAT-QA, Claude DocFinQA: 100-sample runs.

largest on the benchmark it was developed on. RLM Unrestricted is competitive with the best agents on 3/4 DocMath-Eval splits. On the other test sets Restricted sits below the best CLI agent (Table 5), and the Restricted-vs-Unrestricted contrast on a 200-sample FinQA subset is only +1.0 points (81.5 vs. 82.5).

Finding 3. On an agent-selected stress-test subset, tool advantages disappear. The within-RLM tool-conditioned gap vanishes on the curated 200-question subset (42.7% with both tools vs. 44.0% with neither), and 36% of these questions are unsolved across *all* systems. Because the subset is selected on the very outcome we measure, it is a stress test by construction rather than an independent difficulty tier, and it is not evidence of a general comprehension ceiling. The two system types reach comparable accuracy on easier splits through entirely different tool surfaces (Table 13).

5.4 Discussion

Tools change how the model computes, not whether it gets the answer right. Extraction and verification happen in every configuration; tools decide only which of them execute cleanly and how quickly the model reaches its answer. The effects are split-dependent, and the absent Skill-Prompted/Enhanced difference extends the NL-versus-code null of Zhou et al. (2022) to long-context financial reasoning. The description-only control confirms that the prompt text matters: with execution disabled, it tracks the fully equipped configuration within a point on every split across three runs. The gain comes from what the model reads, not what it runs. Our scope is thin wrappers over capabilities the baseline already has, under oracle context; we cannot speak to deterministic

parsers, retrieval, or tools that supply new information.

Document comprehension, not tool availability, limits performance on the hardest questions. The CompLong error taxonomy (Table 15) locates the failure (single-annotator, so treat proportions as indicative): comprehension and extraction errors dominate, and *verifyanswer* fires on most of them without catching them, because the sub-LM inherits the root’s misreading. Consistent with this, the tool advantage vanishes on the hard subset, where no tool in our surface addresses comprehension. Sign and scale mistakes are the one easy fix: a deterministic post-processor could catch them, though our wrappers do not.

6 Conclusion

We isolate what tools contribute to financial document reasoning through a same-model ablation and a cross-framework comparison. The fully equipped configuration beats the restricted-tool baseline overall, though no individual split survives multiple-comparison correction. The gain is not due to the capability that names it: the systems tools go essentially unused, and a description-only control performs almost identically across three runs. Tools act mainly on *how* the model computes, and those effects reverse on CompShort. On a hard subset the tool advantage disappears, many questions go unsolved by every system, and comprehension and extraction errors dominate: comprehension, not tool availability, is the limiting factor. These results are scoped to thin wrappers under oracle context.

Limitations

Model and benchmark scope. All ablation configurations use GPT-5 as root and GPT-5-mini as sub-LM, and we do not have evidence that the patterns generalize to other model families: the five coding agents span different backbones (GPT-5, Claude, Gemini), but they are not part of the controlled ablation. The six-way ablation runs on DocMath-Eval. Restricted is also reported on FinQA and TAT-QA (end-to-end accuracy only), and the Restricted/Unrestricted contrast is replicated only on a 200-sample FinQA subset.

Statistics and variance. Our paired tests treat individual questions as the unit, and DocMath-Eval’s metadata does not expose a document-to-question mapping, so questions from the same filing may be correlated and the intervals slightly optimistic. The conclusions would only change if the true effect were several times smaller than measured. Restricted, Unrestricted, and Description-only were run three times each (Table 3), and per-cell swings are a few points, so single-run differences of that size cannot be attributed to configuration. The other configurations are single runs, and temperature=0 does not make the API models deterministic. Token- and dollar-level usage was not logged, so per-sample cost cannot be reconstructed: iteration counts are not a cost measure, and wall-clock is lower only on CompLong (Table 4). The cost-benefit question is left open.

Scope of the claims. Our reasoning tools are two thin wrappers around llmquery, and richer tool families may behave differently. The null is about this class of tool, not about adding tools in general. The Description-only result separates prompt surface from execution, not delegation from tools, and its interval admits prompt-text effects in either direction. All systems receive the ground-truth document, which isolates reasoning from retrieval but understates deployment difficulty: retrieval, search, chunking and file-system abilities have nothing to do by design, which is the most likely reason the systems tools go unused, and we do not claim these results transfer to a non-oracle condition.

References

Anthropic. 2025. [Claude code overview](#). Accessed: 2026-07-26.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. [Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks](#). *Transactions on Machine Learning Research*.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. 2021. [FinQA: A dataset of numerical reasoning over financial data](#). In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3697–3711, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. [PAL: Program-aided language models](#). In *Fortieth International Conference on Machine Learning*, pages 10764–10799.

Google. 2025. [Gemini CLI | gemini code assist](#). Accessed: 2026-07-26.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Yu, Xinying Song, and Denny Zhou. 2024. [Large language models cannot self-correct reasoning yet](#). In *The Twelfth International Conference on Learning Representations*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. [Self-refine: Iterative refinement with self-feedback](#). In *Thirty-seventh Conference on Neural Information Processing Systems*.

OpenAI. 2025. [Codex CLI](#). Accessed: 2026-07-26.

Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. 2023. [ART: Automatic multi-step reasoning and tool-use for large language models](#). *Computing Research Repository*, arXiv:2303.09014.

Lingfei Qian, Weipeng Zhou, Yan Wang, Xueqing Peng, Han Yi, Yilun Zhao, Jimin Huang, Qianqian Xie, and Jian-yun Nie. 2025. [Fino1: On the transferability of reasoning-enhanced LLMs and reinforcement learning to finance](#). *Computing Research Repository*, arXiv:2502.08127.

Varshini Reddy, Rik Koncel-Kedziorski, Viet Dac Lai, Michael Krumdick, Charles Lovering, and Chris Tanner. 2024. [DocFinQA: A long-context financial reasoning dataset](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 445–458, Bangkok, Thailand. Association for Computational Linguistics.

- Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. [Toolformer: Language models can teach themselves to use tools](#). In *Thirty-seventh Conference on Neural Information Processing Systems*.
- Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. 2024. [Quantifying language models’ sensitivity to spurious features in prompt design or: How I learned to start worrying about prompt formatting](#). In *The Twelfth International Conference on Learning Representations*.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. [Reflection: Language agents with verbal reinforcement learning](#). In *Thirty-seventh Conference on Neural Information Processing Systems*.
- Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. [Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning](#). In *The Thirteenth International Conference on Learning Representations*.
- Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Daniel Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, and 5 others. 2025. [OpenHands: An open platform for AI software developers as generalist agents](#). In *The Thirteenth International Conference on Learning Representations*.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. [Self-consistency improves chain of thought reasoning in language models](#). In *The Eleventh International Conference on Learning Representations*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. 2022. [Chain-of-thought prompting elicits reasoning in large language models](#). In *Thirty-sixth Conference on Neural Information Processing Systems*.
- Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. 2025. [Demystifying LLM-based software engineering agents](#). *Proceedings of the ACM on Software Engineering*, 2(FSE):801–824.
- John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. [SWE-agent: Agent-computer interfaces enable automated software engineering](#). In *Thirty-eighth Conference on Neural Information Processing Systems*.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. [ReAct: Synergizing reasoning and acting in language models](#). In *The Eleventh International Conference on Learning Representations*.
- Alex L. Zhang, Tim Kraska, and Omar Khattab. 2025. [Recursive language models](#). *Computing Research Repository*, arXiv:2512.24601.
- Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. 2024. [AutoCodeRover: Autonomous program improvement](#). In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 1592–1604, New York, NY, USA. Association for Computing Machinery.
- Yilun Zhao, Yitao Long, Hongjun Liu, Ryo Kamoi, Linyong Nan, Lyuhao Chen, Yixin Liu, Xiangru Tang, Rui Zhang, and Arman Cohan. 2024. [DocMath-Eval: Evaluating math reasoning capabilities of LLMs in understanding long and specialized documents](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 16103–16120, Bangkok, Thailand. Association for Computational Linguistics.
- Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. [Judging LLM-as-a-judge with MT-Bench and chatbot arena](#). In *Thirty-seventh Conference on Neural Information Processing Systems (Datasets and Benchmarks)*.
- Hattie Zhou, Azade Nova, Hugo Larochelle, Aaron Courville, Behnam Neyshabur, and Hanie Sedghi. 2022. [Teaching algorithmic reasoning via in-context learning](#). *Computing Research Repository*, arXiv:2211.09066.
- Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. [TAT-QA: A question answering benchmark on a hybrid of tabular and textual content in finance](#). In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3277–3287, Online. Association for Computational Linguistics.
- Jie Zhu, Qian Chen, Huaixia Dou, Junhui Li, Lifan Guo, Feng Chen, and Chi Zhang. 2025. [DianJin-R1: Evaluating and enhancing financial reasoning in large language models](#). *Computing Research Repository*, arXiv:2504.15716.

Appendix A. Experimental Configuration

Table 5 in the main text reports accuracy on the four benchmarks under oracle-context evaluation. This appendix records the exact configuration behind each of its rows.

Table 7 reports the exact configuration of each system in Table 5. All systems received the same oracle context as input and were prompted to return a numerical answer.

The prompts, including the judge prompt, are reproduced in Appendix F. The tool wrappers, inference configurations, predictions, judge labels, and traces are available at <https://github.com/runananya/finbench-clean>, though reproduction requires access to the corresponding GPT-5/GPT-5-mini API models.

Appendix B. Full Statistical Results

Three-run reruns of Restricted, Unrestricted, and Description-only (Table 3), and the paired contrast CIs cited in Section 5.1 are computed from these runs.

This appendix collects every significance test, confidence interval and paired count referred to in Section 5.

B.1 Pairwise Significance Tests

Table 8 reports McNemar’s test for every pairwise system comparison discussed in Section 5, together with Cohen’s h as an effect-size measure. Comparisons are run per split on paired per-question correctness labels.

B.2 Paired Per-Sample Outcomes

Table 9 reports the per-question wins and losses behind the Unrestricted vs. Restricted contrast on each split, with the overall counts in the final column.

B.3 Bootstrap Confidence Intervals

Table 10 reports 95% bootstrap confidence intervals (10,000 resamples) for each configuration on each split, complementing the paired contrast CIs in Table 3.

Appendix C. Tool Adoption and Delegation

Table 11 records which tools each configuration actually used, as opposed to which it was given; Table 12 conditions accuracy on tool use; Table 13 compares tool use between the RLM and the coding agents on the hard subset.

C.1 Sub-LM Delegation

Table 14 breaks down sub-LM delegation rates against accuracy, separating the Restricted and

Unrestricted configurations on the shortest and longest DocMath-Eval splits.

Appendix D. Efficiency Comparison

Table 4 compares iterations and wall-clock time per sample for the RLM configurations and two coding agents; it is discussed in Section 5.2.

Appendix E. Error Analysis and Judge Validation

Repeated verification note. The number of verifyanswer calls exhibits a non-monotonic pattern: one call: 78.4% accuracy ($n=547$), two calls: 58.4% ($n=113$), three or more: 53.8% ($n=26$), and zero calls: 60.5% ($n=114$). This is descriptive (samples needing more verification may be inherently harder) but consistent with a second-guessing dynamic. The pattern does *not* appear on FinQA (stable through two calls), suggesting it is specific to long-context problems where the sub-LM is more likely to err.

We present one representative failure from each major error category (Table 15), drawn from RLM Unrestricted on CompLong.

Comprehension error. Gold: 10.9. Predicted: 195.345. The question asked for a specific financial metric worth \$10.9M. The model instead interpreted it as requesting the sum of six unrelated cost items scattered across a 105,000-character filing (acquisition costs, CEO transition charges, bad debt reserves, etc.). It extracted each figure using regex, summed them correctly as 195.345, and verified with the sub-LM, which also confirmed 195.345. The arithmetic was flawless, but the comprehension was wrong. This exemplifies how verification tools cannot catch errors rooted in misunderstanding the question.

Extraction error (scale mismatch). Gold: −5,714.29. Predicted: −5,714,285.71. The model computed the numerically correct result but reported it in raw dollars rather than the “in thousands” unit convention used throughout the filing’s tables. The prediction is exactly $1,000\times$ the gold answer. Despite 12 sub-LM calls and 3 verification rounds, neither model caught the unit mismatch. SEC filings express values “in thousands” or “in millions” in table headers, and the model sometimes loses track of the implicit scale factor during multi-step calculations.

Verification failure. Gold: 3,000,000. Predicted: 3,450,000. The \$450,000 discrepancy suggests the model included an additional cost component that should not have been part of the calculation. The sub-LM verified the wrong answer, converging on the same misinterpretation of which line items to include. When both models share the same blind spots, self-checking provides no corrective signal: the verifier merely confirms a confidently wrong answer. This failure mode accounts for 13.1% of all errors and represents a fundamental limitation of same-family model verification.

Timeout / incomplete. Gold: 220.81. Predicted: “Insufficient data to compute ROI%...” The model concluded the filing lacked sufficient information and returned a textual explanation instead of a number. The tool trace shows it searched for “net income attributable to acquired properties” and, finding no exact match for that phrase, gave up, despite the data being present under different terminology. This conservative behavior guarantees failure in benchmark evaluation but occurs in only 3.4% of errors.

Human audit of the judge. To check that judge labels track human readings, one author manually audited 100 (prediction, judge label) pairs sampled uniformly at random across systems and splits. The judge’s label matched the author’s reading on 96/100 cases, and the four disagreements were borderline approximate-vs.-incorrect cases, such as percentage values reported in basis points. This is a small single-annotator spot check, not a formal inter-annotator agreement study, so we treat the 96% figure as a rough check rather than a measured agreement rate (see Limitations). For the comparisons that drive the paper’s main claims (Unrestricted vs. Restricted on Complex-Long, Skill-Prompted vs. Enhanced on all splits), the qualitative pattern is unchanged under exact-match scoring alone (Table 6), giving a second check that the judge is not load-bearing, though judge/EM disagreement concentrates on verbose outputs (18.5% in the longest-prediction quartile vs. 2.8–6.0% elsewhere), so we report both scorings rather than treating judge noise as mean-zero.

Appendix F. Prompt Templates and Tool Wrappers

All configurations share the same base system prompt. Each variation adds one prompt block or

Configuration	SS	SL	CS	CL
Restricted	77.5	58.0	85.5	40.3
Skill-Prompted	79.0	64.0	89.0	46.0
Enhanced	80.5	61.0	86.5	43.7
Unrestricted	81.0	63.0	86.0	46.0

Table 6: Exact-match-only accuracy (%): predictions and gold parsed to floats, correct within 1% tolerance. The Unrestricted–Restricted direction matches the judged numbers, though the ordering differs.

changes what the runtime allows; the figures below show the exact text from the released configuration files.

Base prompt (Restricted). The REPL environment is introduced as shown in Figure 3:

Skill-Prompted: strategy text. The prompt additionally contains a recommended-strategies block (Figure 4), with code examples omitted here:

Enhanced: coded tools. The prompt additionally documents the two callable functions (Figure 5), thin wrappers around `llm_query`:

Unrestricted: systems access. The prompt additionally documents terminal-style tools and widens the allowed imports (Figure 6):

Description-only. The prompt is identical to Unrestricted’s, but the runtime blocks every added tool call (`parse_table`, `verify_answer`, and the systems tools) and raises a disabled-tool error, so the model must write any such code itself. The REPL and `llm_query` remain available.

ForceSub. Restricted’s prompt plus a reminder injected mid-loop whenever the model has not called `llm_query` after the first iteration (Figure 7):

Both wrappers are thin layers around `llm_query`. The coded tools add implementation convenience but no capability beyond what Skill-Prompted can achieve through raw `llmquery` calls.

LLM judge. Scoring uses the prompt in Figure 8, identical for every system. The system block fixes the normalization and labeling rules; the user block supplies only the question, the gold answer, and the candidate answer, so the judge never sees which configuration produced a prediction. The judge is decoded at `temperature=0` and its reply is parsed as JSON.

Base prompt (Restricted)

1. A context variable that contains extremely important information about your query.
 2. A `llm_query` function that allows you to query an LLM (that can handle around 500K chars) inside your REPL environment.
 3. A `llm_query_batched` function that allows you to query an LLM on multiple items in parallel.
 4. The ability to use `print()` statements to view the output of your REPL code.
- IMPORTANT: When you are done with the iterative process, you **MUST** provide a final answer inside a `FINAL` function when you have completed your task, **NOT** in plain text.
- Allowed imports: `math`, `re`, `json`, `collections`, `itertools`, `statistics`, `decimal`, `fractions`, `numpy`, `pandas`.
- You cannot use: `os`, `sys`, `subprocess`, `open`, `eval`, `exec`.

Figure 3: The base system prompt used by every configuration.

Skill-Prompted: added block

RECOMMENDED STRATEGIES for financial document reasoning:

Strategy 1: Structured table extraction. When the context contains tables with messy formatting, use `llm_query()` to ask the sub-LM to extract the table into clean CSV format, then parse it with `pandas`. This is much more reliable than writing regex yourself.

Strategy 2: Answer self-verification. Before submitting your final answer, use `llm_query()` to ask the sub-LM to independently answer the same question from the raw evidence. If the answers disagree, re-examine your work.

These strategies are optional: use them when they would help, skip them when the problem is straightforward.

Figure 4: The recommended-strategies block added by Skill-Prompted.

Enhanced: added block

5. A `parse_table(text, description)` function that extracts a table from messy text into a clean `pandas DataFrame`. Use this instead of writing regex to parse tables. It handles inconsistent formatting, merged cells, and financial notation.

6. A `verify_answer(answer, question, evidence)` function that asks the sub-LM to independently compute the answer and checks if it matches yours. Returns a dict with `verified (bool)`, `sublm_answer`, and `explanation`. Use this before calling `FINAL()` to catch errors.

7. Notes workspace: `save_note(key, value)`, `load_note(key)`, and `list_notes()` save intermediate findings across iterations so you don't lose track of extracted values.

Figure 5: The coded-tool documentation added by Enhanced.

Unrestricted: added block

8. `write_file(filename, content)`: Write content to a file in the workspace.
9. `read_file(filename)`: Read a file from the workspace.
10. `list_files(path=".")`: List files in the workspace directory.
11. `run_bash(command, timeout=60)`: Execute a shell command and return output.
12. `pip_install(package)`: Install a Python package.
13. `WORKSPACE`: String path to your workspace directory.
Allowed imports: `math`, `re`, `json`, `collections`, `itertools`, `statistics`, `decimal`, `fractions`, `numpy`, `pandas`, `os`, `sys`, `subprocess`, `shutil`, `pathlib`, `glob`, `tempfile`, `csv`, `io`, `datetime`, `time`. You can also use `open()` for file I/O.

Figure 6: The terminal-style tools and widened imports added by Unrestricted.

ForceSub: injected reminder

IMPORTANT REMINDER: You MUST use `llm_query()` at least once before calling `FINAL()`. The sub-LM is powerful and can help you: 1. Verify your extracted values by asking it to re-read the relevant section 2. Double-check your calculations by asking it to compute independently 3. Parse complex tables by having it extract specific cells
Do NOT skip this step. Call `llm_query()` now, then call `FINAL()` with your verified answer.

Figure 7: The mid-loop reminder injected by ForceSub.

System	Backbone	Iter./turn cap	Wall budget	Tool surface
Codex CLI	GPT-5	CLI default	1h/sample	bash, file I/O (~a full-auto)
Claude Code	Claude	CLI default	1h/sample	bash, file I/O
Gemini CLI	Gemini Pro	CLI default	1h/sample	bash, file I/O
Trae Agent	GPT-5	CLI default	1h/sample	bash, file I/O
Vibe CLI	Gemini Flash	CLI default	1h/sample	bash, file I/O
Gemini Pro Direct	Gemini Pro	1 call	–	none
Gemini Flash Direct	Gemini Flash	1 call	–	none
Gemini Flash RAG	Gemini Flash	1 call	–	top-20 chunks of 512 tokens, BAAI/bge-large-en-v1.5 + FAISS
RLM Restricted	GPT-5 / GPT-5-mini	15 iters	–	REPL + llmquery only
RLM Skill-Prompted	GPT-5 / GPT-5-mini	15 iters	–	REPL + llmquery + NL skill descriptions in system prompt
RLM Enhanced	GPT-5 / GPT-5-mini	15 iters	–	+ parsetable, verifyanswer (llmquery wrappers)
RLM Unrestricted	GPT-5 / GPT-5-mini	15 iters	–	+ run_bash, file I/O, pip_install
RLM Description-only	GPT-5 / GPT-5-mini	15 iters	–	Unrestricted prompt, added tool calls disabled (REPL + llmquery remain)
RLM ForceSub	GPT-5 / GPT-5-mini	15 iters	–	REPL + llmquery, llmquery required before FINAL

Table 7: Per-system configurations (cross-framework comparison, Table 5, RLM ablation, Table 2). RLM: temperature 0. Coding agents: CLI defaults, 1h/sample.

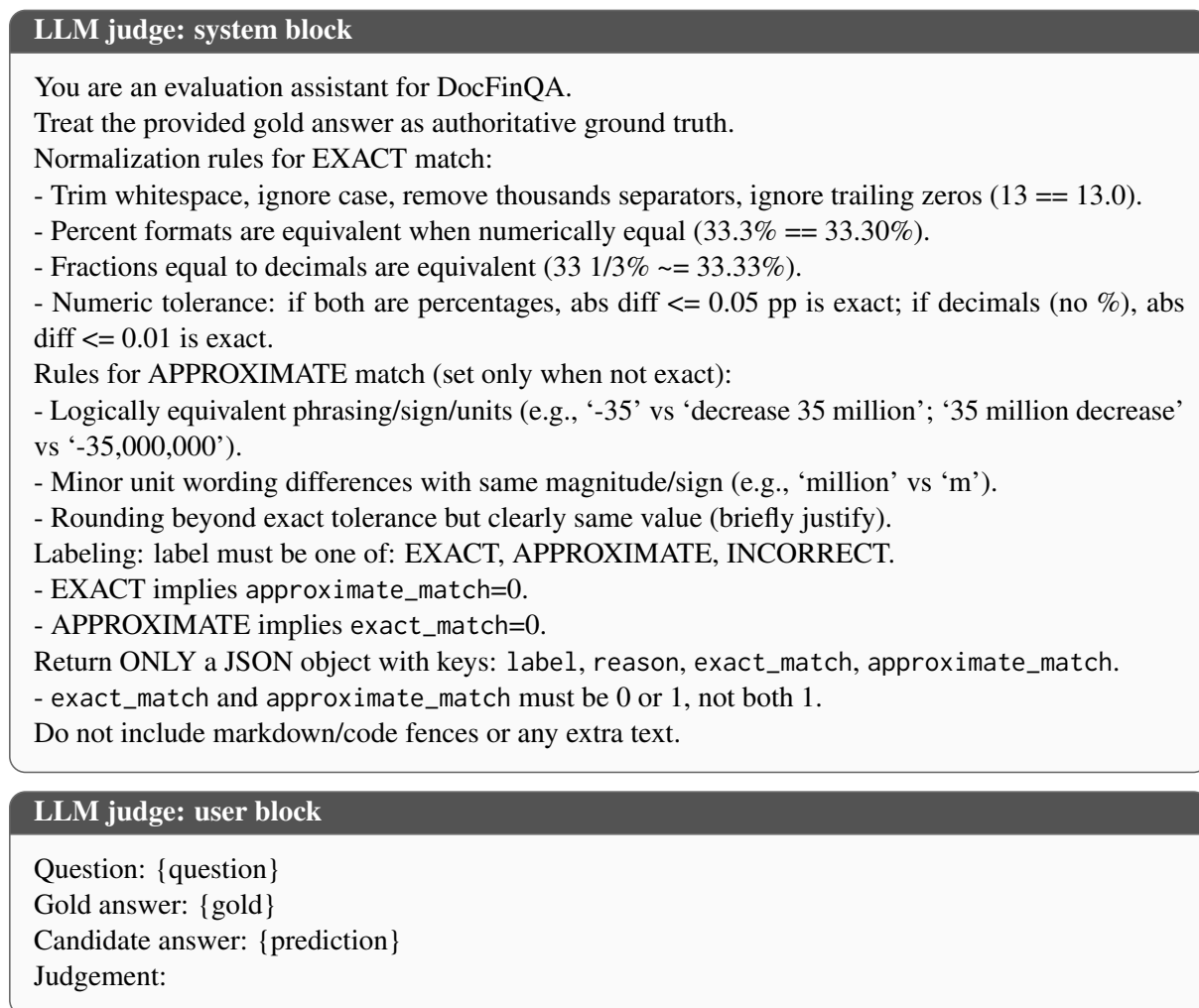


Figure 8: The LLM-as-a-judge prompt. The system block is constant across all runs; the user block is filled per sample and carries no system identity.

System A	System B	Split	<i>p</i> -value	Diff (points)	<i>h</i>
RLM Unr.	RLM Restr.	SS	0.092	+3.5	0.102
RLM Unr.	RLM Restr.	SL	0.648	+3.0	0.064
RLM Unr.	RLM Restr.	CS	0.388	+2.0	0.063
RLM Unr.	RLM Restr.	CL	0.036	+5.0	0.100
RLM Unr.	Codex Agent	SS	0.508	−1.5	0.047
RLM Unr.	Codex Agent	SL	0.791	+2.0	0.043
RLM Unr.	Codex Agent	CS	0.754	−1.0	0.033
RLM Unr.	Claude Code	SS	0.581	+1.5	0.045
RLM Unr.	Claude Code	SL	1.000	0.0	0.000
RLM Unr.	Claude Code	CS	0.057	+4.0	0.121
RLM Unr.	Claude Code	CL	0.677	−1.3	0.027
RLM Restr.	Codex Agent	SS	0.021	−5.0	0.149
RLM Restr.	Codex Agent	CL	<0.001	−9.0	0.180
Codex Agent	Claude Code	CS	0.013	+5.0	0.155
Skill-Pr.	Enhanced	SS	1.000	0.0	0.000
Skill-Pr.	Enhanced	SL	1.000	0.0	0.000
Skill-Pr.	Enhanced	CS	0.250	+1.5	0.046
Skill-Pr.	Enhanced	CL	0.720	+1.0	0.020
Desc.-only	Restr.	SS	0.286	+3.0	0.087
Desc.-only	Restr.	SL	0.845	+2.0	0.042
Desc.-only	Restr.	CS	0.453	+1.5	0.047
Desc.-only	Restr.	CL	0.182	+4.3	0.086
Desc.-only	Unr.	SS	1.000	−0.5	0.015
Desc.-only	Unr.	SL	1.000	−1.0	0.021
Desc.-only	Unr.	CS	1.000	−0.5	0.016
Desc.-only	Unr.	CL	0.901	−0.7	0.014

Table 8: McNemar’s test. Family: 16 tool-config vs. baseline comparisons (Section 5.1). Combined Unrestricted vs. Restricted is also significant under CMH ($\chi^2=9.45$, $p=0.0021$).

Split	+Gain	−Loss	Net	Ratio
SS	10	3	+7	3.3:1
SL	11	8	+3	1.4:1
CS	8	4	+4	2.0:1
CL	30	15	+15	2.0:1

Table 9: Per-sample gains and losses from Unrestricted over Restricted.

System	Split	Acc. (%)	95% CI
RLM Unr.	SS	88.0	[83.5, 92.5]
RLM Unr.	SL	68.0	[59.0, 77.0]
RLM Unr.	CS	89.5	[85.0, 93.5]
RLM Unr.	CL	51.7	[46.0, 57.3]
RLM Restr.	SS	84.5	[79.5, 89.5]
RLM Restr.	SL	65.0	[56.0, 74.0]
RLM Restr.	CS	87.5	[82.5, 92.0]
RLM Restr.	CL	46.7	[41.0, 52.3]
RLM Skill-Pr.	SS	84.5	[79.0, 89.5]
RLM Skill-Pr.	SL	67.0	[58.0, 76.0]
RLM Skill-Pr.	CS	90.5	[86.0, 94.5]
RLM Skill-Pr.	CL	51.0	[45.3, 56.7]
RLM Enh.	SS	84.5	[79.5, 89.5]
RLM Enh.	SL	67.0	[58.0, 76.0]
RLM Enh.	CS	89.0	[84.5, 93.0]
RLM Enh.	CL	50.0	[44.3, 55.7]
RLM ForceSub	SS	86.5	[81.5, 91.0]
RLM ForceSub	SL	63.0	[53.0, 72.0]
RLM ForceSub	CS	88.5	[84.0, 93.0]
RLM ForceSub	CL	50.7	[45.0, 56.3]
RLM Desc.-only	SS	87.5	[83.0, 92.0]
RLM Desc.-only	SL	67.0	[58.0, 76.0]
RLM Desc.-only	CS	89.0	[84.5, 93.5]
RLM Desc.-only	CL	51.0	[45.3, 56.7]
Codex Agent	SS	89.5	[85.0, 93.5]
Codex Agent	CL	55.7	[50.0, 61.3]
Claude Code	SS	86.5	[81.5, 91.0]
Claude Code	CL	53.0	[47.0, 58.7]

Table 10: Bootstrap 95% confidence intervals (10,000 resamples). Paired contrast CIs are in Table 3.

Tool	Calls	Samples	Acc.
<i>Reasoning tools</i>			
verifyanswer	859	686 (85.8%)	74.2%
parsetable	688	430 (53.8%)	81.6%
save_note	3	2 (0.3%)	100%
<i>Systems tools (available but largely unused)</i>			
write_file	1	1 (0.1%)	0%
run_bash	0	0	–
read_file	0	0	–
pip_install	0	0	–
Any reasoning tool	–	732 (91.5%)	–

Table 11: RLM tool adoption across 800 DocMath-Eval samples.

Tool Combination	Accuracy	<i>n</i>
Both parsetable + verifyanswer	83.6%	384
verifyanswer only	62.3%	302
parsetable only	65.2%	46
Neither tool	57.4%	68

Table 12: Tool-adoption conditioned accuracy (RLM Unrestricted, 800 samples, tool use = trace records ≥ 1 call). Correlational, not causal: adoption is confounded with question difficulty.

Tool Category	Coding Agents	RLM Unr.
Execution (bash, mostly Python) [†]	1,113 (55.8%)	1 (0.3%)
Reasoning (parsetable, verifyanswer)	0 (0%) [‡]	390 (99.7%)
Task management	846 (42.4%)	0
File editing	35 (1.8%)	0
Total	1,994	391

Table 13: Tool calls on the same 200 hard questions. [†] 69.9% of bash calls run Python. [‡] Agents had no parsetable/verifyanswer, the RLM used them once (near-zero).

Config	Split	% SubLM	Acc w/	Acc w/o
Restricted	SS	11%	81.8	84.8
Restricted	CL	31%	45.7	47.1
Unrestricted	SS	64%	90.6	83.3
Unrestricted	CL	69%	48.6	58.7

Table 14: Sub-LM usage patterns and accuracy (%). In Restricted, sub-LM usage correlates with *lower* accuracy, in Unrestricted with *higher* accuracy on short documents.

Error Category	Count	%
Comprehension error	48	33.1
Extraction error	38	26.2
Arithmetic error	27	18.6
Verification failure	19	13.1
Table parsing error	8	5.5
Timeout / incomplete	5	3.4
Total	145	99.9

Table 15: Error taxonomy for RLM Unrestricted on CompLong ($n=145$); percentages do not sum to exactly 100 because of rounding. verifyanswer fired in 80.0% (116/145) without catching the error. Sign+scale errors (17.2%) are deterministically recoverable.