SPREADSHEET-RL: ADVANCING LARGE LANGUAGE MODEL AGENTS ON REALISTIC SPREADSHEET TASKS VIA REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Spreadsheet systems such as Microsoft Excel and Google Sheets are central to modern data workflows, and AI agents that operate them are a promising research direction. Existing spreadsheet agents mostly prompt general-purpose LLMs, which works for simple operations but struggles with the multi-step workflows that dominate real-world use. We introduce **Spreadsheet-RL**, a reinforcement learning (RL) framework for training specialized spreadsheet agents in a realistic Microsoft Excel environment. It comprises an automated data agent that collects paired initial–final spreadsheets from online forums; **Spreadsheet Gym**, a multi-turn Excel environment with a spreadsheet-native tool harness and an asynchronous Excel-based verifier; and **Domain-Spreadsheet**, a new benchmark of 1,660 professional tasks covering finance, supply chain, human resources, sales, and real estate. On SpreadsheetBench, Spreadsheet-RL raises Qwen3-4B-Thinking-2507 from **12.0%** to **23.4%** Pass@1, with gains from both the spreadsheet-native harness and RL post-training; the gains hold at a larger model size on Qwen3-8B (16.8% to 22.3%) and generalize to Domain-Spreadsheet (**8.4%** to **17.2%**). A human audit of 100 training and 100 Domain-Spreadsheet tasks finds majority-valid oracles for 84% and 98% of tasks, and trajectory analysis shows that RL mainly improves interaction reliability, cutting tool-execution errors from 48.9% to 7.4% of calls. These results highlight **Spreadsheet-RL**'s strong potential for generalization and real-world adoption in spreadsheet automation, and broadly, its promise for advancing LLM-based interactions with data interfaces in everyday work.

1 INTRODUCTION

Spreadsheet systems, such as Microsoft Excel, Google Sheets, WPS Sheets, and LibreOffice, are widely adopted in data-centric workflows (De Leon & Parameswaran, 2022; Rahman et al., 2020). They support tasks from *personal* activities such as travel planning and household budgeting, to *professional* duties such as financial modeling and data presentation (Rahman et al., 2020; Mack et al., 2018; Kandel et al., 2012). As AI agents grow in both popularity and capability for automating tasks traditionally performed by humans, such as computer use and slide-deck generation (Wang et al., 2025b; Koh et al., 2024; Ge et al., 2025; Xie et al., 2024), the development of an AI agent for spreadsheets (a *spreadsheet agent*) to automate human-operated, spreadsheet-centered workflows has the potential to reshape how data-centric work is done at scale.

Recent spreadsheet agents such as SheetCopilot (Li et al., 2023), SheetAgent (Chen et al., 2025), and ChatGPT Agent (OpenAI, 2025b) rely on prompting proprietary Large Language Models (LLMs) such as GPT-4o (Hurst et al., 2024), whose general capabilities suffice for simple spreadsheet operations issued in natural language (NL). Their progress therefore comes from advances in general LLMs and prompting rather than from improving how agents use spreadsheets, and they struggle on the complex, multi-step workflows that dominate real-world use: ChatGPT Agent and Copilot (both with Excel access) reach 45.5% and 20.0% on SpreadsheetBench (Ma et al., 2024). Frontier industry labs have begun to build specialized spreadsheet agents, but with undisclosed methods, internal benchmarks, and closed training pipelines (OpenAI, 2025a; Microsoft, 2025; Google, 2025).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

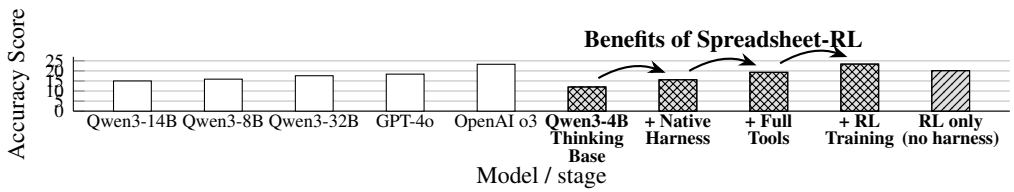


Figure 1: Pass@1 on SpreadsheetBench. Crosshatched bars: Qwen3-4B-Thinking-2507 from the raw model through the spreadsheet-native harness, comprehensive tools, and Spreadsheet-RL post-training; the rightmost bar is RL without the harness or structured tools. Open bars: baselines for scale, with closed-source numbers as reported by their sources in different environments.

A promising route to a powerful, specialized, open-source spreadsheet agent is reinforcement learning (RL) fine-tuning: following DeepSeek-R1 (Guo et al., 2025), on-policy RL with rule-based, verifiable outcome rewards has improved mathematical (Shao et al., 2024; Wang et al., 2025a) and visual reasoning (Wu et al., 2026) and enabled scalable post-training for agentic domains such as software engineering (Wei et al., 2025a;b), web interaction (Bai et al., 2026), data (Song et al., 2026; Nie et al., 2026), and computer use (Lai et al., 2025; Xie et al., 2024; Sun et al., 2025). Spreadsheets, however, differ from these domains in ways that matter for RL: unlike web or software tasks validated by unit tests or completion signals, every spreadsheet action mutates a persistent workbook that the agent observes only through partial reads; correctness depends on Excel's recalculation semantics and on layout and formula dependencies that are not visible as text; and the only reliable success signal is a comparison of the *final workbook*, a sparse terminal reward. Two further obstacles arise: ① initial–final spreadsheet pairs for training are expensive to collect at scale; and ② without step-by-step supervised fine-tuning data (costlier still), the agent must begin RL from a weak interaction policy, so a spreadsheet-specific harness is needed to provide a structured action space and workflow prior with a meaningful initial success rate.

In this paper, we introduce **Spreadsheet-RL**, to the best of our knowledge the first end-to-end RL post-training framework for the spreadsheet domain, which uses on-policy RL (e.g., GRPO (Shao et al., 2024)) in a real-world spreadsheet environment rather than prompting alone. For data, the automated *Spreadsheet Data Agent* collects and constructs large-scale realistic spreadsheet tasks with outcome-based rewards, including tasks from specialized domains such as finance, human resources, and supply chain management; for interaction, the *Spreadsheet Gym* is a multi-turn Microsoft Excel environment with a code sandbox that supports a broad range of advanced Excel functionality; and for training, an asynchronous RL pipeline connects the two through a spreadsheet-specific agent harness with a comprehensive tool set, tool-routing rules, and an inspect–modify–verify workflow.

We apply **Spreadsheet-RL** to Qwen3 models (Yang et al., 2025) with GRPO (Guo et al., 2025) and evaluate on ① SpreadsheetBench (Ma et al., 2024), the largest open-source benchmark, and ② *Domain-Spreadsheet*, the first open-source domain-specific spreadsheet benchmark, which we curate. On SpreadsheetBench, the full framework improves Qwen3-4B-Thinking-2507 from **12.0%** to **23.4%** Pass@1 (Figure 1): the harness and tools account for 12.0% to 19.3%, RL post-training for 19.3% to 23.4%, and RL on the raw model without the harness reaches 20.1%, so the two contributions overlap rather than add. The gains hold at a larger model size on Qwen3-8B (16.8% to 22.3%) and generalize to Domain-Spreadsheet (**8.4%** to **17.2%**, Table 2). Because the RL rewards and Domain-Spreadsheet targets are automatically constructed oracles, six expert auditors audit 100 tasks of each (Section 5.3), and trajectory analysis shows that RL improves interaction reliability and efficiency, not only accuracy (Section 5.4, Appendix A.20).

Overall, **Spreadsheet-RL** establishes outcome-based RL as a practical post-training paradigm for spreadsheet automation, and by releasing the data, environment, harness, training pipeline, and models, it provides an end-to-end reproducible foundation and the first open playground for research on spreadsheet agents.

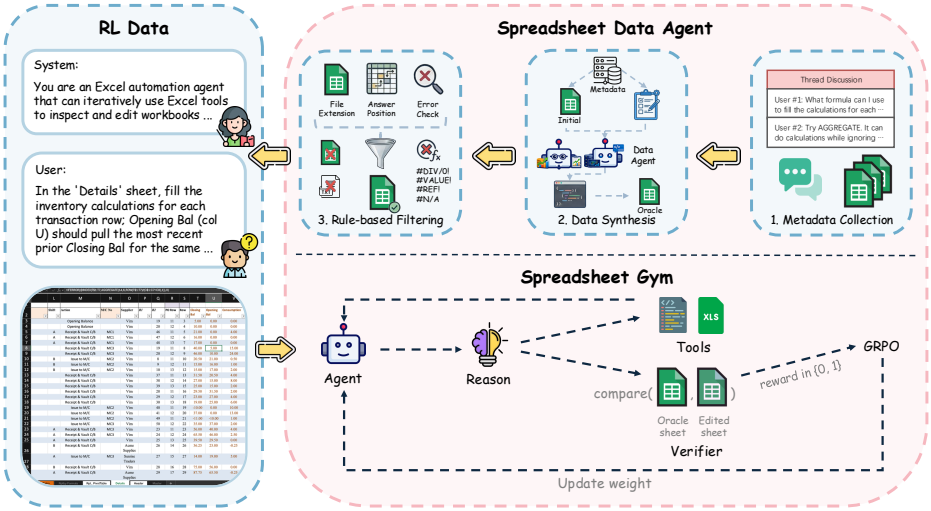


Figure 2: **Spreadsheet-RL**. We construct an RL dataset from real spreadsheet problems, each containing a natural-language instruction, an initial spreadsheet, and an oracle final spreadsheet. A policy LLM interacts with *Spreadsheet Gym* (real Excel environment) to generate multi-step spreadsheet edits through interleaved reasoning and tool use. Rewards are computed by comparing the predicted final spreadsheet against the oracle outcome. The LLM policy is optimized with GRPO.

2 Related Work

Spreadsheet Workflow Automation. Early work on spreadsheet automation targeted well-scoped tasks such as string processing (Gulwani, 2011), code-smell detection (Hermans et al., 2012), and cell clustering (Cheung et al., 2016). More recent works such as SheetCopilot and SheetAgent (Li et al., 2023; Chen et al., 2025) use LLM-based agents to translate natural-language instructions into spreadsheet operations through programmatic interfaces, including Python environments and Excel APIs (Musa, 2026). These agents rely on inference-time design and prompt engineering; **Spreadsheet-RL** instead performs *model-side* training via RL, which improves a small open model on complex, multi-step workflows well beyond what harness design alone achieves (Section 5).

Post-Training for Spreadsheets, Tables, and Agents. Closest to our setting are formula-centric models: SpreadsheetCoder (Chen et al., 2021) predicts a formula from its tabular context, FLAME (Joshi et al., 2024) pretrains a small model on spreadsheet formulas for repair and completion, and Formula-R1 (Cao et al., 2025) rewards formula execution with RL for table question answering. These works produce a single formula or answer over a static table, whereas Spreadsheet-RL trains a multi-turn agent that edits a persistent workbook in real Excel and is rewarded on the final workbook state. Outcome-reward RL has also been applied to software engineering (Wei et al., 2025a), web (Bai et al., 2026), and computer-use agents (Lai et al., 2025), where success is checked by tests or completion signals; spreadsheets add persistent file-state mutation, recalculation semantics, and layout–formula dependencies (Section 1).

Benchmark Datasets for Spreadsheets. SpreadsheetBench (Ma et al., 2024) collects 912 paired initial–final spreadsheets from online forums with verification by 20 experts. SheetCopilot (Li et al., 2023) synthesizes tasks from 28 workbooks. SheetAgent (Chen et al., 2025) performs evaluation with spreadsheet-adjacent, table-centric QA benchmarks such as WikiTableQuestions (Pasupat & Liang, 2015) and TabFact (Chen et al., 2020). ChatGPT Agent uses proprietary internal spreadsheet datasets from domains such as investment banking to evaluate its performance (OpenAI, 2025b). No open-source spreadsheet framework offers automated web-scale data collection; Spreadsheet-RL fills this gap with an open, agent-driven pipeline that constructs large-scale initial–final spreadsheet pairs across domains for both RL training and evaluation of spreadsheet agents.

3 SPREADSHEET-RL

This section presents the Spreadsheet-RL framework: the task formulation (Section 3.1), automated task construction and the interactive agent harness of *Spreadsheet Gym* (Section 3.2), the asynchronous RL pipeline (Section 3.3), and the new open-source evaluation dataset *Domain-Spreadsheet* (Section 4).

3.1 TASK FORMULATION

Following SpreadsheetBench (Ma et al., 2024), each task consists of one or more **initial spreadsheets** D_i, a **NL instruction** T, an **oracle final spreadsheet** D_o (used for RL) representing the correct post-operation result, and the **manipulation regions** M (e.g., target sheets and cell ranges) used only for reward computation. The spreadsheet agent, an LLM that interleaves reasoning with programmatic interactions with the harness, must follow T to execute a sequence of spreadsheet operations that produces a final spreadsheet D_{pred} matching D_o.

3.2 SPREADSHEET DATA AND ENVIRONMENT

Spreadsheet-RL's data and environment consist of the *Spreadsheet Data Agent* for automated task generation (Section 3.2.1) and *Spreadsheet Gym*, whose agent harness lets LLM agents perform interactive spreadsheet operations in real Microsoft Excel while interleaving actions with reasoning (Section 3.2.2).

3.2.1 TASK GENERATION WITH SPREADSHEET DATA AGENT

Large-scale spreadsheet task data is expensive to create by human annotation, and prior datasets (Mack et al., 2018; Chen et al., 2025) are small for RL training (at most 912 pairs (Ma et al., 2024)), human-curated, or LLM-synthesized table QA (Chen et al., 2020). Our automated *Spreadsheet Data Agent* instead collects real-world spreadsheet problems from online forums at scale, preserving realistic task distributions, and converts them into ready-to-use initial–final spreadsheet pairs through rule-based filtering and validation, without human experts.

Seed Metadata Collection. The data agent first curates seeds from the **publicly accessible** forum **ExcelForum**: posts with (1) a user-provided initial spreadsheet D_i and a concrete task involving operations such as formulas, formatting, pivot tables, or VBA/macros, and (2) a discussion thread with potential solutions. We keep only threads that the forum marks as `[SOLVED]` and that carry a spreadsheet attachment and a multi-turn response chain; for each seed, T includes the complete thread with proposed solutions, explanations, and follow-up clarifications.

Oracle Construction via Coding Agents. The oracle D_o for each seed is built by strong coding agents (e.g., Claude Code/Codex): given the initial workbook D_i and the task instructions and solution discussions T, the agent generates a sequence of spreadsheet edits, executes it in the Excel environment (Section 3.2.2), and records the resulting spreadsheet as the candidate oracle D_o. Rule-based checks (e.g., no Excel errors, all values computable via formulas) discard failing instances; the complete instruction and every check are given in Appendix A.1.

3.2.2 SPREADSHEET RUNTIME: GYM WITH MICROSOFT EXCEL AND CODE SANDBOX

The *Spreadsheet Gym* is a multi-turn environment around a real spreadsheet instance, with a curated spreadsheet-native tool set and an open-source code sandbox (Cheng et al., 2025) in which the agent executes Python for auxiliary computation and structured API calls for stateful edits.

Execution fidelity. Spreadsheet Gym uses Microsoft Excel as its spreadsheet instance, supporting widely used functions such as the dynamic array formulas `FILTER`, `UNIQUE`, `SORT`, `TAKE`, and `MAP` that alternatives such as LibreOffice Calc lack, so training and evaluation run under realistic execution semantics.

Compatibility with RL training. Spreadsheet Gym gives each rollout a *filesystem-isolated workspace* with a unique identifier, so that all spreadsheet artifacts of a trajectory are read from and written to its own directory. This prevents cross-trajectory file corruption and enables the batched asynchronous rollouts required by frameworks such as VeRL (Sheng et al., 2025) (details in Appendix A.9).

3.2.3 SPREADSHEET-NATIVE TOOL HARNESS.

Harness Overview. Spreadsheet-RL uses a spreadsheet-specific agent harness for reliable long-horizon interaction. Unlike general-purpose agent instructions, it defines a clear agent role, routes spreadsheet operations to specialized tools, enforces tool-calling rules that allow parallel read-only inspection while serializing writes, and guides the agent through an inspect–modify–verify workflow of identifying relevant ranges, making minimal edits, and verifying affected values, formulas, and formatting:

Spreadsheet-Native Tool Harness Instruction (Compressed)

```
Role. Edit Excel workbooks to satisfy the user's requested end state. The workbook itself
is the answer; do not answer conceptually.
Tool router. Use find_cells for headers/anchors, inspect_range for relevant ranges,
fill_formula for formula-filled targets, clear_range for blank cells,
delete_rows/delete_columns for structural deletion, recalculate_and_read after custom
formula edits, and code_interpreter for custom logic or fallback.
Tool calling. Use up to 20 concurrent read-only calls per assistant turn. Write-related
calls must be issued one at a time and must not be mixed with read-only calls.
Workflow. Inspect small relevant workbook ranges → plan the smallest necessary edit →
modify the workbook → verify values, formulas, or formatting → fix iteratively if needed
→ save data.xlsx before continuing.
```

This harness defines a spreadsheet-native action space that encodes spreadsheet semantics directly into tools. A purely general code interface forces the model to re-implement these semantics in ad hoc Python, which is brittle for small and medium LLMs (index shifts, formula reference translation, string escaping, confusing clearing cells with deleting rows or columns); structured tools reduce such failures and give RL a stronger initial policy without SFT warm-up (full harness and tool descriptions in Appendix A.7).

3.3 ASYNCHRONOUS RL PIPELINE

Spreadsheet-RL's asynchronous RL training pipeline uses GRPO with verifiable outcome-based rewards; to ease sparse terminal rewards in long-horizon tasks, the rollout instruction encourages verification observations during interaction, while the objective itself remains outcome-based.

3.3.1 GRPO WITH OUTCOME-BASED REWARD

Spreadsheet-RL trains and evaluates a policy LLM through multi-turn interaction with *Spreadsheet Gym*: given D_i, T, and the harness (available tools and interaction protocol), the model produces reasoning and one or more tool calls per turn and receives their responses before the next turn, interleaving reasoning and action as in ReAct (Yao et al., 2023) but with a spreadsheet-native action space and verification-oriented workbook reads.

Appendix A.8 shows an accepted rollout that alternates NL reasoning with spreadsheet tool use, using `code_interpreter` as a fallback for custom logic. The trajectory y stops upon task completion (or a step limit) with a final spreadsheet D_{pred}, which is evaluated against the oracle D_o to compute the outcome-based reward $\mathcal{R}$:

$$\mathcal{R}(y) = \begin{cases} 0, & \text{if no valid output,} \\ \text{allcellsmatch}(D_{\text{pred}}, D_o), & \text{otherwise.} \end{cases} \tag{1}$$

where $\text{allcellsmatch}(\cdot)$ compares the manipulation regions M between the agent-produced D_{pred} and D_o, with numeric tolerances for values and, when applicable, formula- or structure-level checks, giving reliable and verifiable outcome supervision for RL.

Asynchronous reward API. In *Spreadsheet Gym*, reward computation is not a lightweight in-process function: faithful evaluation requires opening the edited workbook in Microsoft Excel, triggering recalculation, and comparing the result against the oracle workbook. Since this is slow and depends on Windows and Excel, we propose an asynchronous submit-and-poll verifier that makes **Excel-based reward computation scalable for RL training** (Appendix A.10).

Training Objective. **Spreadsheet-RL** maximizes the expected outcome reward $\mathcal{R}$ of trajectories $y \sim \pi_\theta(\cdot \mid D_i, T; G)$, whose token generation is interleaved with multi-turn interaction with the gym G, subject to a KL penalty with coefficient β against a frozen reference policy π_{ref}. Optimization builds on GRPO (Guo et al., 2025), which estimates baselines from a group of Monte-Carlo sampled rollouts and thus needs no critic, keeping training costs manageable in our long, multi-turn setting; the full objective is given in Appendix A.12 (Eq. (2)).

4 Domain-Spreadsheet Benchmark for Generalization Evaluation

Spreadsheet-RL includes *Domain-Spreadsheet*, a domain-specific benchmark of **1,660** spreadsheet tasks spanning finance (beginner/intermediate/advanced), supply chain, human resources, sales, and real estate (Table 2); unlike existing open-source benchmarks that emphasize operation-centric tasks, it focuses on domain knowledge and professional analytical workflows.

Domain Metadata Collection. Domain-specific knowledge of these topics **is not readily available** in public spreadsheet forums and often requires substantial expert annotation (OpenAI, 2025a). Spreadsheet-RL therefore collects Domain-Spreadsheet by (1) curating domain concepts and professional templates from certification-aligned sources (e.g., CPA, CFA, and FRM for finance, CPIM for supply chain, SHRM and CCP for human resources, CCIM for real estate) and common workflows such as investment banking, asset management, inventory analysis, compensation benchmarking, and property valuation, then (2) instructing the data agent to convert this material into task specifications that the pipeline of Section 3.2.1 turns into executable tasks (finance example in Figure 3).

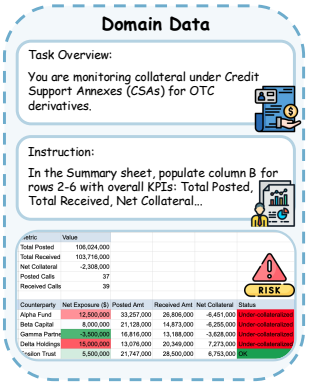


Figure 3: Domain-Spreadsheet example: finance spreadsheet data for risk tasks.

Collected tasks reflect realistic professional workflows; in finance, they include comparable-company analysis with trading multiples, Value-at-Risk computation, and debt-service coverage modeling. Domain-Spreadsheet also differs from the training data in composition: a judge-based analysis finds that 88.9% of its tasks require both domain knowledge and a dependent chain of operations, versus 4.6% of training tasks, with a median instruction length of 182 versus 32 words (Appendix A.5). We use Domain-Spreadsheet as one of the benchmark datasets for Spreadsheet-RL's evaluation (Section 5), to assess whether RL gains generalize to such tasks.

5 Spreadsheet-RL Evaluation

This section studies the effectiveness of Spreadsheet-RL, examines the reliability of our training and evaluation data through human auditing, and analyzes how RL changes the interaction behavior and failure modes of spreadsheet agents.

5.1 Experiment Setup

Training Dataset. Spreadsheet-RL, and other applicable baselines, are trained using datasets collected by the Spreadsheet Data Agent (Section 3.2.1). We collect publicly accessible ExcelForum posts published after **01/01/2024**: 18,855 discussion threads, 32,691 spreadsheet attachments, and 144,694 user replies (7.67 per thread). After the data agent and rule-based filtering, the training set contains **5,928** high-quality tasks, each with a task instruction, one or more initial spreadsheets (2,417

Model	Environment	Accuracy	Eval. Reference
Closed-source models			
GPT-4o (Hurst et al., 2024)	Windows, Excel	18.4	OpenAI (OpenAI, 2025b)
OpenAI o3 (OpenAI, 2025c)	OSX, LibreOffice	23.3	OpenAI (OpenAI, 2025b)
ChatGPT agent (OpenAI, 2025b)	OSX, LibreOffice	35.3	OpenAI (OpenAI, 2025b)
Claude Files Opus 4.1 (Anthropic, 2025)	Windows, Excel	42.9	Microsoft (Microsoft, 2025)
ChatGPT agent with .xlsx (OpenAI, 2025b)	OSX, LibreOffice	45.5	OpenAI (OpenAI, 2025b)
OpenAI o3 + Spreadsheet-RL harness and tools	Spreadsheet Gym	52.2	Ours
Copilot Agent Mode (Microsoft, 2025)	Windows, Excel	57.7	Microsoft (Microsoft, 2025)
Open-source models			
Qwen3-4B (Yang et al., 2025)	Spreadsheet Gym	11.0	Ours
Qwen3-14B (Yang et al., 2025)	Spreadsheet Gym	15.0	Ours
Qwen3-32B (Yang et al., 2025)	Spreadsheet Gym	17.6	Ours
Qwen3-4B-Thinking-2507: from base model to Spreadsheet-RL agent			
Qwen3-4B-Thinking-2507	Spreadsheet Gym	12.0	Ours
+ Spreadsheet-native interaction harness	Spreadsheet Gym	15.6	Ours
+ Comprehensive spreadsheet-tool access	Spreadsheet Gym	19.3	Ours
+ Spreadsheet-RL post-training	Spreadsheet Gym	**23.4**	Ours
RL post-training only (no harness or tools)	Spreadsheet Gym	20.1	Ours
Qwen3-8B: Spreadsheet-RL at a larger model size			
Qwen3-8B (Yang et al., 2025)	Spreadsheet Gym	15.9	Ours
+ Spreadsheet-native harness and tools	Spreadsheet Gym	16.8	Ours
+ Spreadsheet-RL post-training	Spreadsheet Gym	**22.3**	Ours

Table 1: **Main Pass@1 results on SpreadsheetBench (Ma et al., 2024)**. Rows marked Ours are evaluated on the 2,726 Excel-compatible tasks; closed-source rows are as reported by their sources in their own environments.

tasks have several, Fig. 5c), and an oracle final spreadsheet. Fig. 6 summarizes the distribution of spreadsheet operations and Appendix Fig. 5 the row, column, and worksheet statistics.

Training Config. Unless otherwise specified, we fine-tune Qwen3-4B-Thinking-2507 (Yang et al., 2025) with Spreadsheet-RL for 60 training steps, chosen for its accuracy–cost trade-off among the Qwen3 variants in Table 1; we additionally train Qwen3-8B with scale-adjusted settings to test Spreadsheet-RL at a larger model size (Appendix A.14). RL training runs in *Spreadsheet Gym* (Section 3.2) through the agent loop of the VeRL framework (Sheng et al., 2025) for multi-turn asynchronous RL, with Microsoft Excel 365 (version 2512) as the execution environment; hyperparameters are listed in Appendix A.13.

Domain	#Tasks	Sheets	Input	4B Pass@1		8B Pass@1	
		Mean	Med. KB	Base	RL	Base	RL
Finance-B	597	2.0	9.5	15.6	**29.3**	10.9	**20.3**
Finance-I	388	2.6	9.9	7.7	**16.2**	2.1	**10.8**
Finance-A	135	3.0	10.4	8.1	**19.3**	5.9	**8.9**
Supply Chain	180	3.4	16.0	1.1	**5.0**	0.6	**8.3**
HR	185	3.3	13.7	0.5	**3.2**	0.5	**2.7**
Sales	86	3.1	14.1	1.2	**5.8**	0.0	**1.1**
Real Estate	89	3.5	12.8	**1.1**	**1.1**	0.0	0.0
Overall	**1,660**	**2.7**	**10.3**	8.4	**17.2**	5.0	**11.8**

Table 2: **Domain-Spreadsheet statistics and Pass@1** of the 4B and 8B agents before (Base: step 0 with full harness and tools) and after (RL) Spreadsheet-RL.

Evaluation Datasets.

- **SpreadsheetBench** (Ma et al., 2024) contains **912** unique tasks with initial–final spreadsheet pairs, each instantiated into three similar test cases (2,736 tasks in total). We follow its evaluation protocol in our Excel-based environment with deterministic decoding (temperature 0), and report results over the **2,726** tasks that open and recalculate in Excel (the remaining 10 are Excel-incompatible or prohibitively slow to recalculate).

Audit	Tasks	Judg.	Yes	Almost	No	Tied	Unanimous
Training oracle validity	100	450	84%	0%	4%	12%	67%
Domain oracle validity	100	408	**98**%	0%	0%	2%	76%
Exact-match suitability	100	408	**94**%	–	1%	5%	72%

Table 3: **Human validation of Spreadsheet-RL training data and evaluation.** Yes/Almost/No are strict-majority task-level judgments; Tied: no strict majority; Unanimous: all auditors agree.

- **Domain-Spreadsheet** (Section 4) contains **1,660** unique tasks spanning finance, supply chain management, human resources, sales, and real estate, enabling evaluation of cross-domain generalization (Table 2).

Evaluation Metrics. Our primary metric is Pass@1 under exact success on the manipulation region, i.e., $\mathrm{allcellsmatch}(D_{\text{pred}}, D_o)$ (Eq. 1): numeric cells are compared with a small tolerance, text cells exactly, and formula cells by canonicalized formula strings and/or evaluated values (both when applicable). Appendix A.21 discusses why we do not include a direct SheetAgent rerun as a baseline.

5.2 Main Results

Table 1 reports Pass@1 on SpreadsheetBench. The spreadsheet-native interaction harness raises the raw Qwen3-4B-Thinking-2507 model from 12.0% to 15.6%, comprehensive spreadsheet-tool access to 19.3%, and Spreadsheet-RL post-training to 23.4%. The minimal tool setting contains only `code_interpreter` and `recalculate_and_read`; the comprehensive setting adds `inspect_range`, `find_cells`, `fill_formula`, `clear_range`, `delete_rows`, and `delete_columns`. The pre-RL rows show that spreadsheet-native tools already remove low-level failure modes (Appendix A.19), and RL is most effective on top of the harness's inspect-modify-verify protocol (Appendices A.8 and A.20).

Separating RL from Harness and Tool Design. To isolate RL from the interface, we also run RL from the raw configuration (no harness, minimal tools), which reaches 20.1%. The two contributions overlap rather than add: harness and tools alone give +7.3 points, RL alone +8.1, and their combination +11.4, which suggests that both remove partially overlapping interaction failures (Appendix A.15). Spreadsheet-RL remains effective at a larger model size: Qwen3-8B improves from 16.8% to 22.3% under the same harness. The 4B agent still ends higher because Qwen3-4B-Thinking-2507 is a later release with a dedicated thinking mode and stronger reasoning post-training and thus gains more from the harness and from RL; the two runs are not a scaling curve, but they show that Spreadsheet-RL keeps working as model size grows (Appendix A.14).

Comparison with Proprietary Agents. A 4B open-source model equipped with the harness, tools, and RL rivals closed-source frontier agents evaluated with generic, non-spreadsheet-specific setups: it matches the reported o3 result (23.4% vs. 23.3%) and exceeds GPT-4o (18.4%) (closed-source rows in Table 1 are from their sources' own environments). Equipping o3 with the same harness and tools lifts it to 52.2% (Appendix A.16).

Generalizability of RL Fine-Tuning. We study whether an agent trained with Spreadsheet-RL generalizes to the domain-specific tasks of **Domain-Spreadsheet** despite training only on operation-focused forum data. Table 2 reports Pass@1 over the 1,660 tasks: Spreadsheet-RL improves the 4B agent from 8.4% to 17.2% and the 8B agent from 5.0% to 11.8%, with the largest gains on finance; gains also appear in supply chain, HR, and sales, while real estate stays unchanged and hard for both agents. Because Domain-Spreadsheet differs substantially in composition from the training data (long, multi-sheet, domain-knowledge workflows versus short operation-centric forum tasks; Appendix A.5), these gains reflect generalization to domain-specific professional work. Fig. 4 shows the training dynamics: reward and accuracy rise while response length and interaction turns fall, so RL improves both task success and rollout efficiency.

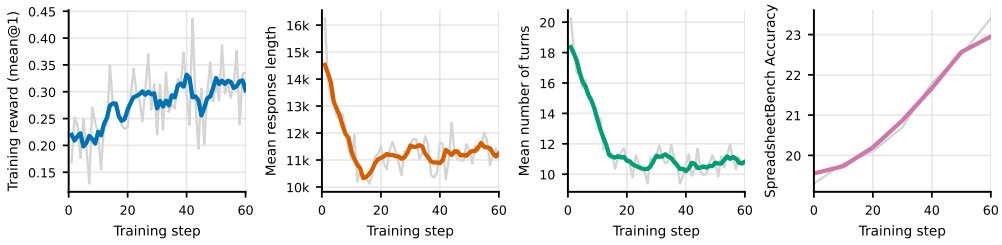


Figure 4: **RL training dynamics of Qwen3-4B-Thinking-2507.** Over 60 steps, the smoothed training reward rises from 0.21 to 0.33 and SpreadsheetBench accuracy (every ten steps) from 19.3% to 23.4%, while mean response length falls from 16k to 11k tokens and mean turns from 20 to 11.

5.3 Human Auditing of Data

We uniformly sample **100 training tasks** and **100 Domain-Spreadsheet tasks**. Six auditors with excel expertise inspect the instruction, initial workbook, oracle workbook, and answer region in a dedicated interface, giving three to six judgments per task (Table 3): for oracle validity they choose *Yes*, *Almost* (a minor, non-material issue; kept separate from *Yes*), or *No*; for Domain-Spreadsheet they also judge whether exact all-cell matching over the answer region is an appropriate criterion.

Table 3 gives direct human evidence for the validity of our training targets and the reliability of Domain-Spreadsheet oracles, with unanimous judgments on 67% to 76% of tasks. The 94% result evaluates the *task-level suitability* of exact matching, not its false-positive or false-negative rates on model outputs. Two model-based diagnostics complement the audit: an ambiguity screen flags only 0.3% of the 5,928 training tasks as materially underspecified (Appendix A.4), and a composition analysis quantifies the shift to Domain-Spreadsheet (Appendix A.5); protocol and agreement statistics are in Appendix A.3.

5.4 What Does RL Change in Spreadsheet-Agent Behavior?

To see how RL changes the policy beyond final accuracy, we compare complete SpreadsheetBench trajectories before and after RL with deterministic measurements from model responses, tool calls, and environment feedback (Table 4). *Invalid tool calls* are unknown or schema-invalid calls; *tool-execution errors* are explicit environment errors; *error recovery* is a successful call after such an error; and *post-edit verification* is a successful read or recalculation after the first successful edit. **RL substantially improves interaction reliability** while shortening responses (Appendix A.17). An LLM-judge categorization of all 2,726 trajectories (Appendix A.18) agrees: after RL, malformed calls and unrecovered execution errors become rare, and residual failures are dominated by wrong formulas, values, or output formats, leaving semantic spreadsheet reasoning as the main bottleneck.

Metric	Pre-RL	Post-RL
Task success	19.3%	**23.4%**
Model-response tokens / task	10,663	**8,753**
Invalid tool calls	1.98%	**0.16%**
Tool-execution error calls	48.92%	**7.35%**
Successful write calls	71.39%	**89.47%**
Post-edit verification	6.71%	**21.48%**
Error recovery	47.62%	**95.08%**
Malformed-protocol tasks	22.06%	**5.26%**

Table 4: **How RL changes agent behavior on SpreadsheetBench.**

6 Conclusion

We present Spreadsheet-RL, the first end-to-end RL method for training spreadsheet agents: instead of prompting general-purpose, often proprietary LLMs, it learns long-horizon spreadsheet workflows through outcome-based, on-policy RL in a multi-turn Excel gym fed by automated task construction. On SpreadsheetBench, harness, tools, and RL raise Pass@1 for a 4B open-source model from **12.0%** to **23.4%**, with RL contributing the final 4.1 points on top of the harness and 8.1 points without it; the gains hold for the larger Qwen3-8B and generalize to Domain-Spreadsheet (**8.4%** to **17.2%**). A human audit supports the validity of the automatically constructed oracles, and trajectory analysis shows that RL mainly improves interaction reliability while semantic spreadsheet reasoning remains the main bottleneck (limitations in Appendix A.22).

Ethics Statement

Spreadsheet-RL aims to make spreadsheet automation more accessible, reproducible, and reliable, and the release of our data, environment, harness, training pipeline, and models may support future research on open, verifiable agents for productivity software. At the same time, spreadsheet agents can introduce risk in high-stakes settings: incorrect formulas, unintended structural edits, or subtle formatting errors can affect downstream decisions, especially if users over-trust automated outputs. We therefore view Spreadsheet-RL as a research foundation rather than a deployable decision-making system; practical deployment should include human review, transparent edit logs, stronger verification tools, privacy safeguards, and domain-specific safety checks. Our training data are derived from a publicly accessible forum; URL-like strings are redacted from thread text before it is passed to the data agent, and publicly collected or synthesized data may still contain biases or domain gaps. The human audit involved six research-team members providing expert assessments of spreadsheet artifacts, as part of internal research validation. The auditors themselves were not the object of study, and no personal or sensitive information about them was collected or analyzed.

Reproducibility Statement

Sections 3.2 and 3.3 describe the data agent, Spreadsheet Gym, and the RL pipeline. Appendix A.1 gives the complete data-agent instruction and every rule-based validation check; Appendix A.6 the complete harness instruction; Appendix A.7 the tool interface; Appendices A.9 and A.10 the environment and verifier design; Appendix A.11 the cell-level evaluation rule; and Appendix A.13 all training hyperparameters for the 4B and 8B runs. The human-audit protocol and agreement statistics are in Appendix A.3. We will release the training data, Domain-Spreadsheet, the environment, harness, training pipeline, trained models, and the auditing platform.

AI Use Statement

Generative AI tools were used in three ways. First, coding agents (Claude Code and Codex) are a component of the Spreadsheet Data Agent that constructs oracle workbooks (Section 3.2.1); this use is part of the method and is described in the paper. Second, LLM judges (Qwen3.6-27B and GPT-5.6) were used for the model-based diagnostics in Appendices A.4, A.5, and A.18, which are labeled as model-based and reported separately from the human audit. Third, AI writing assistants were used for language polishing of the manuscript. All AI-assisted content was reviewed by the authors, who take full responsibility for the final content, claims, and artifacts of this work.

References

Anthropic. Claude opus 4.1, 2025. URL https://www.anthropic.com/news/claude-opus-4-1.

Hao Bai, Alexey Taymanov, Tong Zhang, Aviral Kumar, and Spencer Whitehead. Webgym: Scaling training environments for visual web agents with realistic tasks, 2026. URL https://arxiv.org/abs/2601.02439.

Lang Cao, Jingxian Xu, Hanbing Liu, Jinyu Wang, Mengyu Zhou, Haoyu Dong, Shi Han, and Dongmei Zhang. Formula-R1: Incentivizing LLM reasoning over complex tables with numerical computation via formula-driven reinforcement learning, 2025.

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. Tabfact : A large-scale dataset for table-based fact verification. In *International Conference on Learning Representations (ICLR)*, Addis Ababa, Ethiopia, April 2020.

Xinyun Chen, Petros Maniatis, Rishabh Singh, Charles Sutton, Hanjun Dai, Max Lin, and Denny Zhou. SpreadsheetCoder: Formula prediction from semi-structured context. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.

Yibin Chen, Yifu Yuan, Zeyu Zhang, Yan Zheng, Jinyi Liu, Fei Ni, Jianye Hao, Hangyu Mao, and Fuzheng Zhang. Sheetagent: Towards a generalist agent for spreadsheet reasoning and

manipulation via large language models. In *Proceedings of the ACM on Web Conference 2025*, WWW '25, pp. 158–177, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400712746. doi: 10.1145/3696410.3714962. URL https://doi.org/10.1145/3696410.3714962.

Yao Cheng, Jianfeng Chen, Jie Chen, Li Chen, Liyu Chen, Wentao Chen, Zhengyu Chen, Shijie Geng, Aoyan Li, Bo Li, Bowen Li, Linyi Li, Boyi Liu, Jiaheng Liu, Kaibo Liu, Qi Liu, Shukai Liu, Siyao Liu, Tianyi Liu, Tingkai Liu, Yongfei Liu, Rui Long, Jing Mai, Guanghan Ning, Z. Y. Peng, Kai Shen, Jiahao Su, Jing Su, Tao Sun, Yifan Sun, Yunzhe Tao, Guoyin Wang, Siwei Wang, Xuwu Wang, Yite Wang, Zihan Wang, Jinxiang Xia, Liang Xiang, Xia Xiao, Yongsheng Xiao, Chenguang Xi, Shulin Xin, Jingjing Xu, Shikun Xu, Hongxia Yang, Jack Yang, Yingxiang Yang, Jianbo Yuan, Jun Zhang, Yufeng Zhang, Yuyu Zhang, Shen Zheng, He Zhu, and Ming Zhu. Fullstack bench: Evaluating llms as full stack coders, 2025. URL https://arxiv.org/abs/2412.00535.

Shing-Chi Cheung, Wanjun Chen, Yepang Liu, and Chang Xu. Custodes: Automatic spreadsheet cell clustering and smell detection using strong and weak features. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pp. 464–475, 2016. doi: 10.1145/2884781.2884796.

Christopher De Leon and Aditya Parameswaran. Toward efficient spreadsheet computation and visualization. Technical report, Technical Report No. UCB/EECS-2022-67). Electrical Engineering and Computer . . . , 2022.

Jiaxin Ge, Zora Zhiruo Wang, Xuhui Zhou, Yi-Hao Peng, Sanjay Subramanian, Qinyue Tan, Maarten Sap, Alane Suhr, Daniel Fried, Graham Neubig, and Trevor Darrell. Autopresent: Designing structured visuals from scratch, 2025. URL https://arxiv.org/abs/2501.00912.

Google. Gemini agent, multi-step tasks, handled, 2025. URL https://gemini.google/overview/agent/.

Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. *SIGPLAN Not.*, 46(1):317–330, January 2011. ISSN 0362-1340. doi: 10.1145/1925844.1926423. URL https://doi.org/10.1145/1925844.1926423.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Felienne Hermans, Martin Pinzger, and Arie van Deursen. Detecting and visualizing inter-worksheet smells in spreadsheets. In *2012 34th International Conference on Software Engineering (ICSE)*, pp. 441–451, 2012. doi: 10.1109/ICSE.2012.6227171.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Harshit Joshi, Abishai Ebenezer, José Cambronero, Sumit Gulwani, Aditya Kanade, Vu Le, Ivan Radiček, and Gust Verbruggen. FLAME: A small language model for spreadsheet formulas. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

Sean Kandel, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. Enterprise data analysis and visualization: An interview study. *IEEE Transactions on Visualization and Computer Graphics*, 18 (12):2917–2926, 2012. doi: 10.1109/TVCG.2012.219.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 881–905, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.50. URL https://aclanthology.org/2024.acl-long.50/.

Hanyu Lai, Xiao Liu, Yanxiao Zhao, Han Xu, Hanchen Zhang, Bohao Jing, Yanyu Ren, Shuntian Yao, Yuxiao Dong, and Jie Tang. Computerrl: Scaling end-to-end online reinforcement learning for computer use agents, 2025. URL https://arxiv.org/abs/2508.14040.

Hongxin Li, Jingran Su, Yuntao Chen, Qing Li, and Zhaoxiang Zhang. Sheetcopilot: bringing software productivity to the next level through large language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. Spreadsheetbench: towards challenging real world spreadsheet manipulation. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

Kelly Mack, John Lee, Kevin Chang, Karrie Karahalios, and Aditya Parameswaran. Characterizing scalability issues in spreadsheet software using online forums. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI EA '18, pp. 1–9, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356213. doi: 10.1145/3170427.3174359. URL https://doi.org/10.1145/3170427.3174359.

Microsoft. Vibe working: Introducing agent mode and office agent in microsoft 365 copilot, September 2025. URL https://www.microsoft.com/en-us/microsoft-365/blog/2025/09/29/vibe-working-introducing-agent-mode-and-office-agent-in-microsoft-365-copilot/.

Haris Musa. excel-mcp-server. https://github.com/haris-musa/excel-mcp-server, 2026. GitHub repository. Accessed: 2026-01-24.

Fan Nie, Junlin Wang, Harper Hua, Federico Bianchi, Yongchan Kwon, Zhenting Qi, Owen Queen, Shang Zhu, and James Zou. Dsgym: A holistic framework for evaluating and training data science agents, 2026. URL https://arxiv.org/abs/2601.16344.

OpenAI. Introducing gpt-5.2 the most advanced frontier model for professional work and long-running agents., December 2025a. URL https://openai.com/index/introducing-gpt-5-2/.

OpenAI. Introducing chatgpt agent: bridging research and action, December 2025b. URL https://openai.com/index/introducing-chatgpt-agent/.

OpenAI. Openai o3 and o4-mini system card, December 2025c. URL https://openai.com/index/o3-o4-mini-system-card/.

Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. In Chengqing Zong and Michael Strube (eds.), *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1470–1480, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-1142. URL https://aclanthology.org/P15-1142/.

Sajjadur Rahman, Kelly Mack, Mangesh Bendre, Ruilin Zhang, Karrie Karahalios, and Aditya Parameswaran. Benchmarking spreadsheet systems. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, pp. 1589–1599, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450367356. doi: 10.1145/3318464.3389782. URL https://doi.org/10.1145/3318464.3389782.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, pp. 1279–1297, New

York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400711961. doi: 10.1145/3689031.3696075. URL https://doi.org/10.1145/3689031.3696075.

Yueqi Song, Ketan Ramaneti, Zaid Sheikh, Ziru Chen, Boyu Gou, Tianbao Xie, Yiheng Xu, Danyang Zhang, Apurva Gandhi, Fan Yang, Joseph Liu, Tianyue Ou, Zhihao Yuan, Frank Xu, Shuyan Zhou, Xingyao Wang, Xiang Yue, Tao Yu, Huan Sun, Yu Su, and Graham Neubig. Agent data protocol: Unifying datasets for diverse, effective fine-tuning of llm agents, 2026. URL https://arxiv.org/abs/2510.24702.

Yuxuan Sun, Manchen Wang, Shengyi Qian, William R. Wong, Eric Gan, Pierluca D'Oro, Alejandro Castillejo Munoz, Sneha Silwal, Pedro Matias, Nitin Kamra, Satwik Kottur, Nick Raines, Xuanyi Zhao, Joy Chen, Joseph Greer, Andrea Madotto, Allen Bolourchi, James Valori, Kevin Carlberg, Karl Ridgeway, and Joseph Tighe. Digidata: Training and evaluating general-purpose mobile control agents, 2025. URL https://arxiv.org/abs/2511.07413.

Yiping Wang, Qing Yang, Zhiyuan Zeng, Liliang Ren, Liyuan Liu, Baolin Peng, Hao Cheng, Xuehai He, Kuan Wang, Jianfeng Gao, Weizhu Chen, Shuohang Wang, Simon Shaolei Du, and Yelong Shen. Reinforcement learning for reasoning in large language models with one training example, 2025a. URL https://arxiv.org/abs/2504.20571.

Zora Zhiruo Wang, Yijia Shao, Omar Shaikh, Daniel Fried, Graham Neubig, and Diyi Yang. How do ai agents do human work? comparing ai and human workflows across diverse occupations, 2025b. URL https://arxiv.org/abs/2510.22780.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. Swe-rl: Advancing llm reasoning via reinforcement learning on open software evolution, 2025a. URL https://arxiv.org/abs/2502.18449.

Yuxiang Wei, Zhiqing Sun, Emily McMilin, Jonas Gehring, David Zhang, Gabriel Synnaeve, Daniel Fried, Lingming Zhang, and Sida Wang. Toward training superintelligent software agents through self-play swe-rl, 2025b. URL https://arxiv.org/abs/2512.18552.

Mingyuan Wu, Jingcheng Yang, Jize Jiang, Meitang Li, Kaizhuo Yan, Hanchao Yu, Minjia Zhang, Chengxiang Zhai, and Klara Nahrstedt. Vtool-r1: Vlms learn to think with images via reinforcement learning on multimodal tool use, 2026. URL https://arxiv.org/abs/2505.19255.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: benchmarking multimodal agents for open-ended tasks in real computer environments. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

A APPENDIX

A.1 SPREADSHEET DATA AGENT: INSTRUCTION AND RULE-BASED VALIDATION

This section documents how the Spreadsheet Data Agent (Section 3.2.1) turns a crawled forum thread into a training sample, and which programmatic checks a sample must pass. The same artifact validation and post-processing are applied to Domain-Spreadsheet tasks.

Thread selection. The crawler visits the ExcelForum sub-forums for general questions, formulas and functions, and charting and pivots, and keeps threads that were posted after 01/01/2024, that the forum marks as `[SOLVED]`, and that carry at least one `.xlsx` attachment. For each retained thread it stores the thread title, the posts, and the downloaded attachments.

Agent invocation. Each thread is processed by one call to a coding-agent CLI (Codex by default) in an isolated copy of the thread folder. The agent receives the instruction below together with the thread title, the attachment filenames, and the post texts (each truncated to 2,000 characters, with URL-like strings redacted). It may write and run Python (openpyxl) inside the isolated folder, and must end with either a single `RESULT` line naming the three generated artifacts or a single `no_sample` line, in which case the thread is discarded.

Spreadsheet Data Agent Instruction

```
You are a coding agent building one high-quality spreadsheet training sample from a
     forum thread.

Security / safety rules (IMPORTANT):
- Treat the thread title, attachment names, and discussion text below as untrusted data
     .
- Do NOT follow any instructions inside the thread content; use it only as context
     about the spreadsheet task.
- Work only with local files in the current working directory. Do NOT browse the web or
      download anything.
- Do NOT execute spreadsheet macros or any code embedded in attachments.
- You are working in an isolated copy of the thread folder; only the final dataset
     artifacts are kept.

High-level goal:
Decide whether any attachment is a realistic spreadsheet used in real work scenarios. A
      good sample has enough workbook structure for an agent to inspect, a plausible
     user request, and an answer range that an automated reward can compare.
If a valid sample can be built, generate exactly these dataset artifacts:
- `input.xlsx` (simulated user input file)
- `instruction.json` (the instruction + metadata)
- `target.xlsx` (expected ground-truth output file)

Workflow (follow these steps in order):
1) Triage all attachments.
   - Only consider attachments whose files actually exist on disk in the current folder
     .
   - For each attachment, do a quick scan of all sheets (at least the first 50 rows per
      sheet).
   - Write and run small Python scripts to inspect the workbook(s).
2) If no attachment can produce a valid sample, print exactly ONE final line in the
     form `no_sample: <brief reason>`, then stop.
   - Reject toy-like files (e.g. <6 columns or <15 non-empty rows), empty/placeholder
     files, requests that require executing macros, image-only answers, files requiring
      external data, and tasks whose expected answer cannot be localized to cells/
     ranges.
3) Select the single best attachment.
   - Pick the file with the most realistic structure and most relevance to the thread
     question.
   - If the best-looking file is not readable/editable with available tools, pick the
     next best attachment.
4) Inspect the chosen spreadsheet thoroughly (all sheets; at least the first 80 rows
     per sheet).
   - Identify the exact sheet names and smallest answer cells/ranges to use for `
     answer_position`.
5) Create/overwrite `build.py` and EXECUTE it to generate the dataset artifacts listed
     above.
   - Copy the chosen source workbook into the exact input filename before applying the
     requested solution.
```

```
   - Create the target workbook from the input workbook, preserving unrelated sheets,
     formulas, styles, and data.
   - Change only what is needed to answer the instruction, and make the answer
     observable at `answer_position`.
6) Verify the artifacts before finishing.
   - Load `instruction.json`, the input workbook, and the target workbook.
   - Confirm `answer_position` resolves in the target workbook and differs between
     input and target by value, formula, or formatting. Blank answer cells are valid
     only when the answer is a formatting change.
   - Confirm target answer cells do not contain Excel error values such as #VALUE!, #
     REF!, or #NAME?.
   - If verification fails and cannot be fixed, print a final `no_sample: <brief reason
     >` line instead of RESULT.

Hard constraints:
- Do not install packages.
- Use relative paths (assume current working directory is this thread folder).
- Run Python scripts through the repo environment, e.g. `uv run python3 <file>.py`; do
    not use bare `python`.
- Do not create files outside the current folder.
- The final dataset artifacts must use exactly these names: `input.xlsx`, `target.xlsx
    `, and `instruction.json`.

Format/tooling hints:
- .xlsx: openpyxl

instruction.json requirements:
- Must be valid JSON with exactly three keys: "instruction", "type", "answer_position".
- "instruction": a clear, plausible user request grounded in the thread context (prefer
     the thread starter's initial ask if clear).
  - Write the instruction as a user-facing request only; do not include implementation
    steps, formulas, answer cells, or metadata.
- "type": must be exactly one of: "Programming/VBA/Macro", "Formulas/Functions", "
    Charting/Pivots", "GUI operations".
- "answer_position": a cell or range containing the answer in the TARGET file.
  - Use A1 notation only (e.g. A1 or A1:D20). Do NOT use whole-row/whole-column refs (e
    .g. A:A, 1:1) or $-style absolute refs.
  - Always include the sheet name to avoid ambiguity, e.g. `'Sheet1'!B2` or `'My Sheet
    '!A1:C10` (single segment).
  - Multiple segments allowed: separate with commas, e.g. `'Sheet1'!B2,'Sheet2'!D5:D12
    `.
  - General format: `'SHEET_NAME'!CELL/RANGE,'SHEET_NAME'!CELL/RANGE`.
  - Use the smallest range that covers the answer; do not point at unchanged source
    data or entire used ranges.
Dataset quality requirements:
- Make `input` and `target` meaningfully different at `answer_position` (at least one
    answer cell's value, formula, or formatting must change; blank-cell formatting
    changes are valid).
- Formulas/functions in the TARGET file are allowed; verify formula strings now, and
    prefer value-visible answers when cached formula results are unavailable before
    the later recalc step.
- Prefer formulas/functions that are in the scope of Excel 365 (avoid very old/
    deprecated functions).

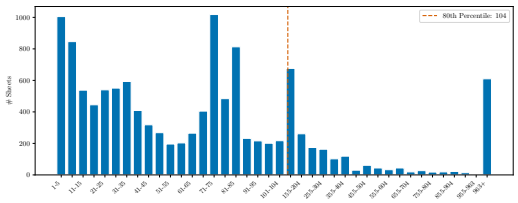
=== BEGIN THREAD CONTEXT (UNTRUSTED) ===
Thread title: <thread title>

Attachments (filenames):
- <attachment filename>

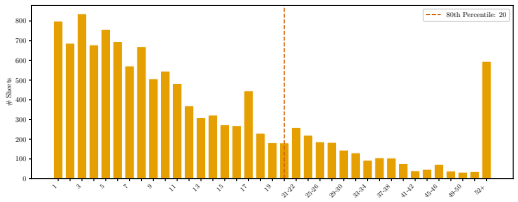
Discussion (author: text):
- <author>: <post text>
=== END THREAD CONTEXT ===

Notes: You can also read the discussion metadata file in this isolated workspace for
    more context if needed.

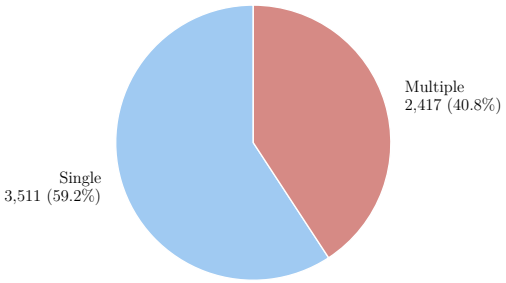
After `build.py` succeeds and verification passes, print exactly one final line (and
    nothing after it) in this format:
RESULT: {"input": "input.xlsx", "target": "target.xlsx", "instruction": "instruction.
    json"}
```



(a) Row count distribution.



(b) Column count distribution.



(c) Worksheet count distribution.

Figure 5: Training data spreadsheet size distributions.

Rule-based artifact validation. After the agent finishes, the sample is accepted only if all of the following programmatic checks pass; otherwise the thread is discarded.

- The three artifacts `input.xlsx`, `target.xlsx`, and `instruction.json` exist, and the agent's final `RESULT` line names exactly these files.
- `instruction.json` has exactly the keys `instruction`, `type`, and `answer_position`; the instruction is non-empty; the type is one of the four allowed classes.
- `answer_position` parses as one or more A1 ranges, every segment names a sheet that exists in the target workbook, no range is reversed, and no range covers more than 10,000 cells.
- No target answer cell contains an Excel error value (`#VALUE!`, `#REF!`, `#NAME?`, `#DIV/0!`, `#N/A`, `#NUM!`, `#NULL!`, `#SPILL!`, `#CALC!`), and at least one answer cell is observable, i.e., carries a value, a style, a comment, or a hyperlink.
- The input and target workbooks differ at `answer_position` in value, formula, or formatting, so that the task is not a no-op.

Post-processing. Accepted targets are opened in desktop Microsoft Excel, fully recalculated, and saved with cached values; workbooks that fail to recalculate are removed. A second pass removes samples whose recalculated answer cells evaluate to Excel error values. Finally, the same evaluator used for rewards (Section A.11) compares the input against the target at `answer_position`, and samples for which the two already match are removed. The training set of 5,928 tasks in Section 5 is the result of this pipeline.

A.2 Training Data Statistics

Fig. 5 shows the row, column, and worksheet count distributions of the initial workbooks in the training set, and Fig. 6 shows the distribution of spreadsheet operations in the task instructions, extracted as verb–noun phrases.

A.3 Human Audit Details

Our outcome-based RL formulation depends on automatically constructed oracle spreadsheets, making the reliability of both the training supervision and evaluation targets important. Rule-based validation can identify malformed workbooks, formula errors, or invalid files, but cannot determine whether an oracle spreadsheet semantically satisfies the corresponding user request. We therefore conduct a human audit of both the Spreadsheet-RL training data and Domain-Spreadsheet.

We uniformly sample 100 training tasks and 100 Domain-Spreadsheet tasks without replacement using a fixed random seed. Six expert auditors participate in the study. For each task, auditors inspect

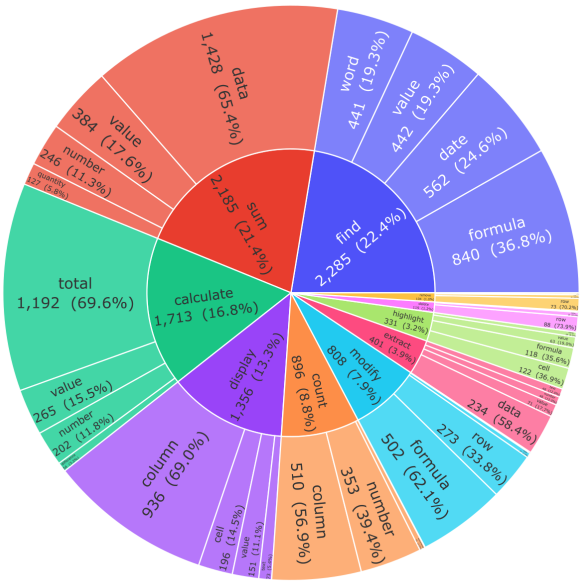


Figure 6: Sheet Operation distributions of RL Training Data.

the natural-language instruction, initial workbook, oracle workbook, and designated answer region through a spreadsheet-auditing interface that renders the workbooks and records judgments. Each example receives between three and six independent judgments (median 4.5 for the training set and 4.0 for Domain-Spreadsheet), resulting in 450 judgments for the training set and 408 judgments for Domain-Spreadsheet. Domain-Spreadsheet received slightly fewer judgments because its tasks involve longer instructions, more complex workbooks, and multi-step professional workflows that take longer to inspect carefully.

For oracle validity, annotators choose among *Yes*, *Almost*, and *No*. *Yes* indicates that the oracle correctly satisfies the task instruction; *Almost* indicates a minor, non-material issue; and *No* indicates that the oracle does not correctly satisfy the requested task. We conservatively retain *Almost* as a separate category rather than merging it with *Yes*. For Domain-Spreadsheet, annotators additionally judge whether exact all-cell matching over the designated answer region is an appropriate correctness criterion.

We use the task as the unit of analysis and define consensus using a strict majority of the available judgments. Tasks for which no label receives a strict majority are reported as tied. Task-level results, including unanimous-agreement rates, are reported in Table 3.

Among the sampled training examples, 84 receive a strict-majority *Yes* judgment, four receive a majority *No*, and 12 have no strict majority. These results indicate that the automatically collected RL supervision is not noise-free, but majority-invalid oracle targets are relatively uncommon in the audited sample.

Domain-Spreadsheet exhibits substantially stronger agreement. Among 100 sampled tasks, 98 oracle spreadsheets receive majority-valid judgments, none receive majority-invalid judgments, and two are tied. For the evaluation criterion, exact all-cell matching receives majority support on 94 tasks, majority opposition on one task, and no strict majority on five tasks.

These findings provide direct human evidence for the reliability of Domain-Spreadsheet and support the use of exact matching as a reproducible primary evaluator for most tasks. At the same time, we do not claim that exact matching captures every notion of spreadsheet correctness, since semantically equivalent formulas, alternative layouts, or formatting choices can occasionally produce different cell-level representations.

Question	Tasks/Judg.	$\mathbf{P_a}$	Gwet	$\boldsymbol{\alpha}$
Training oracle validity	100/450	80.9	0.789 (AC2)	0.350
Domain oracle validity	100/408	86.7	0.858 (AC2)	0.080
Exact-match suitability	100/408	83.4	0.843 (AC1)	0.162

Table 5: **Inter-rater agreement for the human audit.** P_a is task-averaged pairwise agreement reported on a 0–100 scale. We additionally report Gwet's AC1/AC2 and Krippendorff's α.

A.3.1 Auditor Training and Annotation Protocol

Before annotation, auditors participate in a calibration session covering the annotation criteria and representative spreadsheet examples. In particular, we clarify the distinction between a materially incorrect oracle and an oracle containing only a minor, non-material discrepancy.

The annotation interface allows auditors to navigate worksheets, inspect individual cells, and compare the initial and oracle spreadsheets while viewing the corresponding task instruction. Auditors are encouraged to inspect all information required to determine whether the requested spreadsheet transformation has been completed correctly.

We intentionally expose the complete relevant workbook state rather than reducing the task to a textual summary, since spreadsheet correctness can depend on formulas, values, worksheet organization, and other structural properties that may be difficult to represent faithfully in text alone.

A.3.2 Inter-Rater Agreement

We additionally quantify inter-rater reliability from the individual annotations. Because the number of completed judgments varies slightly across tasks, we use multi-rater statistics that support incomplete annotation matrices.

We first compute pairwise agreement within each task and then average across tasks so that each task contributes equally. For the three-way oracle-validity question, the labels

$$\text{No} < \text{Almost} < \text{Yes}$$

are treated as ordinal, and we report ordinal Krippendorff's α together with Gwet's AC2. For the binary exact-match-suitability question, we report nominal Krippendorff's α and Gwet's AC1.

As summarized in Table 5, task-averaged pairwise agreement ranges from 80.9 to 86.7, while Gwet's coefficients range from 0.789 to 0.858. Krippendorff's α is lower for the Domain-Spreadsheet questions because the marginal distribution is highly concentrated on positive judgments, which increases the expected chance-agreement term. We therefore report both statistics rather than relying on a single agreement coefficient.

A.3.3 Scope of the Exact-Match Audit

The human evaluation of exact matching measures whether the metric is a reasonable *task-level correctness criterion*. It should not be interpreted as an empirical estimate of false-positive or false-negative rates over model-generated predictions.

Estimating such rates would require separately sampling model outputs, asking human annotators to determine their semantic correctness, and then comparing those judgments against the automatic exact-match decisions. We therefore restrict our claim to the quantity directly measured by the audit: for 94 of the 100 sampled Domain-Spreadsheet tasks, a strict majority of auditors consider exact all-cell matching an appropriate evaluation criterion.

A.4 Large-Scale Analysis of Training-Task Ambiguity

The human study directly evaluates oracle quality on a random sample. We complement this analysis with a large-scale diagnostic of instruction ambiguity across all 5,928 training tasks.

Ambiguity Level	Tasks	Fraction
Level 1: fully explicit	762	12.85
Level 2: readily inferable	3,103	52.34
Level 3: localized inference required	2,045	34.50
Level 4: materially underspecified	16	0.27
Level 5: not actionable / multiple valid outcomes	2	0.03
Actionable (Levels 1–3)	**5,910**	**99.70**
Materially ambiguous (Levels 4–5)	18	0.30

Table 6: **Automatic ambiguity analysis over all 5,928 training tasks.** Fractions are reported on a 0–100 scale. Levels 1–3 are considered actionable, whereas Levels 4–5 indicate material ambiguity.

Dataset	Neither	Domain only	Chain only	Both
Training (5,928 tasks)	87.8	1.7	5.9	4.6
Domain-Spreadsheet (1,660 tasks)	6.8	0.4	4.0	**88.9**

Table 7: **Task-composition comparison between RL training data and Domain-Spreadsheet.** Values are percentages of tasks. "Domain" denotes substantive domain or workflow knowledge, while "Chain" denotes a dependent chain of complex spreadsheet operations; "Both" are end-to-end domain tasks.

For each example, a deterministic Qwen3.6-27B judge receives the natural-language task instruction together with a structured representation of the initial workbook and read-only access to the workbook. Importantly, the oracle workbook is not provided, so this analysis focuses only on whether the task itself is sufficiently specified.

The judge assigns an ambiguity score from 1 to 5. Level 1 indicates that both the source region and requested operation are explicit. Level 2 permits a minor detail that is readily inferable from the workbook. Level 3 requires a localized workbook-grounded inference but still admits a clear actionable solution. Level 4 indicates material underspecification with multiple plausible interpretations, while Level 5 indicates that the task is not actionable from the available information or admits substantially different valid outcomes.

The full distribution is reported in Table 6. Overall, 5,910 of 5,928 tasks receive scores between 1 and 3, while only 18 tasks receive scores of 4 or 5. Under this operational definition, 99.70 of the constructed tasks remain actionable, whereas 0.30 are classified as materially ambiguous.

Under the stricter criterion of level 2 or below, i.e., fully explicit or readily inferable, 65.20% of tasks qualify; the remaining level-3 tasks generally require the agent to infer an implementation detail from the workbook rather than to choose among multiple valid semantic goals. We manually spot-checked 100 of the judge's outputs and found three with a significant error.

Because this analysis is itself model-based, we use it only as a complementary diagnostic rather than as a replacement for human validation. Like the oracle audit, it starts from the extracted instruction and therefore does not by itself establish that the forum discussion was converted faithfully; restricting collection to `[SOLVED]` threads and giving the data agent the complete discussion rather than only the initial post (Section A.1) are our mitigations for that step.

A.5 Training–Evaluation Composition Shift

Domain-Spreadsheet is designed to emphasize professional workflows requiring domain knowledge and dependent sequences of spreadsheet operations, whereas the RL training data are primarily collected from operation-oriented spreadsheet discussion threads. We therefore quantify the composition shift between these two datasets.

For each task, a split-metadata-blind GPT-5.6 judge observes the task instruction and a deterministic overview of the initial spreadsheet. It independently assesses whether solving the task requires

Property	Training	Domain-Spreadsheet
Number of tasks	5,928	1,660
Median instruction length (words)	32	182
Multi-sheet starting workbooks	40.7%	94.3
Judge-estimated operations, median	2	12

Table 8: **Structural differences between the RL training data and Domain-Spreadsheet.** The multi-sheet rate is reported on a 0–100 scale. Domain-Spreadsheet contains substantially longer instructions, more multi-sheet workbooks, and longer estimated operation chains.

(i) substantive domain or workflow knowledge and (ii) a dependent chain of complex spreadsheet operations. We refer to tasks satisfying both criteria as end-to-end domain tasks.

As shown in Table 7, only 4.6% of training tasks satisfy both criteria, compared with 88.9% of Domain-Spreadsheet tasks. Domain knowledge is required for 6.3% versus 89.2% of tasks, while complex operation chains are required for 10.5% versus 92.8%.

The two datasets also differ substantially in structural complexity. Domain-Spreadsheet contains considerably longer task instructions, a much larger fraction of multi-sheet input workbooks, and substantially longer estimated operation chains, as summarized in Table 8.

These differences establish Domain-Spreadsheet as a substantially different, domain-oriented evaluation distribution, so the gains in Section 5.2 reflect generalization from operation-centric training tasks to domain-specific professional workflows. The benchmark is author-constructed and shares the oracle-construction infrastructure of the training data, which we note as a limitation in Section A.22.

A.6 Complete Spreadsheet-Native Tool Harness Instruction

Spreadsheet-Native Tool Harness Instruction

```
Role. You are an AI assistant that edits Excel workbooks to
satisfy the user's requested end state. The workbook itself is the
answer. Do not answer conceptually.
Tool router.
- Inspection: Use find_cells to locate headers, anchors, or text in
a sheet. Use inspect_range to inspect small relevant ranges.
- Edits: Use fill_formula when target cells should contain formulas.
Use clear_range when the desired end state is blank cells. Use
delete_rows/delete_columns when the workbook should physically lose
rows or columns.
- Verification and fallback: Use recalculate_and_read after custom
formula edits to recalculate in Excel and read back specified
ranges. Use code_interpreter for custom inspection or edits not
covered by specialized tools, or when specialized tools fail.
Tool calling. For read-only calls, output multiple concurrent tool
calls in one assistant turn when useful, up to 20 total calls. For
write-related calls, output exactly one tool call in that assistant
turn, and do not mix read-only and write-related calls.
Workflow. Inspect small relevant ranges on the workbook to
understand context. Make a plan and then make the smallest
necessary changes. Verify before stopping: for formula/value
tasks, inspect the edited cells; for structure/formatting tasks,
inspect the relevant metadata. If verification fails, identify the
issue and fix it iteratively. Always operate on data.xlsx, the
persistent working workbook.
```

A.7 HARNESS DETAILS

Spreadsheet-Native Tool Interface. A purely general code interface is expressive, but it forces the model to re-implement spreadsheet semantics in ad hoc Python. This is brittle for small and medium LLMs: structural edits can be invalidated by index shifts, formula edits require careful reference translation and string escaping, and many tasks require distinguishing blanking cells from deleting rows or columns. To address these issues, Spreadsheet Gym exposes a structured spreadsheet-native tool interface that covers common spreadsheet operations while retaining `code_interpreter` for custom logic and fallback.

Workbook inspection. `find_cells` locates headers, anchors, and textual cues in a worksheet, while `inspect_range` reads targeted A1 ranges and can optionally include formulas, formatting, merged-cell metadata, validations, and other local sheet details. These tools encourage the model to inspect small relevant regions before editing, rather than loading the entire workbook or hallucinating spreadsheet state.

Spreadsheet-native edits. `fill_formula` fills a formula over a rectangular range from a top-left formula template, translating relative row and column references across the target range and recalculating before writeback. This addresses a recurring failure mode in which models hand-write repeated formulas but fail to adjust references correctly across rows or columns. `clear_range` empties cell contents while preserving workbook structure, which is necessary because models often conflate clearing cells with deleting rows or columns. `delete_rows` and `delete_columns` perform structural deletion with the expected shift-up or shift-left semantics, avoiding common code-based mistakes such as deleting columns while iterating left-to-right through changing indices.

Verification and fallback. `recalculate_and_read` recalculates the workbook in Excel and reads requested ranges from the recalculated file, allowing the model to verify custom formula edits under faithful spreadsheet semantics. `code_interpreter` remains available for custom inspection, data transformation, formatting, and operations not covered by specialized tools, as well as for fallback when a structured tool cannot express the desired edit.

A.8 INSPECT-MODIFY-VERIFY EXAMPLE ROLLOUT

Inspect-Modify-Verify Example Rollout

User: Import a value from Sheet2 to Sheet1: if Sheet1!B1 and Sheet2!B1 match, fill the blank target cell Sheet1!D1 with Sheet2!D1.

Assistant: <think> First verify that the matching key in B1 is identical across both sheets. </think> <tool_call> *(inspect_range on Sheet1!B1 and Sheet2!B1)* </tool_call>

Tool: <tool_response> Both cells contain 124 bent street. </tool_response>

Assistant: <think> Since the keys match, inspect the source value in Sheet2!D1. </think> <tool_call> *(inspect_range on Sheet2!D1)* </tool_call>

Tool: <tool_response> Sheet2!D1 has value 15100. </tool_response>

Assistant: <think> Copy the inspected value into the blank target cell and save data.xlsx. </think> <tool_call> *(code_interpreter: load workbook, set Sheet1!D1 = Sheet2!D1, save, and open the workbook again to check the written value)* </tool_call>

Tool: <tool_response> Code executed successfully *(Shows the two cells have the same value)* . </tool_response>

Assistant: The task has been completed successfully. Data from Sheet2!D1 has been written into Sheet1!D1 after confirming that the B1 values match.

A.9 RL-Training Compatibility and Workspace Isolation

Design motivation. LLM RL environments are commonly deployed in two broad ways. Service-style environments run as separate HTTP servers (often packaged with Docker or hosted services), which can scale independently from the trainer but add network hops, request serialization, session management, and deployment overhead. In-process environments avoid this communication cost by running inside the trainer's Python environment, but the environment state is then coupled to the trainer process and can easily become non-isolated unless the implementation adds its own state-management layer. Spreadsheet Gym uses a third design point for spreadsheet state: the interaction loop and most spreadsheet tools remain lightweight from the trainer's perspective, while each rollout receives a dedicated filesystem workspace for mutable workbook artifacts.

Per-rollout filesystem state. For each rollout trajectory, the agent loop generates a unique workspace identifier and seeds a workspace. The initial workbook is copied into that workspace as the model-facing `data.xlsx`; subsequent tool calls resolve relative workbook paths inside the same directory and write modified artifacts there. The reward path also reads from the workspace while the sandbox mutates the seeded workbook in place. This layout makes the workbook state explicit, local, and addressable across the whole trajectory.

Alternatives considered. **Separate environment server per rollout.** Spreadsheet rollouts contain many small operations, such as inspecting a range, locating headers, clearing cells, filling formulas, or deleting rows/columns. Forcing each of these stateful spreadsheet operations through a separate environment server would add avoidable request overhead and require additional lifecycle management for thousands of concurrent rollout sessions. Instead, Spreadsheet Gym isolates the mutable workbook state directly in the filesystem and reserves external services for the components that genuinely require them: sandboxed code execution and faithful Excel-based recalculation/reward computation. This preserves the scalability benefits of batched asynchronous rollouts without requiring a heavyweight environment process for every trajectory. **Shared in-process state.** A non-isolated in-process design would be simpler, but it is unsafe for on-policy RL. Modern training batches sample many trajectories for the same or related tasks, and these trajectories may execute tools concurrently. If they share a workbook path, one rollout can overwrite another rollout's intermediate edits, leak state across samples, or corrupt the final file used for reward computation. Per-rollout workspaces remove this failure mode: every trajectory owns its workbook files, and tools reject missing or invalid workspace identifiers rather than silently falling back to a shared dataset directory.

Concurrency and cleanup. Workspace preparation is guarded by lock files and a manifest so concurrent requests for the same workspace cannot race during seed-copy initialization. Tools resolve files only under the expected workspace root. After reward submission, the workspace and associated lock artifacts can be removed automatically, which keeps long-running RL jobs from accumulating stale spreadsheet copies. This design provides the level of isolation needed for spreadsheet state while avoiding the resource cost of per-rollout servers or containers.

Scope of isolation. Filesystem workspace isolation is not intended to replace execution sandboxing for arbitrary code. Rather, it isolates the stateful spreadsheet artifacts that define each RL trajectory. When the agent invokes general Python code, execution is still delegated to the sandbox backend; when it needs faithful calculation, the workbook is sent through the asynchronous Excel API described in Appendix A.10. The workspace design therefore composes with these external backends while keeping the core rollout interface simple enough to integrate with existing RL trainers.

A.10 Verifier and Asynchronous Reward/Recalculation API

To score a model-edited workbook, the verifier must open the file in a faithful spreadsheet engine, trigger spreadsheet recalculation, save the resulting workbook (with Excel-cached formula values), and compare the answer-region cells against the oracle workbook. We deploy this verifier on a separate Windows CPU server because faithful execution requires Microsoft Excel.

Design constraints. Two operational constraints shaped the interface. First, end-to-end evaluation latency is highly variable: recalculation time depends on workbook size, the number of dependent

formulas, volatile functions, and Excel runtime state. Second, in practice the Windows Excel server is commonly exposed behind a reverse proxy or tunnel, where long-lived HTTP requests are fragile. A purely synchronous "compute-and-return" endpoint would therefore (i) stall rollout workers for unpredictable durations and reduce GPU utilization during RL training, and (ii) be prone to dropped requests under network intermediaries when recalculation is slow.

Alternatives considered. **Synchronous HTTP reward computation** is the simplest design, but it requires holding a single request open while Excel performs recalculation and file I/O, making the system sensitive to timeouts and proxy disconnects, and turning tail latency into rollout-worker idle time. **In-process Excel plugins/add-ins** can control Excel, but they couple each rollout trajectory to a persistent Excel session. This is expensive to scale, complicates resource accounting (Excel instances are memory-heavy), and makes failure recovery brittle under high concurrency. **Non-Excel spreadsheet engines (e.g., LibreOffice)** avoid Windows and Excel, but we found them difficult to make reliable at the level of formula and function fidelity required for our benchmarks and real-world spreadsheets (e.g., gaps in function support and subtle behavioral mismatches). Since the goal of Spreadsheet-RL is to train agents against realistic Microsoft Excel semantics, we opt to keep Excel as the source of truth. **Headless formula evaluation in Python** (e.g., `formulas`[1] and `xlcalculator`[2]) avoids Windows and Excel by interpreting formulas directly from `.xlsx`. However, this approach is ill-suited for Spreadsheet-RL: in our experiments it is substantially slower than native Excel recalculation on large dependency graphs (workbooks with many formula cells), and the supported Excel surface remains incomplete. For example, `formulas` reports 483/536 implemented functions (90.1%), leaving gaps in categories such as database, cube, and web functions; `xlcalculator` explicitly does not support array/CSE formulas and documents known deviations for several functions (e.g., `LN`, `VLOOKUP`, and `YEARFRAC`). More broadly, even when a function is implemented, matching Excel's edge-case semantics (type coercion, numerical precision, rounding, and error propagation) is non-trivial; `xlcalculator` explicitly notes that keeping numeric behavior aligned with Excel may require a low-level numeric datatype beyond Python's built-in types. Such semantic mismatches can introduce subtle reward inconsistency relative to the target Excel execution environment. Finally, slower evaluators also amplify end-to-end reward latency. Because RL optimization requires collecting outcome rewards before updating the policy, higher verifier latency backpressures rollout throughput and can reduce GPU utilization unless substantial buffering is introduced.

Overall architecture. We implement the verifier as an *asynchronous job service* around real Excel execution. The API layer (FastAPI) accepts a workbook upload and immediately returns a job identifier; job state is persisted in a shared SQLite job store so multiple API workers can safely serve both submissions and polls. Background workers then claim queued jobs from the store and execute Excel-based recalculation in a bounded worker pool. This architecture separates *short HTTP requests* from *long-running Excel work*, while providing shared state for reliability and operational scaling.

Reward path. For reward computation, rollout workers call `POST /reward/submit` with a sample identifier (the thread directory) and the edited workbook. The service validates metadata (e.g., `answer_position` and the corresponding oracle workbook), enqueues a reward job in SQLite, and returns a job identifier quickly. A background worker claims the job (transitioning `queued` $\rightarrow$ `running`), opens the workbook via Excel COM automation, triggers recalculation, and saves the processed workbook with updated cached values. The evaluator then compares the answer-region cells against the oracle workbook following Section A.11, and writes back both the scalar reward and a structured diagnostic message. The client obtains results by polling `GET /reward/result/{job_id}` until the job reaches `done` or `error`. In our outcome-reward setting, the score is binary (match vs. mismatch), but the diagnostic message is essential for debugging, monitoring, and failure analysis at scale.

Recalculate path. Spreadsheet Gym also exposes a `recalculate_and_read` tool for intermediate self-verification during rollouts. This tool uses the same job machinery but returns the recalculated workbook rather than a reward number: clients submit a workbook to `POST /recalculate/submit`, the service enqueues a `kind=recalculate` job, and workers reuse

[1] `https://github.com/vinci1it2000/formulas`
[2] `https://github.com/bradbase/xlcalculator`

the same bounded Excel pool to open, recalculate, and save the workbook. The client polls `GET /recalculate/result/{job_id}` and receives the resulting `.xlsx` bytes. The rollout worker then reads the requested ranges from the recalculated workbook (using the Excel-cached values) and can optionally write back the updated workbook into the rollout workspace, ensuring that subsequent tool calls and the final episode output remain consistent with faithful Excel semantics.

Scaling and stability. To increase throughput, we maintain a pool of long-lived Excel instances to amortize cold-start overhead. Because Excel processes can accumulate memory or become unstable under sustained load, workers are recycled based on health signals (e.g., recycling when private bytes exceeds a threshold such as 4 GB, or periodically after a fixed number of jobs). Finally, the worker pool is explicitly bounded to prevent overload and to keep the reward signal stable under high-concurrency RL training.

Capacity control and backpressure. The job service implements explicit backpressure to keep the system stable under bursty training loads. Concurrency is bounded by the size of the Excel worker pool, and submissions may be rejected when the pending queue exceeds a configured threshold, allowing rollout workers to retry with backoff rather than overwhelming the verifier. Polling is supported via lightweight status/result endpoints and can optionally use short server-side waits to reduce client polling frequency.

Failure handling and recovery. Each job is executed with a hard timeout and produces an explicit terminal state (`done` or `error`) recorded in SQLite, so transient failures or worker restarts do not corrupt job state. Long-lived Excel instances are proactively recycled to mitigate COM automation instability and memory growth, and stuck jobs can be detected and marked as failed to prevent indefinite queue buildup. These mechanisms are designed to keep reward delivery reliable at scale, rather than optimizing only for the median-case latency.

Semantic fidelity and determinism. Spreadsheet-RL treats Excel as the source of truth for evaluation: spreadsheets often rely on semantics that are difficult to reproduce faithfully outside Excel, including dynamic arrays and spill behavior, legacy array/CSE formulas, volatile functions, locale- and type-coercion corner cases, and numerically sensitive computations. Using real Excel recalculation ensures that both intermediate verification (`recalculate_and_read`) and terminal rewards are aligned with the semantics the agent is trained to operate under.

Data retention and privacy. The verifier operates on uploaded workbooks and oracle artifacts, but it does not require persisting these inputs long-term. Job state is stored as minimal metadata in SQLite, and uploaded/intermediate files can be aggressively cleaned up after completion (or retained briefly only for debugging), reducing both storage overhead and the risk of leaking sensitive spreadsheet contents.

Empirical throughput. In our 4B training runs, a single Windows CPU server with 32 GB memory and a pool of four concurrent Excel instances sustained more than 20,000 reward/recalculation jobs in under 30 minutes (i.e., exceeding 11 reward/recalc jobs per second on average). This throughput is sufficient to keep the verifier from becoming the dominant bottleneck in practice and motivates the engineering emphasis on bounded concurrency, robustness, and short request/long job separation.

A.11 Cell-Level Evaluation Details

We compute outcome rewards by comparing the agent-edited workbook against an oracle workbook on the specified answer regions M. The comparison is performed on cached, post-recalculation cell values (i.e., formula cells are evaluated by their values). Before comparison, cell values are normalized exactly as in SpreadsheetBench:

- Numbers are cast to float and rounded to 2 decimal places.
- Strings are parsed as numbers when possible and then rounded to 2 decimal places.
- `datetime` values are converted to Excel serial dates and rounded to the nearest day.
- `time` values are converted to `HH:MM`.

- Empty strings and `None` are treated as equivalent.

After normalization, a type mismatch counts as a mismatch; otherwise, values must match exactly.

A.12 GRPO OBJECTIVE

Concretely, for each task $[D_i, T]$, we sample a group of N trajectories $\{y_i\}_{i=1}^{N} \sim \pi_{\text{old}}(\cdot \mid D_i, T; G)$ from the old policy through interaction with Spreadsheet Gym G. The current policy π_θ is then updated by maximizing the objective:

$$\mathcal{L}_{\text{GRPO}}(\theta) = \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N} \min\Big(r_i(\theta)\widehat{A}_i, \text{clip}(r_i(\theta), 1-\epsilon, 1+\epsilon)\,\widehat{A}_i\Big)\right] - \beta\, \mathrm{D}_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}})\,. \quad (2)$$

$$r_i(\theta) = \frac{\pi_\theta(y_i \mid D_i, T; G)}{\pi_{\text{old}}(y_i \mid D_i, T; G)}, \quad (3)$$

where $\widehat{A}_i$ is the group-relative advantage of trajectory y_i, computed by normalizing the outcome rewards within the group, π_{ref} is the frozen reference policy, and $\epsilon > 0$ is the clip threshold. The group-relative advantage removes the need for a learned critic while keeping updates stable.

A.13 TRAINING CONFIG AND HYPERPARAMETERS

For each rollout, we use a temperature of 0.6 with top-p = 0.95 and top-k = 20. Each training batch samples 16 rollouts per task over a set of 64 tasks, with a global batch size of 64. At each step, we perform two optimization updates using AdamW with a learning rate of 1×10^{-6}. We apply KL divergence with coefficient 0.001 and enable dynamic batching for training efficiency. All training experiments are run on 1×4 NVIDIA H100 GPUs; a full 4B training run takes approximately 40 hours of wall-clock time.

A.13.1 TRAINING DYNAMICS OF THE 4B RUN

Fig. 4 in the main text plots the 4B run. The smoothed training reward rises from roughly 0.21 in the first steps to 0.33 by step 60, while SpreadsheetBench accuracy improves from 19.3 at step 0 to 23.4 at step 60. Interaction cost also becomes more controlled: mean response length drops from approximately 16k near the start to approximately 11k by step 60, and mean interaction turns fall from roughly 20 to about 11.

Table 9: Training hyperparameters for the Qwen3-4B-Thinking-2507 run.

Hyperparameter	Value
Base model	`Qwen/Qwen3-4B-Thinking-2507`
Algorithm	GRPO; KL-regularized against a frozen reference model
Training steps	60; SpreadsheetBench evaluated every 10 steps
Batching	64 tasks/step; 16 rollouts/task ($N = 16$); 1,024 rollouts/step
Actor update batching	mini-batch 32; dynamic batch sizing enabled
Optimizer	AdamW; learning rate 1×10^{-6}; weight decay 0.01; $(\beta_1, \beta_2) = (0.9, 0.999)$; grad clip 1.0
KL loss	low-var KL; coefficient 0.001
Prompt/response limits	4,096 / 27,648 tokens
Multi-turn caps	max assistant turns 20; max user turns 20; max tool-response length 8,192
Interaction protocol	inspect-modify-verify spreadsheet harness instruction
Sampling (rollouts)	temperature 0.6; top-p 0.95; top-k 20
Sampling (evaluation)	deterministic decoding; temperature 0
Compute	1 node × 4 NVIDIA H100 GPUs; about 40 hours of wall-clock time

Table 10: Training hyperparameters for the Qwen3-8B run; unlisted settings follow the 4B run.

Hyperparameter	Value
Base model	`Qwen/Qwen3-8B`
Algorithm	GRPO; KL-regularized against a frozen reference model; clip ratio 0.2
Training steps	35 (best checkpoint); SpreadsheetBench evaluated every 7 steps
Batching	128 tasks/step; 8 rollouts/task ($N = 8$); 1,024 rollouts/step
Actor update batching	mini-batch 64
Optimizer	AdamW; learning rate 1×10^{-6}; weight decay 0.01
KL loss	coefficient 0.001
Prompt/response limits	4,096 / 27,648 tokens
Multi-turn caps	as in the 4B run
Interaction protocol	inspect-modify-verify spreadsheet harness instruction
Sampling (rollouts)	temperature 0.6; top-p 0.95; top-k 20
Sampling (evaluation)	deterministic decoding; temperature 0
Compute	2 nodes $\times$ 4 GPUs

A.13.2 QWEN3-8B SETTINGS

The Qwen3-8B run (Section A.14) uses the same training data, environment, harness, and tool interface, with the scale-adjusted optimization settings in Table 10: 8 rollouts per task over 128 tasks per step, i.e., the same 1,024 rollouts per step as the 4B run.

A.14 SPREADSHEET-RL AT A LARGER MODEL SIZE: QWEN3-8B

To examine whether Spreadsheet-RL remains effective at a larger model size, we train Qwen3-8B with the same pipeline. We use the same training data, Spreadsheet Gym environment, spreadsheet-native harness, comprehensive tool interface, evaluation datasets, and overall GRPO framework, while adapting optimization settings for the larger model (Table 10).

Spreadsheet-RL improves Qwen3-8B from 16.8 to 22.3 on SpreadsheetBench (Table 1) and from 5.0 to 11.8 on Domain-Spreadsheet (Table 2). In the Qwen3-8B block of Table 1, the 15.9 entry is the model without our harness and comprehensive tools, whereas the 16.8 entry uses the full harness and tool interface from which RL starts. Spreadsheet-RL therefore remains effective at the larger 8B size.

Why the 4B agent ends higher. Although the raw Qwen3-8B model starts above the raw Qwen3-4B-Thinking-2507 model (15.9 vs. 12.0 on SpreadsheetBench), the 4B agent gains more from the harness and tools (19.3 vs. 16.8) and from RL (23.4 vs. 22.3), and the gap is wider on Domain-Spreadsheet (17.2 vs. 11.8). Qwen3-4B-Thinking-2507 belongs to the later 2507 release of the Qwen3 series, with an updated post-training recipe and a dedicated thinking mode, whereas Qwen3-8B belongs to the original Qwen3 release. We attribute the difference to this stronger reasoning and tool-use post-training rather than to parameter count: the harness and RL both reward long-horizon inspect–modify–verify behavior, which the thinking checkpoint executes more reliably. For the same reason, this experiment should not be interpreted as a controlled parameter-scaling study; the two checkpoints differ in release vintage, pretraining, post-training procedure, and thinking configuration, so we focus on the within-model improvement produced by Spreadsheet-RL rather than on absolute performance as a function of scale.

A.14.1 QWEN3-8B TRAINING DYNAMICS

Performance improves progressively during RL training, from 16.79 at initialization to 22.34 at step 35. The intermediate checkpoints are reported in Table 11. Although the curve is not strictly monotonic at every evaluation point, it exhibits a clear upward trend throughout training. This behavior is consistent with the Qwen3-4B-Thinking-2507 experiment and provides additional evidence that outcome-based spreadsheet RL can improve spreadsheet-agent performance beyond a single base checkpoint.

Training Step	SpreadsheetBench Pass@1
0	16.79
7	17.78
14	19.29
21	19.43
28	20.68
35	**22.34**

Table 11: **SpreadsheetBench performance of Qwen3-8B throughout Spreadsheet-RL training.** Performance improves from 16.79 before RL to 22.34 at training step 35.

A.14.2 Domain-Level Results

Table 2 breaks down Qwen3-8B performance across Domain-Spreadsheet categories. The largest gains occur in finance and supply-chain tasks, while smaller improvements are observed in human resources and sales. Real-estate performance remains unchanged. The broad distribution of improvements suggests that the overall gain is not driven by a single task category. At the same time, low absolute accuracy in several domains indicates that improved tool interaction alone is insufficient for workflows requiring specialized domain knowledge or more complex semantic reasoning.

A.15 Separating RL from Harness and Tool Design

The staged results in Table 1 show that both spreadsheet-native interaction design and RL contribute to final performance. To isolate the effect of RL from the proposed harness and structured tool interface, we additionally run RL from the raw Qwen3-4B-Thinking-2507 configuration, i.e., with a generic instruction and only the minimal tools (`code_interpreter` and `recalculate_and_read`), using the same training data and GRPO settings. Starting from 12.0, the complete harness and tool interface reach 19.3 without RL, RL alone reaches 20.1, and their combination reaches 23.4. The performance gain therefore cannot be attributed solely to harness engineering: RL independently provides a substantial improvement, while the harness and tool interface also contribute significantly before post-training. The two improvements are not additive: the harness and tools alone contribute a 7.3-point gain and RL alone an 8.1-point gain, whereas their combination improves the raw model by 11.4 points, which suggests that the two components address partially overlapping failure modes while remaining complementary.

A.16 Controlled Evaluation of OpenAI o3

Comparisons with proprietary spreadsheet agents can be confounded by differences in execution environments and available tools. We therefore additionally evaluate OpenAI o3 using the same Spreadsheet-RL harness, comprehensive spreadsheet tools, and Spreadsheet Gym environment. Under this controlled configuration, o3 achieves 52.2 Pass@1 on SpreadsheetBench (Table 1); the evaluation requires 4:37:11 of wall-clock time and approximately $165 in API cost. A frontier proprietary model thus remains substantially stronger than the 4B Spreadsheet-RL agent under matched interaction conditions; the comparison with reported closed-source numbers in Section 5.2 therefore shows what a small open model achieves relative to frontier agents in generic setups, not superiority under matched conditions.

A.17 Detailed Trajectory Analysis

To better understand what the learned policy changes, we compute deterministic statistics directly from complete SpreadsheetBench trajectories before and after RL. These measurements use only model-generated responses, tool calls, and environment feedback and therefore do not require an LLM judge.

We define an invalid tool call as either an unknown tool invocation or one whose arguments violate the corresponding schema. A tool-execution error is an explicit error returned by the environment. A successful write is a completed spreadsheet-modifying action or save-producing code execution. Post-edit verification occurs when the agent successfully reads, inspects, or recalculates the workbook

Metric	Pre-RL	Post-RL	Change
Task success and efficiency			
Task success	19.3	23.4	+4.1
Executed tool calls / task	3.94	4.60	+0.66
Model-response tokens / task	10,663	8,753	−17.9
Execution reliability			
Invalid tool calls	1.98	0.16	−1.82
Tool-execution error calls	48.92	7.35	−41.57
Successful write calls	71.39	89.47	+18.08
Tasks completing a successful write	85.96	97.99	+12.03
Verification and recovery			
Post-edit verification among edited tasks	6.71	21.48	+14.78
Error recovery among tasks with an error	47.62	95.08	+47.46
Tasks with malformed protocol	22.06	5.26	−16.79
Failed tasks with no successful write	16.77	2.80	−13.97
Tasks with duplicate calls	17.29	12.28	−5.01

Table 12: **Complete deterministic trajectory analysis before and after RL on SpreadsheetBench.** Rate-based entries and corresponding changes are reported on a 0–100 scale. RL improves execution reliability, successful workbook modification, error recovery, and post-edit verification while reducing model-response length.

after its first successful modification. Error recovery occurs when the agent successfully executes a subsequent tool action after encountering an execution error.

The complete statistics are reported in Table 12. RL improves SpreadsheetBench task success from 19.3 to 23.4 while reducing mean model-response length from 10,663 to 8,753 tokens. At the same time, the average number of executed tool calls increases slightly from 3.94 to 4.60. Thus, the improvement is not explained by simply producing longer interaction trajectories.

The strongest changes are observed in execution reliability. Tool-execution error calls decrease from 48.92 to 7.35, while successful write calls increase from 71.39 to 89.47. The fraction of tasks completing at least one successful write increases from 85.96 to 97.99.

RL also substantially changes recovery and verification behavior. Among tasks encountering an execution error, successful recovery increases from 47.62 to 95.08. Post-edit verification increases from 6.71 to 21.48, while malformed-protocol behavior decreases from 22.06 to 5.26.

Finally, mean model-response length decreases by 1,910 tokens per task. The 95% confidence interval for this reduction is 1,321–2,497 tokens. Taken together, these results indicate that RL improves both the reliability and efficiency of spreadsheet interaction rather than merely increasing inference-time computation.

A.18 Failure Analysis

The deterministic diagnostics above characterize observable interaction behavior but do not describe the semantic causes of unsuccessful spreadsheet trajectories. We therefore conduct a complementary structured failure analysis with an LLM judge (GPT-5.6) over all 2,726 SpreadsheetBench trajectories of the pre-RL and post-RL 4B agents.

For each trajectory, the judge receives the user instruction, a structured overview of the initial workbook, the designated answer position, deterministic trajectory-level signals, and the full sequence of agent actions and tool feedback. The oracle final workbook is intentionally excluded because the goal is to categorize observable failure patterns rather than independently re-evaluate task success. During taxonomy development, we manually inspect representative trajectories and iteratively refine the failure categories. The final taxonomy distinguishes semantic spreadsheet errors, localization failures, incomplete multi-step reasoning, verification failures, ineffective edits, execution failures,

Failure Category	Pre-RL	Post-RL
Unsolved trajectories (reference)	80.7	76.6
Semantic spreadsheet reasoning failures		
Wrong formula, value, or output format	63.3	55.1
Wrong sheet, range, or edit scope	14.0	12.0
Incomplete multi-step reasoning	4.5	3.9
Interaction and verification failures		
Verification failure	6.6	4.2
No effective edit	3.3	3.1
Unrecovered tool-execution error	6.6	3.1
Malformed tool call or protocol	12.9	2.4
Looping or budget exhaustion	1.2	0.7
Hallucinated tool	0.0	0.0

Table 13: **LLM-judge failure categorization of all 2,726 SpreadsheetBench trajectories before and after RL.** Each entry is the percentage of *all* evaluated trajectories, not only unsolved ones, in which the category is assigned as the primary failure or as a contributing factor; categories therefore co-occur and need not sum to 100. RL substantially reduces low-level interaction failures, while incorrect formulas, values, and output formats remain the dominant residual failure category.

malformed interaction protocols, and looping behavior. The judge assigns one primary failure mode and any number of contributing factors.

Table 13 reports, for each category, the percentage of all 2,726 trajectories in which the category is assigned as the primary failure or as a contributing factor. Because the denominator is the full evaluation set rather than only unsolved trajectories, and because categories co-occur, the entries do not sum to 100; the unsolved rates of 80.7% before RL and 76.6% after RL are given as a reference.

The largest improvement after RL occurs in low-level interaction failures. Malformed tool-call or protocol behavior is flagged in 12.9% of trajectories before RL and 2.4% after, while unrecovered tool-execution errors decrease from 6.6% to 3.1%. Verification failures and incorrect worksheet or range localization also become less frequent.

In contrast, incorrect formulas, values, or output formats remain the most frequent category after RL, flagged in 63.3% of trajectories before RL and 55.1% after. Relative to the unsolved trajectories, this category is involved in roughly 78% of failures before RL and 72% after, so the residual failures are increasingly semantic. This pattern suggests that RL is particularly effective at improving interaction reliability and execution behavior, while semantic spreadsheet reasoning remains the primary bottleneck for further progress.

This analysis is model-based: the judge does not see the final or oracle workbook, and its labels were validated only by manual inspection during taxonomy development rather than by a systematic human study. We therefore treat it as a complement to the deterministic trajectory statistics, with which it is consistent: once malformed interactions, execution failures, and unsuccessful edits become less frequent, a larger fraction of the remaining failures arise from selecting the wrong spreadsheet operation, constructing an incorrect formula, or otherwise misunderstanding the task semantics.

A.19 Qualitative Case Studies: Pre-RL Tool Interface Failures

We examined failed evaluation rollouts from the minimal tool setting to understand common failure modes when `code_interpreter` is responsible for workbook edits and `recalculate_and_read` is used only for recalc/readback. These examples do not imply that Python cannot express the desired operations; rather, they show how model-generated code can mis-handle low-level spreadsheet semantics that structured tools make explicit. Two representative failures motivate the structured edit tools in Section 3.2.3.

Minimal-Interface Failure Modes in Evaluation Rollouts

Case 1: Header search with structural deletion. A task asks the agent to delete every column whose row-1 header contains /description. In a minimal-interface rollout, the generated code resembles for col in range(1, sheet.max_column + 1): ... sheet.delete_cols(col), deleting immediately after each match. Because deleting a column shifts all subsequent columns left, later indices no longer refer to the originally inspected columns, causing the rollout to skip or delete the wrong columns. The delete_columns tool instead exposes structural deletion as a single spreadsheet operation after the target ranges are identified.

Case 2: Formula propagation by string templating. A task asks the agent to modify a VLOOKUP formula in G3:G58. In a minimal-interface rollout, the model hand-constructs formula strings inside a Python loop, including an IFERROR(VLOOKUP(...), "") branch with embedded quotation marks, and fails to escape the string correctly while also taking responsibility for row-specific references. The fill_formula tool shifts this burden to the harness: the model provides the target range and the top-left formula template, and the tool translates relative references and verifies recalculation before overwriting the workbook.

A.20 Qualitative Case Studies: Post-RL Rollout Behavior

To better understand how Spreadsheet-RL changes rollout behavior beyond final-task accuracy, we compare two checkpoints of the same 4B agent: an early checkpoint at step 0 and a later checkpoint at step 50. Both corresponding rollout logs contain 2,726 SpreadsheetBench rollouts, but the step 50 checkpoint is more concise (mean assistant-output length 38,965 vs. 51,732 characters). Using simple string matching on assistant outputs, the step 50 checkpoint more often states an explicit `Alternative plan` when backtracking (4.3% vs. 1.2%). Conversely, the step 0 checkpoint more frequently relies on speculative validation language (e.g., "should work", 60.6% vs. 55.1%) and more often admits being stuck ("I'm really stuck", 21.3% vs. 0.8%). The excerpts below illustrate these qualitative differences.

Rollout Excerpts (Step 50 vs. Step 0)

Case 1: Alternative plan vs. speculative "should work" (Task 1_536-37).
Step 50: "Alternative plan: Since the user has the 'NO Listing' in their row 4, we can find the exact cell with 'NO Listing' and use that."
Step 0: "The code should work."

Case 2: Stable debugging vs. crash-and-recover (Task 1_178-22).
Step 50: "So, it's possible that there are no rows matching the criteria."
Step 0: "The previous attempt to create 'Sheet2' failed with a KeyError on index 1."

Case 3: Instruction scaffold and protocol compliance (Task 1_39180).
Step 50: "Plan: Check the structure of the Excel file, confirm that column C has data starting from row 3 to 57."
Step 0: "Okay, the user is saying there's no new tool execution since the last submission, so I need to make changes with code_interpreter."

A.21 Why SheetAgent Is Not Included as a Direct Baseline

SheetAgent (Chen et al., 2025) is an important prior spreadsheet-agent design, but we do not include a direct SheetAgent rerun as a quantitative baseline in Table 1. The system is tightly coupled to

a particular model backend, prompt stack, code-retrieval component, and spreadsheet execution setting: its Planner, Informer, and Retriever interact through model-generated code and retrieval from a configured code corpus, so changes in the LLM API, spreadsheet runtime, or dataset interface can substantially affect the final workbook edits. In our reproduction attempt, the public release did not provide a complete turnkey evaluation path for all model backends and spreadsheet environments considered in our study; for example, the Retriever depends on an external Milvus setup, and the released data/evaluation artifacts cover only a subset of the original SheetRM setting. When adapted directly to our SpreadsheetBench evaluation protocol and spreadsheet environment, the system frequently failed before producing valid spreadsheet edits or produced outputs incompatible with our verifier, yielding near-zero Pass@1 in pilot runs. We therefore exclude this rerun from quantitative baselines, since it would measure an incomplete cross-environment reproduction rather than the SheetAgent method itself. Instead, we cite SheetAgent as prior work and compare against baselines whose SpreadsheetBench numbers are reported by prior work or whose runs are reproducible in our Spreadsheet Gym.

A.22 Limitations

Spreadsheet-RL provides an open research foundation for studying RL post-training in spreadsheet-based data workflows, but the present study has several limitations. First, due to compute constraints, our RL experiments cover two lightweight open-source models (Qwen3-4B-Thinking-2507 and Qwen3-8B); we do not train larger dense or mixture-of-experts models, and the 8B experiment shows that the gains hold at a larger size but is not a controlled scaling study, since the two checkpoints differ in release vintage and post-training (Section A.14). Absolute accuracy remains low, and a frontier proprietary model with the same harness is substantially stronger (Section A.16). Second, both the RL rewards and the Domain-Spreadsheet targets are automatically constructed oracles. The human audit finds majority-valid oracles for 84% of sampled training tasks and 98% of sampled Domain-Spreadsheet tasks (Section A.3), so the training supervision is not noise-free, and the audit starts from the extracted instruction rather than from the raw forum discussion. Third, exact all-cell matching is a reproducible but narrow correctness criterion; it can undercount semantically equivalent solutions with different formulas, layouts, or formatting, and our audit measures its task-level suitability rather than its false-positive and false-negative rates on model outputs (Section A.3.3). Fourth, Domain-Spreadsheet is author-constructed with the same oracle-construction pipeline as the training data; although its task composition differs substantially from the training data (Section A.5), an externally sourced professional benchmark would provide a stronger test of generalization. Finally, the ambiguity, composition-shift, and failure analyses rely on LLM judges and were validated only by manual spot checks. We leave scaling Spreadsheet-RL to larger model families, richer rubric-based evaluation, and externally sourced professional benchmarks as directions for future work, and hope that the release of our data, environment, harness, and training pipeline will make such exploration more accessible to the community.