

LLM-in-Sandbox Elicits General Agentic Intelligence

Daixuan Cheng^{αβ} Shaohan Huang^β Yuxian Gu^γ Huatong Song^α Guoxin Chen^α
Li Dong^β Wayne Xin Zhao^{α†} Ji-Rong Wen^α Furu Wei^{β†}

^αGSAI, Renmin University of China ^βMicrosoft Research ^γTsinghua University
<https://llm-in-sandbox.github.io>

Abstract

We introduce **LLM-in-Sandbox**, enabling LLMs to explore within a code sandbox (i.e., a virtual computer), to elicit general intelligence in non-code domains. We first demonstrate that strong LLMs, without additional training, exhibit generalization capabilities to leverage the code sandbox for non-code tasks. For example, LLMs spontaneously access external resources to acquire new knowledge, leverage the file system to handle long contexts, and execute scripts to satisfy formatting requirements. We further show that these agentic capabilities can be enhanced through LLM-in-Sandbox Reinforcement Learning (LLM-in-Sandbox-RL), which uses only non-agentic data to train models for sandbox exploration. Experiments demonstrate that LLM-in-Sandbox, in both training-free and post-trained settings, achieves robust generalization spanning mathematics, physics, chemistry, biomedicine, long-context understanding, and instruction following. Finally, we analyze LLM-in-Sandbox’s efficiency from computational and system perspectives, and open-source it as a Python package to facilitate real-world deployment.

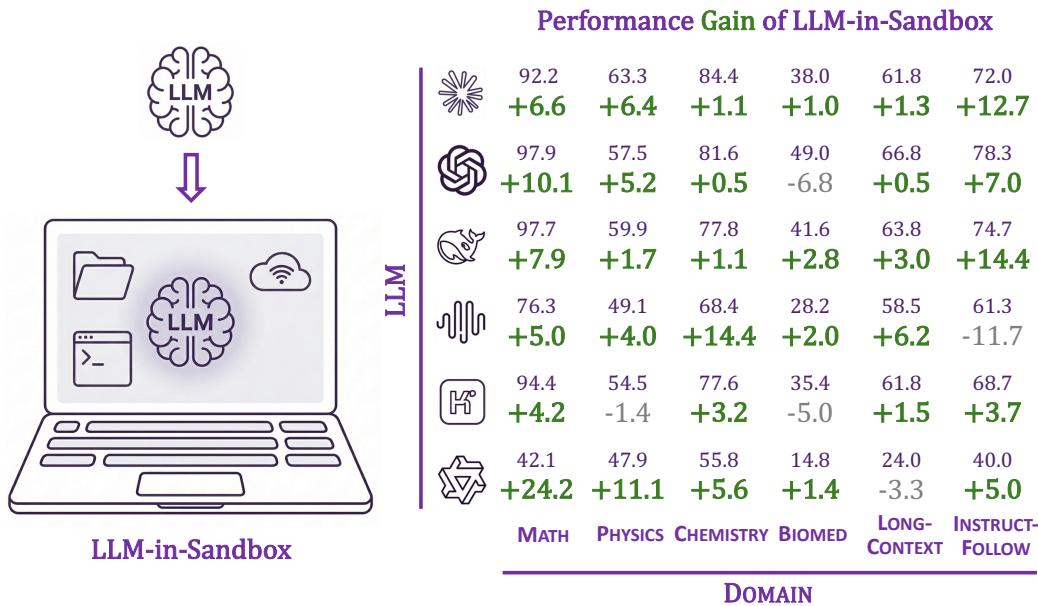


Figure 1: Overview of LLM-in-Sandbox. We enable LLMs to explore within a code sandbox (i.e., virtual computer), unlocking significant performance gains across diverse LLMs and domains. Green values indicate improvements over vanilla LLMs. **All LLMs are evaluated without additional training.**

Email: daixuancheng6@gmail.com [†]Corresponding Authors.

1 Introduction

The capabilities of Large Language Models (LLMs) have been progressively unlocked through different paradigms. In-context learning showed that models could generalize to new tasks without task-specific finetuning (Brown et al., 2020). Chain-of-thought prompting then elicited reasoning by guiding models to decompose problems into steps (Wei et al., 2022). Recently, agentic frameworks empowered models to leverage diverse tools across multiple turns (Anthropic, 2025b). Following this trajectory, how can we further unlock their capabilities?

In this work, we propose **LLM-in-Sandbox**—enabling LLMs to explore within a code sandbox—as a promising next step along this trajectory. As shown in Figure 1, the sandbox is essentially a virtual computer with terminal capabilities, widely used by code agents such as Claude Code (Anthropic, 2025a). While it is typically used for software engineering (Jimenez et al., 2023), we argue its potential extends far beyond coding. Computers are perhaps the most versatile platform ever created—virtually any task can be accomplished through them. This versatility stems from three meta-capabilities: *external resource access* (e.g., the internet), *file management*, and *code execution*. We hypothesize that combining LLMs with a virtual computer may unlock their potential for general intelligence.

To validate this potential, we evaluate LLM-in-Sandbox on challenging non-code tasks. Given a task input, LLMs explore within a sandbox with basic computer capabilities across multiple turns until task completion. Remarkably, without any additional training, LLMs can spontaneously leverage the code sandbox for non-code tasks, such as installing domain-specific tools to gain new abilities, utilizing file storage to process documents beyond context limits, and executing scripts to meet formatting requirements. As a result, state-of-the-art agentic LLMs achieve substantial performance gains across mathematics, physics, chemistry, biomedicine, long-context understanding, and instruction-following.

To further advance this paradigm, we propose **LLM-in-Sandbox Reinforcement Learning (LLM-in-Sandbox-RL)**. While strong agentic models benefit directly from LLM-in-Sandbox, weaker models often struggle: performing worse in LLM-in-Sandbox mode than in vanilla LLM mode (i.e., directly generating the output without sandbox). LLM-in-Sandbox-RL bridges this gap using only general, non-agentic data. Specifically, we use general context-based tasks (Cheng et al., 2024) where the context is pre-placed as text files in the sandbox rather than directly in the model prompt, requiring the model to explore and interact with the environment. With only outcome-based rewards (Guo et al., 2025), LLM-in-Sandbox-RL enables weaker models to excel in LLM-in-Sandbox mode, significantly outperforming their LLM mode, while also enhancing models that already possess strong agentic capabilities. Crucially, this training elicits strong generalization: leading to consistent improvements across diverse out-of-domain tasks, and even enhancing vanilla LLM mode. This suggests LLM-in-Sandbox-RL can be a general method to elicit both agentic and non-agentic intelligence across models and domains.

Beyond performance, we analyze practical considerations of deploying LLM-in-Sandbox in real-world systems, covering computational cost, speed, and sandbox infrastructure. We find that LLM-in-Sandbox dramatically reduces token consumption by up to $8\times$ in long-context scenarios ($100K \rightarrow 13K$ tokens), achieves competitive query-level throughput on average, and incurs minimal sandbox infrastructure overhead. Finally, we open-source LLM-in-Sandbox as a Python package that integrates seamlessly with popular inference backends (e.g., vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2024)) and API-based LLMs, to accelerate the transition towards general agentic intelligence.

Our contributions are summarized as follows:

- We introduce LLM-in-Sandbox, demonstrating that strong agentic LLMs exhibit generalization capabilities to exploit code sandboxes for non-code tasks across diverse domains, without additional training (Section 2).
- We propose LLM-in-Sandbox Reinforcement Learning, which trains LLMs to explore sandbox environments using only general non-agentic data, enhancing generalization of both agentic and non-agentic intelligence across domains (Section 3).

-
- We analyze LLM-in-Sandbox’s efficiency from computational and system perspectives, and open-source it as a Python package for real-world deployment (Section 4).

2 LLM-in-Sandbox Elicits General Intelligence

The core idea of LLM-in-Sandbox is to grant LLMs access to a computer where they can freely operate to complete user-specified tasks. Specifically, computers possess three meta-capabilities that form the foundation for general task-solving:

- *External resource access*: fetching resources from external services (e.g., the internet);
- *File management*: reading, writing, and organizing data persistently;
- *Code execution*: writing and executing arbitrary programs.

Just as humans leverage computers to accomplish virtually any task, we hypothesize that combining LLMs’ powerful reasoning and agentic capabilities with a code sandbox may unlock their potential for general intelligence.

To explore the full potential of this paradigm, our design of LLM-in-Sandbox emphasizes two principles: **minimal**—providing a basic code sandbox with these three capabilities, and **exploratory**—encouraging models to discover diverse solution strategies. In the following, we describe our sandbox environment (Section 2.1), the LLM-in-Sandbox workflow (Section 2.2), as well as experiments (Section 2.3) and analysis (Section 2.4) in general domains.

2.1 Code Sandbox

A code sandbox is a virtualized computing environment, typically an Ubuntu-based system implemented via Docker containers, that provides LLMs with terminal access and full system capabilities. Within this environment, LLMs can execute arbitrary bash commands, create and modify files, and access network resources. The containerized nature ensures isolation from the host system, enabling safe execution of model-generated code.

	SWE Agents	LLM-in-Sandbox
Environment Setup	Task-specific	General-purpose
Dependencies	Pre-configured	Runtime installation
Storage Scaling	Per-task images	Single shared image

Table 1: Comparison of sandbox design between SWE agents and LLM-in-Sandbox.

Lightweight General-Purpose Design. Code sandboxes have recently emerged as a critical infrastructure for code agents like Claude Code (Anthropic, 2025a). However, existing sandbox-based systems, especially those for software engineering tasks (Jain et al., 2025; Wang et al., 2024; Yang et al., 2024), require complex, task-specific environments. Instead, we provide a lightweight and general-purpose environment equipped only with a standard Python interpreter and essential scientific computing libraries (e.g., NumPy, SciPy), and delegate domain-specific tool acquisition to the model itself. During execution, models can install or create any tools they deem necessary. Table 1 summarizes the key differences. This design offers two advantages: (1) *Generalizability*: the same environment supports diverse tasks without manual reconfiguration, and (2) *Scalability*: the uniform setup enables efficient large-scale inference and training without per-task overhead. For example, when scaling to thousands of tasks, SWE agents may require up to 6 TB of storage for task-specific images (Pan et al., 2024), whereas our shared image approach maintains a constant footprint of only \sim 1.1 GB.

Minimal Toolset with Meta-Capabilities. Within the code sandbox, we equip the model with three fundamental tools that together realize the core capabilities of a computer: (1) `execute_bash` for executing arbitrary terminal commands—the most fundamental yet versatile interface that enables virtually any computer operation, including but not limited

Algorithm 1 LLM-in-Sandbox Workflow

Require: Task prompt p , Task requirements r (optional), Sandbox \mathcal{S} , Maximum turns T
Ensure: Final output o

```
1: Configure sandbox  $\mathcal{S}$  with task requirements  $r$  (if any)
2:  $t \leftarrow 0$ 
3: Tools: {execute_bash, str_replace_editor, submit}
4: while  $t < T$  do
5:   Model generates tool call  $a_t$  based on prompt  $p$  and history
6:   if  $a_t$  is submit then
7:     break
8:   end if
9:   Execute  $a_t$  in  $\mathcal{S}$ , obtain observation  $obs_t$ 
10:  Append  $(a_t, obs_t)$  to interaction history
11:   $t \leftarrow t + 1$ 
12: end while
13: Extract output  $o$  from sandbox  $\mathcal{S}$  (e.g., /testbed/answer.txt)
14: return  $o$ 
```

to installing packages, managing files, and running programs; (2) `str_replace_editor` for file creation, viewing, and editing; and (3) `submit` for indicating task completion. Detailed specifications are provided in Appendix A.

2.2 LLM-in-Sandbox Workflow

Our workflow builds on the ReAct framework (Yao et al., 2022), where the model iteratively reasons and acts based on environmental feedback. As shown in Algorithm 1 (purple highlights indicate sandbox-specific components), at each turn, the model generates a tool call, receives the execution result from the sandbox, and decides the next action. This multi-turn interaction continues until the model calls `submit` or reaches a maximum turn limit. To accommodate diverse scenarios in general tasks, our workflow encourages free exploration and supports flexible input/output handling.

Prompting for Exploration. We design a system prompt that guides models to fully utilize the sandbox. First, it encourages models to leverage computational tools rather than performing calculations through natural language. Second, it emphasizes deriving answers through program execution instead of directly hardcoding results. Third, it informs models that the sandbox is a safe, isolated environment where they can freely explore diverse approaches to complete tasks. The full system prompt is provided in Appendix F.

Task Input/Output Handling. We leverage the sandbox’s file system to flexibly handle diverse input/output formats. For inputs, content can be provided not only via the model prompt but also through files. For example, for long-context understanding tasks that require reading documents, we can regard the documents as task requirements and place documents in `/testbed/documents/`. For outputs, the model is instructed to place the final result at a designated location (e.g., `/testbed/answer.txt`), containing only the final result without intermediate content. After task completion, the result is extracted from this location as the final output. This approach cleanly separates exploration from final output and naturally accommodates various data formats.

2.3 Experiments on General Domains

We conduct experiments to investigate whether sandbox access improves LLM performance on general tasks. Below we present the experimental setup and results.

Setup We compare LLM-in-Sandbox with vanilla LLM generation (i.e., directly generating the output without sandbox) across diverse models and domains. The evaluated LLMs

Model	Mathematics			Physics			Chemistry		
	LLM	LLM	Δ	LLM	LLM	Δ	LLM	LLM	Δ
Claude-Sonnet-4.5-Think	85.6	92.2	+6.6	56.9	63.3	+6.4	83.3	84.4	+1.1
GPT-5	87.8	97.9	+10.1	52.3	57.5	+5.2	81.1	81.6	+0.5
DeepSeek-V3.2-Thinking	89.8	97.7	+7.9	58.2	59.9	+1.7	76.7	77.8	+1.1
MiniMax-M2	71.3	76.3	+5.0	45.1	49.1	+4.0	54.0	68.4	+14.4
Kimi-K2-Thinking	90.2	94.4	+4.2	55.9	54.5	-1.4	74.4	77.6	+3.2
Qwen3-Coder-30B-A3B	17.9	42.1	+24.2	36.8	47.9	+11.1	50.2	55.8	+5.6
Qwen3-4B-Instruct-2507	41.3	35.4	-5.9	40.5	36.3	-4.2	56.2	50.7	-5.5

Model	Biomedicine			Long-Context			Instruct. Follow.		
	LLM	LLM	Δ	LLM	LLM	Δ	LLM	LLM	Δ
Claude-Sonnet-4.5-Think	37.0	38.0	+1.0	60.5	61.8	+1.3	59.3	72.0	+12.7
GPT-5	55.8	49.0	-6.8	66.3	66.8	+0.5	71.3	78.3	+7.0
DeepSeek-V3.2-Thinking	38.8	41.6	+2.8	60.8	63.8	+3.0	60.3	74.7	+14.4
MiniMax-M2	26.2	28.2	+2.0	52.3	58.5	+6.2	73.0	61.3	-11.7
Kimi-K2-Thinking	40.4	35.4	-5.0	60.3	61.8	+1.5	65.0	68.7	+3.7
Qwen3-Coder-30B-A3B	13.4	14.8	+1.4	27.3	24.0	-3.3	35.0	40.0	+5.0
Qwen3-4B-Instruct-2507	10.4	10.6	+0.2	30.8	5.8	-25.0	32.7	28.7	-4.0

Table 2: Task performance of models under LLM and LLM-in-Sandbox generation modes across domains. LLM denotes LLM-in-Sandbox mode. $\Delta = \text{LLM-in-Sandbox} - \text{LLM}$ denotes the performance difference of LLM-in-Sandbox relative to LLM.

cover frontier proprietary, open-weight, code-specialized, and smaller general-purpose models: Claude-Sonnet-4.5-Thinking ([Anthropic, 2025b](#)), GPT-5 ([Singh et al., 2025](#)), DeepSeek-V3.2-Thinking ([Liu et al., 2025](#)), MiniMax-M2 ([MiniMax, 2025](#)), Kimi-K2-Thinking ([Team et al., 2025](#)), Qwen3-Coder-30B-A3B-Instruct ([Yang et al., 2025a](#)), and Qwen3-4B-Instruct-2507 ([Yang et al., 2025a](#)).

We test on challenging tasks in six non-code domains: Mathematics, Physics, Chemistry, Biomedicine, Long-Context Understanding, and Instruction Following. For long-context tasks, we store the input documents in the sandbox environment rather than including them in the prompt, to test the model’s ability to leverage the sandbox. Since models have internet access in the sandbox, we reframe test problems to prevent benchmark hacking and manually verify sampled trajectories to ensure valid reasoning. The detailed sandbox implementations, model configurations, and evaluation protocols are in Appendices A-C.

Results As shown in Table 2, strong agentic models consistently benefit from LLM-in-Sandbox, with improvements observed across all evaluated domains—from computation-intensive tasks (Mathematics) to knowledge-intensive tasks (Chemistry, Biomedicine) to general capabilities (Instruction Following, Long-Context). The largest gains reach +24.2% (Qwen3-Coder on Mathematics). However, weaker models like Qwen3-4B-Instruct fail to benefit and even perform worse. We analyze the reasons in the following sections.

2.4 Analysis on Sandbox Utilization

To understand how models leverage the sandbox environment, we conduct a case study and quantitative analysis. Specifically, we focus on the three core capabilities of sandbox: *external resource access*, *file management*, and *code execution*. Code execution can serve many purposes; here we specifically track *computation-oriented* operations. We identify these behaviors through pattern matching on model actions: (1) *external resources*—network requests (e.g., `curl`, `requests.get`) and package installation (e.g., `pip install`); (2) *file management*—file I/O operations (e.g., `open()`, `json.load`) and shell commands (e.g., `cat`, `grep`); (3) *computation*—numerical solvers, iterative algorithms, and simulation loops. Detailed classification patterns are provided in Appendix D.

2.4.1 Case Study

We conduct a case study to illustrate how strong agentic models utilize the sandbox’s core capabilities to solve non-code tasks, where strong models refer to those that perform better in LLM-in-Sandbox mode than in LLM mode in Table 2. Below, we present representative trajectory snippets and *Gray text indicates our commentary*.

External Resources Access. In the chemistry task, the model is asked to predict molecular properties given only compound names. To solve this, the model autonomously installs Java runtime via apt-get and downloads the OPSIN library to convert chemical names to molecular structures—acquiring domain-specific tools unavailable in the base environment.

Chemistry Trajectory (External Resources Access)

```
[Turn 1] Action: execute_bash 'pip install rdkit-pypi -q'  
// RDKit installed but fails to import due to NumPy version conflicts  
  
[Turn 8] Action: execute_bash 'apt-get install -y default-jre'  
[Turn 8] Observation: Setting up openjdk-21-jre-headless ... done  
  
[Turn 12] Action: execute_bash 'curl -s \  
https://api.github.com/repos/dan2097/opsin/releases/latest | grep tag_name'  
[Turn 12] Observation: "tag_name": "2.8.0"  
  
[Turn 14] Action: execute_bash 'apt-get install -y libopsin-java'  
[Turn 14] Observation: Setting up libopsin-java (2.8.0-3) ... done  
  
[Turn 18] Action: execute_bash 'java -jar opsin-cli.jar -o smi < input.txt'  
[Turn 18] Observation: NCC(CCCCN1/C(/C(C2=CC=CC=C12) ... //molecular structure
```

File Management. In the long-context task, the model is given lengthy industry reports (exceeding 100K tokens) and asked to extract specific information. Rather than processing entire documents in memory, the model uses shell tools like grep and sed to locate relevant sections, then writes Python scripts to systematically extract the information.

Long-Context Understanding Trajectory (File Management)

```
[Turn 5] Action: execute_bash 'ls -la /testbed/documents/'  
Observation:  
DAF-COMP(2023)14.en.txt    146781 bytes  
The-State-of-Competition.txt   29048 bytes ...  
  
[Turn 8] Action: execute_bash 'grep -n -i 'infringement' ...'  
Observation:  
26: ... as measured by infringement notices issued by the ...  
252: ... estimate the number of consumer-related infringements ...  
  
[Turn 9] Action: execute_bash 'sed -n '240,280p' ...'  
// Jump to line range based on grep results  
Observation: There have been 12 infringement notices ...  
  
[Turn 15] Action: str_replace_editor create extract_industries.py  
// Write script to extract industry-infringement pairs  
def find_sentences_with_infringement_and_industry(text):  
    sentences = re.split(r'[!?.]', text)  
    ...  
    industry_matches = re.findall(r'(\b[\w\s]+)industry', ...)
```

Computation. In the instruction-following task, the model is asked to generate three sentences about medieval history, with a strict constraint: all sentences must have the *same character count* while using *entirely different words*. To satisfy this constraint, the model writes

Python scripts to count characters, detect word overlaps, and iteratively refine candidate sentences—a task that would be extremely difficult through pure text generation.

Instruction Following Trajectory (Computation)

```
[Turn 3] Action: str_replace_editor create helper.py
// Count characters and extract word sets
def char_count(s): return len(s)
def word_set(s): return set(re.findall(r'\b\w+\b', s.lower()))

[Turn 4] Action: execute_bash 'python3 helper.py'
Observation:
Lengths: [90, 95, 91]
Overlap between 0 and 1: {'the'}
// Constraints not satisfied

[Turn 12] Action: str_replace_editor create generate.py
// Combinatorial search over sentence templates
for s1, s2, s3 in itertools.product(political, religious, legal):
if check_constraints([s1, s2, s3]): candidates.append(...)

[Turn 13] Action: execute_bash 'python3 generate.py'
Observation:
Found 363 candidates with equal length and no overlapping words
```

2.4.2 Quantitative Analysis

We quantify these behavioral patterns to analyze how sandbox utilization varies across different models and task domains. Using the pattern matching approach described above, we measure how frequently models invoke each of the three core capabilities during their interactions, computed as the ratio of capability invocations to total interaction turns.

Variation across Task Domains. Figure 2 shows strong models adapt their usage patterns to task requirements. *Mathematics* benefits most due to high computation frequency (43.4%)—models verify solutions through numerical computation. *Chemistry* shows the highest external resource frequency (18.4%) as models install domain-specific packages. In contrast, *BioMedicine* shows less stable improvements with the shortest exploration (6.5 turns), suggesting that models fail to fully leverage the sandbox environment for these tasks.

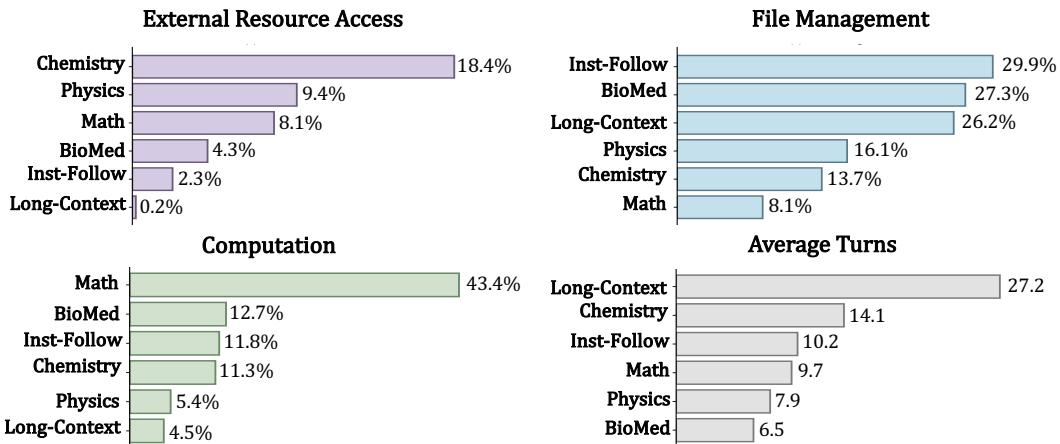


Figure 2: Sandbox behavior patterns across task domains for strong agentic models. (a)-(c): Capability usage rate, computed as capability invocations / total turns. (d): Average number of interaction turns per task.

Long-Context Tasks Benefit from File-based Context. In Figure 2, Long-Context tasks show high file operation frequency with minimal external resource usage, indicating that models focus on understanding local context. To further validate the benefit of file-based context handling, we compare two settings under LLM-in-Sandbox mode: placing documents directly in the prompt vs. storing them in the sandbox. As shown in Table 3, storing documents in the sandbox yields substantial gains on average, with Claude, DeepSeek, and Kimi showing the largest improvements. This suggests that LLM-in-Sandbox serves as a promising solution for handling long-context tasks by offloading extensive data to the environment. However, performance varies across models: Qwen perform worse with sandbox-based context, highlighting the need for training models to effectively explore file-based information.

Strong vs. Weak Models. Table 4 compares sandbox utilization between strong and weak models. Strong models effectively leverage all three capabilities with high usage rate (6–21%), while the weak model (Qwen3-4B-Instruct) achieves far lower frequency (<3%) despite taking nearly twice as many turns (23.7 vs. 12.6). This indicates that the weak model “wanders” in the sandbox without effective tool utilization—consuming more turns while accomplishing less.

Model Type	External	File	Computation	Avg. Turns
Strong Models	6.2%	21.1%	12.5%	12.6
Weak Model	0.8%	2.9%	2.9%	23.7

Table 4: Sandbox capability usage rate comparison: strong models (average of all models except Qwen3-4B-Instruct) vs. weak model (Qwen3-4B-Instruct). Usage rate = capability invocations / total turns.

3 LLM-in-Sandbox Reinforcement Learning Enhances Generalization

The preceding experiments demonstrate that the sandbox environment holds significant potential for enhancing general intelligence: strong agentic models consistently benefit across diverse domains. This raises a natural question: *can we directly train LLMs within this paradigm to further unlock their potential?* Motivated by this, we propose **LLM-in-Sandbox Reinforcement Learning (LLM-in-Sandbox-RL)**, which trains models on general context-based tasks within the sandbox, enabling them to effectively explore the sandbox environment without requiring specialized agentic data.

3.1 Method

To train LLMs to effectively utilize code sandboxes for general tasks, the ideal approach should satisfy two criteria: (1) training within a sandbox to learn to explore the environment, and (2) using general-domain data to ensure broad transferability. As shown in Table 5, existing approaches achieve some but not all of these goals. For example, vanilla RL training for LLMs on text-only tasks (hereafter LLM-RL; Lambert et al., 2024), though capable of leveraging general data, does not involve sandbox interaction; SWE-RL (Wei et al., 2025; Luo et al., 2025), which trains models on software engineering tasks within sandboxes, enables sandbox interaction but relies on domain-specific data.

We propose training LLMs within sandbox environments configured with general-purpose data. Specifically, we adopt *context-based tasks*: each task consists of background materials (e.g., documents) and a task objective that must be completed based on these materials. Since completing the objective depends on the provided materials, models must actively

Model	Prompt	Sandbox
Claude	11.9	61.8
GPT-5	66.3	66.8
DeepSeek	16.8	63.8
MiniMax	61.0	58.5
Kimi	51.0	61.8
Qwen-Coder	30.5	24.0
Qwen-4B	11.8	5.8
Average	35.6	48.9

Table 3: Long-context performance, both use LLM-in-Sandbox mode but place the context differently: in prompt vs. in sandbox.

explore the sandbox to find relevant information—naturally learning to leverage its capabilities. Meanwhile, our method uses only a general-purpose sandbox without task-specific configurations, making it easy to scale. Moreover, general-purpose tasks are much simpler to curate than software engineering tasks, enabling easy data scaling (Cheng et al., 2024; Jain et al., 2025). Overall, LLM-in-Sandbox-RL combines the benefits of sandbox-based training with general-domain data and easy scalability.

	LLM-RL	SWE-RL	LLM-in-Sandbox-RL
Sandbox Utilization	✗	✓	✓
General Domain	✓	✗	✓
Data Scalability	✓	✗	✓
Environment Scalability	N/A	✗	✓

Table 5: Comparison of different RL training paradigms. LLM-RL refers to RL training for LLMs on text-only tasks; SWE-RL refers to RL training on software engineering tasks within sandboxes.

Data Source. We source general data from context-based task datasets, specifically the seed data used for fine-tuning the synthesizer in *Instruction Pre-Training* (Cheng et al., 2024). The data covers diverse domains including encyclopedia, fiction, expert materials, academic tests, news, social media, and trivia. Each data instance has background material as context, along with a series of related tasks. The task types include free-form generation, multiple choice, and reasoning.

Sandbox Configuration. As illustrated in Figure 3, we store task contexts as files within the sandbox environment. To enrich the context and increase the task difficulty, we employ several strategies:

- **Multi-document or long contexts:** If a task is based on multiple documents or a single long document, we split them into separate files. For example, a research paper is divided into sections (e.g., introduction.txt, methods.txt, ...).
- **Single-file contexts with distractors:** If the context results in only one file, we sample multiple additional files from the same dataset as distractors, encouraging the model to navigate and filter relevant information.

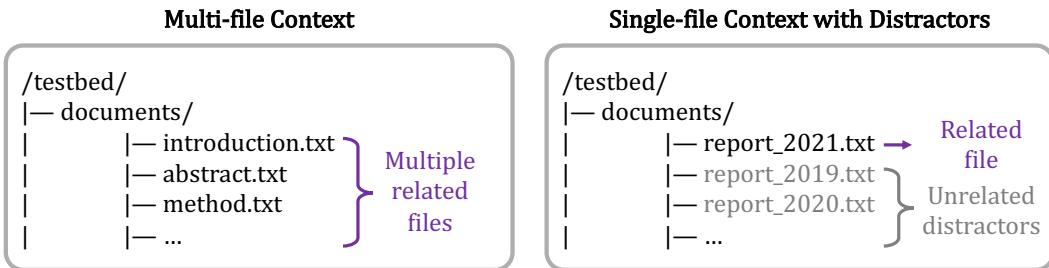


Figure 3: Sandbox Configuration for LLM-in-Sandbox-RL: task contexts are stored as files within the sandbox environment. (a) Multi-document or long contexts are split into separate files. (b) Single-file contexts are supplemented with distractors.

Task Setup. For each data instance, the context may correspond to multiple related tasks that depend on each other in a fixed order. As shown in Figure 4, we sample one task as the testing task, using prior tasks as in-context examples in the prompt. Additionally, we inform the model in the prompt that relevant files are in /testbed/documents/ and instruct it to write the

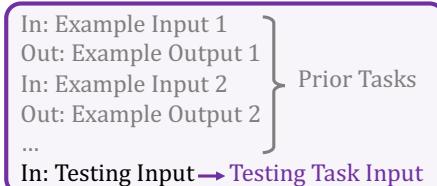


Figure 4: Task Setup. Prior related tasks are used as in-context examples.

	Math		Physics		Chem.		Biomed.		Long-Cont.		Instruct.		SWE	
	LLM	LLM	LLM	LLM	LLM	LLM	LLM	LLM	LLM	LLM	LLM	LLM	LLM	LLM
Qwen3-4B-Instruct-2507														
Base LLM	41.3	35.4	40.5	36.3	56.2	50.7	10.4	10.6	30.8	5.8	32.7	28.7	11.2	
LLM-RL	44.0	32.3	41.1	40.0	54.2	48.7	12.0	8.8	29.8	7.0	35.7	29.3	12.8	
Δ	+2.7	-3.1	+0.6	+3.7	-2.0	-2.0	+1.6	-1.8	-1.0	+1.2	+3.0	+0.6	+1.6	
(LLM)-RL	47.9	50.2	46.5	47.7	56.9	59.8	10.0	14.4	35.0	16.8	33.7	37.7	12.4	
Δ	+6.6	+14.8	+6.0	+11.4	+0.7	+9.1	-0.4	+3.8	+4.2	+11.0	+1.0	+9.0	+1.2	
Qwen3-Coder-30B-A3B														
Base LLM	17.9	42.1	36.8	47.9	50.2	55.8	13.4	14.8	27.3	24.0	35.0	40.0	45.0	
LLM-RL	14.6	42.1	39.9	46.0	47.6	56.7	17.0	17.2	30.8	21.8	34.3	26.3	47.6	
Δ	-3.3	0.0	+3.1	-1.9	-2.6	+0.9	+3.6	+2.4	+3.5	-2.2	-0.7	-13.7	+2.6	
(LLM)-RL	17.3	43.5	40.2	49.1	51.3	56.9	16.4	18.4	27.8	30.5	34.0	42.7	48.0	
Δ	-0.6	+1.4	+3.4	+1.2	+1.1	+1.1	+3.0	+3.6	+0.5	+6.5	-1.0	+2.7	+3.0	

Table 6: Main results comparing LLM-in-Sandbox-RL with LLM-RL baseline, both use general context-based task data. (LLM) denotes LLM-in-Sandbox mode. Δ indicates performance change from Base LLM (green = gain, red = decline).

final answer to `/testbed/answer.txt`. Upon task completion, we extract the answer from this file as the model’s final output.

RL Training. Recent advances in RL for LLMs typically adopt outcome-based rewards: given a prompt, the model generates a trajectory, and the entire trajectory is rewarded based on the correctness of the final output (Guo et al., 2025). Our RL baseline follows this paradigm to train LLMs on context-based tasks without sandbox interaction, which we refer to as LLM-RL. LLM-in-Sandbox-RL adopts the same framework, with the only difference being that trajectory generation uses LLM-in-Sandbox mode instead of vanilla LLM mode.

3.2 Experiments

Setup We train from two base models that exhibit different capability levels in our evaluation in Section 2: (1) *Qwen3-4B-Instruct-2507*, a small general-purpose LLM with weaker agentic capabilities that performs worse in LLM-in-Sandbox mode than in LLM mode; and (2) *Qwen3-Coder-30B-A3B*, a code-specialized model with strong agentic abilities that already shows better performance in LLM-in-Sandbox than in LLM mode. For benchmarks, in addition to the six non-code domains in Section 2, we also evaluate on software engineering (SWE) tasks: SWE-bench Verified (Jimenez et al., 2023), to examine whether training on general data would impair code agentic ability. Since SWE tasks inherently require a sandbox and have no LLM mode, we only report LLM-in-Sandbox results. Evaluation details are in Appendix C. We use rule-based functions for rewards. Detailed training settings are provided in Appendix E.

Main Results. Table 6 compares LLM-in-Sandbox-RL with the LLM-RL baseline. LLM-in-Sandbox-RL demonstrates broad generalization along three axes:

- **Domains:** Our training uses only general context-based data, with no overlap with the training or test sets of any evaluated benchmark. Yet LLM-in-Sandbox-RL improves performance across all domains, such as Long-Context, Math, Physics, and even tasks with vastly different formats such as Instruction-Following and SWE.
- **Model Capabilities:** For weaker models (*Qwen3-4B-Instruct*), LLM-in-Sandbox mode significantly outperforms LLM mode after LLM-in-Sandbox-RL training on most tasks (e.g., Biomed: 14.4 vs. 10.0, Instruction Following: 37.7 vs. 33.7). For stronger models (*Qwen3-Coder*), LLM-in-Sandbox-RL still yields consistent gains across domains.
- **Inference Modes:** Surprisingly, although trained exclusively in LLM-in-Sandbox mode, LLM-in-Sandbox-RL also improves LLM mode and even outperforms LLM-RL on

most tasks, suggesting that agentic skills can transfer back to non-agentic generation. In contrast, LLM-RL primarily improves LLM mode, with limited gains in LLM-in-Sandbox mode.

Data Source and Context Placement To understand the impact of training data, we compare four variants of LLM-in-Sandbox-RL on Qwen3-4B-Instruct-2507: (1) **Math**: mathematical data from DAPO (Yu et al., 2025); (2) **SWE**: software engineering data from R2E-Gym (Jain et al., 2025); (3) **Gen. in Prompt**: our general context-based data with context placed in the prompt; and (4) **Gen. in Sandbox**: the same data but with context placed in the sandbox. As shown in Table 7, all variants achieve some degree of cross-domain generalization, demonstrating the broad applicability of our training paradigm. Among them, Gen. Sandbox achieves the best overall performance. Notably, the comparison between Gen. Prompt and Gen. Sandbox highlights the importance of sandbox interaction: placing context in the sandbox forces the model to actively explore the environment, yielding stronger generalization than directly providing context in the prompt.

	Math		Physics		Chem.		Biomed.		Long-Cont.		Instruct.		SWE	
	LLM	[LLM]	LLM	[LLM]										
Base LLM	41.3	35.4	40.5	36.3	56.2	50.7	10.4	10.6	30.8	5.8	32.7	28.7	11.2	
Math	43.1	49.0	43.1	46.8	55.3	61.8	10.6	12.8	28.5	14.3	34.3	36.7	15.2	
SWE	46.9	30.0	46.2	42.5	53.8	51.1	10.4	12.2	29.3	7.8	33.3	31.7	17.4	
Gen. in Prompt	45.4	33.1	46.9	46.6	56.0	60.2	11.8	13.2	28.0	12.8	32.3	29.0	16.2	
Gen. in Sandbox	47.9	50.2	46.5	47.7	56.9	59.8	10.0	14.4	35.0	16.8	33.7	37.7	12.4	

Table 7: Data ablation: comparing LLM-in-Sandbox-RL with different training data (math-specific, SWE-specific, and general context-based). Gen. in Prompt and Gen. in Sandbox both use general context data but place the context differently: in prompt vs. in sandbox. [LLM] denotes LLM-in-Sandbox mode.

3.3 Analysis on Generalization

To understand why LLM-in-Sandbox-RL training leads to broad generalization, we first examine how models change their sandbox capability usage after training, and then investigate how sandbox-mode training transfers to LLM mode. We adopt the same capability classification and quantification method as in Section 2.4.

Generalization across Domains. Table 8 shows that after training, models exhibit increased sandbox capability usage across all three dimensions (external resources, file management, computation). As analyzed in Section 2, these capabilities benefit diverse task domains, explaining why the learned exploration skills transfer broadly.

Model		External	File	Computation	Avg. Turns
Qwen3-Coder-30B-A3B	Base LLM	5.7%	24.1%	11.1%	9.5
	[LLM]-RL	5.7%	24.4%	11.9%	10.0
Qwen3-4B-Instruct-2507	Base LLM	0.8%	2.9%	2.9%	23.7
	[LLM]-RL	4.1%	7.3%	7.2%	7.0

Table 8: Sandbox behavior usage rate changes after LLM-in-Sandbox-RL training. Usage rate = capability invocations / total turns.

Generalization across Model Capabilities. As shown in Table 8, weaker models (Qwen3-4B) show larger improvements: capability usage rate increases substantially while average turns decrease dramatically from 23.7 to 7.0. Recall from Section 2.4.2 that the base Qwen3-4B model “wanders” in the sandbox with many ineffective turns; after LLM-in-Sandbox-RL training, the model learns to accomplish tasks with fewer but more purposeful interactions. Stronger models (Qwen3-Coder) already have high capability usage, so improvements are more modest but still consistent.

Generalization across Inference Modes. We analyze reasoning patterns in outputs of vanilla LLM mode before and after LLM-in-Sandbox-RL training. Specifically, we measure two categories of behaviors through pattern matching: (1) *Structural Organization*—Markdown formatting elements (headers, separators, bullet points, math blocks) that indicate explicit step-by-step reasoning; and (2) *Verification Behaviors*—phrases indicating self-checking (e.g., “let’s verify”, “check that”, confirmation markers). As shown in Table 9, both models exhibit increased structural organization and verification behaviors after training. These reasoning patterns, learned through multi-turn sandbox interaction where each action receives explicit feedback, transfer to LLM mode even without sandbox access.

Model	Verification		Structure	
	Base LLM	[LLM]-RL	Base LLM	[LLM]-RL
Qwen3-Coder-30B-A3B	0.77	0.88	10.30	16.12
Qwen3-4B-Instruct-2507	20.22	36.91	19.13	20.64

Table 9: Reasoning pattern changes in outputs of vanilla LLM mode after LLM-in-Sandbox-RL training. Values are counts per response.

4 LLM-in-Sandbox Enables Efficient Deployment

We analyze practical considerations of deploying LLM-in-Sandbox in real-world systems from two perspectives: computational analysis (Section 4.1) and sandbox infrastructure overhead (Section 4.2). We conduct experiments with local model serving across different LLMs and serving engines: DeepSeek-V3.2-Thinking and Kimi-K2-Thinking are served using SGLang (Zheng et al., 2024), while MiniMax-M2 and Qwen3-Coder-30B-A3B are served using vLLM (Kwon et al., 2023). We use a single NVIDIA DGX node for all experiments, with query concurrency set to 64, sampling the same number of task instances from each benchmark, and other settings following Section 2.3.

4.1 Computational Analysis

Cost. We first measure the total token consumption per query, which directly reflects the compute budget since inference FLOPs scale linearly with token count. As shown in Table 10, the results vary across task types. For most tasks, LLM-in-Sandbox consumes more tokens due to multi-turn exploration. However, for long-context tasks, LLM-in-Sandbox dramatically reduces tokens by storing content in local files rather than in the prompt. The reduction reaches up to $8\times$ for Qwen (100K \rightarrow 13K tokens). When aggregating across all tasks, LLM-in-Sandbox consumes only **0.5–0.8×** the total tokens of LLM mode.

Task	DeepSeek			MiniMax			Kimi			Qwen		
	LLM	[LLM]	Δ	LLM	[LLM]	Δ	LLM	[LLM]	Δ	LLM	[LLM]	Δ
Math	14.9	16.8	+1.9	22.4	20.9	-1.5	25.7	14.9	-10.8	2.5	10.4	+7.9
Phy.	8.2	11.6	+3.4	8.1	7.7	-0.4	10.0	13.6	+3.6	1.5	6.5	+5.0
Chem.	3.6	22.0	+18.4	9.4	10.9	+1.5	5.3	13.2	+7.9	0.6	10.5	+9.9
Biomed.	2.5	11.5	+9.0	3.2	6.9	+3.7	4.2	9.8	+5.6	0.7	4.2	+3.5
Long.	90.3	25.4	-64.9	88.4	13.6	-74.8	91.8	21.7	-70.1	102.9	12.9	-90.0
Inst.	2.4	14.9	+12.5	4.2	8.8	+4.6	6.0	9.6	+3.6	1.6	8.9	+7.3
Average	20.3	17.0		22.6	11.5		23.8	13.8		18.3	8.9	
Ratio		0.84×			0.51×			0.58×			0.49×	

Table 10: Token consumption per query (in thousands). Each cell shows total tokens (prompt + model-generated + environment-generated tokens). Δ = LLM-in-Sandbox – LLM. The “Ratio” row shows $\sum N_{\text{LLM-in-Sandbox}} / \sum N_{\text{LLM}}$ computed over all tasks, reflecting the overall token savings. [LLM] denotes LLM-in-Sandbox mode.

Speed. In LLM-in-Sandbox mode, a significant portion of tokens come from the environment, such as code execution results. Unlike model-generated tokens that require slow autoregressive decoding, environment tokens are processed via the fast *Prefill* (Dao et al., 2022). As shown in Table 11, environment tokens constitute 37%–51% of the trajectory, yet environment execution accounts for less than 4% of total time. We measure end-to-end query throughput using QPM (Queries Per Minute), i.e., the number of queries processed per unit time from submission to final answer. Overall, LLM-in-Sandbox achieves competitive throughput: MiniMax achieves $2.2\times$ speedup, while others range from $0.6\times$ to $1.1\times$.

	$N_{\text{env}} / N_{\text{total}}$	$T_{\text{exe}} / T_{\text{total}}$	QPM Ratio
DeepSeek	43.6%	2.3%	0.6×
MiniMax	51.1%	2.2%	$2.2\times$
Kimi	36.9%	1.9%	$1.0\times$
Qwen	50.3%	3.5%	$1.1\times$

Table 11: Inference efficiency of LLM-in-Sandbox averaged over tasks. $N_{\text{env}} / N_{\text{total}}$: fraction of tokens from environment, processed via fast prefill rather than slow decoding. $T_{\text{exe}} / T_{\text{total}}$: fraction of total time spent on environment execution. QPM Ratio: $\text{QPM}_{\text{LLM-in-Sandbox}} / \text{QPM}_{\text{LLM}}$; values $\geq 1\times$ indicate comparable or faster throughput.

4.2 Sandbox Infrastructure

We analyze the infrastructure overhead of these sandboxes in terms of storage, memory. A key advantage of LLM-in-Sandbox is its *general, lightweight sandbox* design. Table 12 summarizes the infrastructure overhead, which is negligible in practice. For storage, typical code agent often requires task-specific environments with particular dependencies; in contrast, LLM-in-Sandbox employs a single Docker image (~ 1.1 GB) shared across all tasks. Models autonomously install task-specific packages at runtime, reducing storage by orders of magnitude. For memory, each sandbox container consumes only ~ 50 MB idle and ~ 200 MB at peak. Even with $K = 512$ concurrent sandboxes on a single DGX node, the memory overhead is only 5% of the system RAM.

Dataset	Storage	Configuration	Memory	% of 2TB
SWE-Gym (Pan et al., 2024)	6 TB	Per container (idle)	50 MB	–
SWE-Smith (Yang et al., 2025b)	295 GB	Per container (peak)	200 MB	–
SWE-bench Verified (Jimenez et al., 2023)	257 GB	$K = 64$ sandboxes	13 GB	0.7%
LLM-in-Sandbox	1.1 GB	$K = 512$ sandboxes	100 GB	5%

Table 12: **Left:** Storage overhead comparison. LLM-in-Sandbox uses a general-purpose lightweight image, reducing storage by orders of magnitude. **Right:** Memory overhead on a DGX node with 2 TB system RAM.

5 LLM-in-Sandbox Goes Beyond Text Generation

Previous sections evaluate LLM-in-Sandbox on tasks where both vanilla LLMs and LLM-in-Sandbox can produce outputs for end-to-end comparison. However, LLM-in-Sandbox also enables capabilities that are *fundamentally impossible* for standalone LLMs. By granting LLMs access to a virtual computer, LLM-in-Sandbox transcends the text-in-text-out paradigm and unlocks new possibilities:

- **Cross-Modal Capabilities:** LLMs are confined to text-in-text-out, but LLM-in-Sandbox enables processing and generating images, videos, audio, and interactive applications by orchestrating specialized software within the sandbox.
- **File-Level Operations:** Rather than describing what a file should contain, LLM-in-Sandbox directly produces actual files—.png, .mp4, .wav, .html—that users can immediately use, with grounded feedback from real execution.

- **Autonomous Tool Acquisition:** Unlike predefined tool-use where LLMs call fixed APIs, LLM-in-Sandbox enables LLMs to autonomously discover, install, and learn to use arbitrary software libraries on demand—effectively granting unlimited tool access.

Case Studies. We demonstrate these capabilities through four representative examples in Figure 5. Full trajectories and interactive demos are available at our project page.

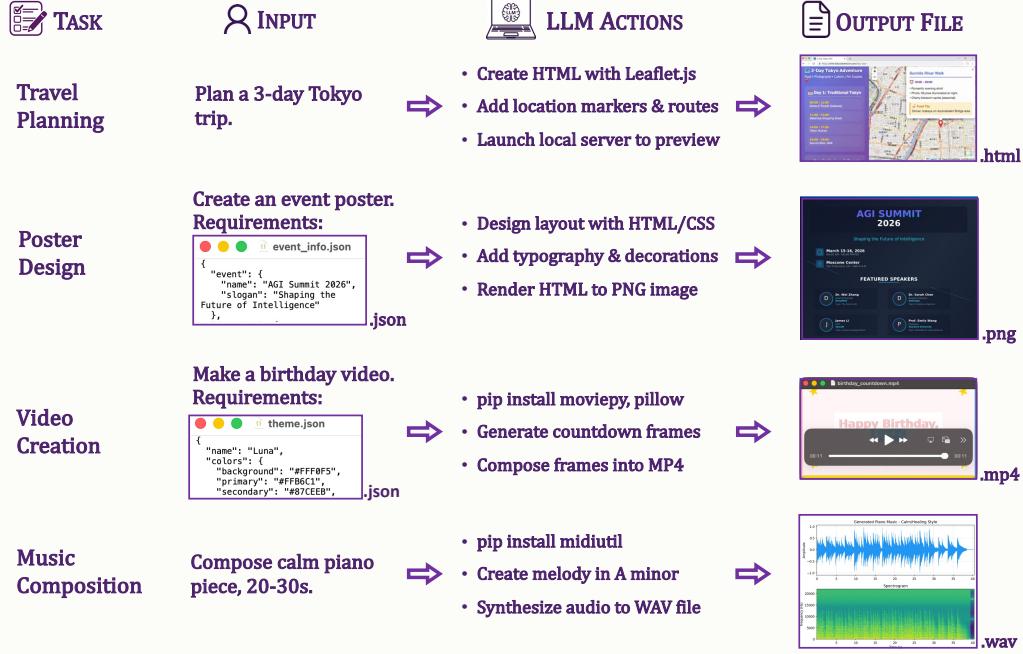


Figure 5: LLM-in-Sandbox transcends the text-in-text-out paradigm. By granting LLMs access to a basic virtual computer, they can autonomously install tools, write and execute programs, and produce usable files—interactive webpages (.html), images (.png), videos (.mp4), and audio (.wav).

Case 1: Travel Planning → Interactive Map. Given a natural language query for a 3-day Tokyo trip itinerary, the agent installs Leaflet.js, designs a data structure for 12 locations, generates JavaScript for markers with popups and color-coded route polylines, producing a fully functional map .html with clickable markers and day-by-day visualization.

Case 2: Event Specification → Conference Poster. From a JSON file containing event details (“AGI Summit 2026”, venue, speakers, sessions), the agent designs an SVG layout with gradient backgrounds, implements typography hierarchy, and converts to PNG via CairoSVG¹, outputting professional poster .svg and poster .png files.

Case 3: Theme Configuration → Animated Video. Given a JSON theme specifying recipient name, color palette, and aesthetic style, the agent generates 360 frames using PIL with animated decorations, compiles them at 30fps via moviepy², producing a 11-second birthday_countdown.mp4 .

Case 4: Style Description → Original Music. From a natural language request for a calm piano piece in A minor, the agent uses midiutil³ to compose melody and chord progressions, renders audio via FluidSynth⁴, outputting composition.mid, preview.wav, and sheet_music.md.

¹<https://github.com/Kozea/CairoSVG>

²<https://github.com/Zulko/moviepy>

³<https://github.com/MarkCWirt/MIDIUtil>

⁴<https://github.com/FluidSynth/fluidsynth>

Discussion. While these examples demonstrate the potential of LLM-in-Sandbox to achieve general intelligence beyond text generation, we acknowledge that current results have limitations. The generated videos are limited to simple 11-second animations without complex scenes. The composed music, though structurally correct, lacks the expressiveness and creativity of human compositions. The posters follow basic design principles but may not match professional graphic design quality.

Nevertheless, these cases reveal a promising direction: as LLMs become more capable and sandbox environments more sophisticated, LLM-in-Sandbox could evolve into a truly general-purpose digital creation system. We believe this paradigm—LLMs interacting with computational environments rather than generating text in isolation—represents a compelling path toward general intelligence.

6 Conclusion and Future Work

LLM-in-Sandbox as Default Inference Infrastructure We introduce LLM-in-Sandbox, a paradigm that grants LLMs access to a virtual computer and show that strong LLMs exhibit emergent capabilities to leverage this environment for general tasks. Looking forward, we envision LLM-in-Sandbox becoming the default paradigm for serving LLMs: analytical tasks gain verifiable computation, long-context tasks benefit from file-based management, and creative tasks yield actual outputs—images, videos, applications—rather than text descriptions. We anticipate sandbox environments will become standard infrastructure, transforming LLMs from text generators into general-purpose digital workers.

LLM-in-Sandbox as an Agentic Capability Benchmark Beyond serving as an inference paradigm, LLM-in-Sandbox naturally provides a standardized testbed for evaluating agentic capabilities. Unlike existing benchmarks that focus on specific downstream tasks, LLM-in-Sandbox measures fundamental skills like exploration, tool use, and self-verification through a unified framework. The metric $\Delta = \text{LLM-in-Sandbox} - \text{LLM}$ offers a meaningful indicator: it quantifies how effectively a model can leverage computational environments, revealing agentic potential that raw LLM performance alone cannot capture.

Sandbox-Native Model Training We propose LLM-in-Sandbox-RL, a lightweight RL method that trains sandbox interaction as a transferable skill using only general, non-agentic data. Looking ahead, we advocate for *sandbox-native* models where sandbox interaction becomes a first-class training objective—not only through large-scale RL with real environmental feedback, but also by incorporating sandbox-style reasoning into the pretraining stage itself.

Acknowledgments

The first author would like to thank Yeqie Wang, Lisheng Huang, and Shuang Sun for helpful discussions, and the R2E-Gym (Jain et al., 2025), DeepSWE (Luo et al., 2025), and rLLM (Tan et al., 2025) teams for their valuable open-source contributions.

References

- Anthropic. Claude code, 2025a. URL <https://claude.com/product/claude-code>.
- Anthropic. Claude sonnet, 2025b. URL <https://www.anthropic.com/claude/sonnet>.
- Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- Daixuan Cheng, Yuxian Gu, Shaohan Huang, Junyu Bi, Minlie Huang, and Furu Wei. Instruction pre-training: Language models are supervised multitask learners. In *Proceedings*

of the 2024 Conference on Empirical Methods in Natural Language Processing, pp. 2529–2550, 2024.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

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.

HuggingFace. Math-verify, 2025. URL <https://github.com/huggingface/Math-Verify>.

Naman Jain, Jaskirat Singh, Manish Shetty, Tianjun Zhang, Liang Zheng, Koushik Sen, and Ion Stoica. R2e-gym: Procedural environment generation and hybrid verifiers for scaling open-weights swe agents. In *Second Conference on Language Modeling*, 2025.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. T\ ulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pp. 74–81, 2004.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

Michael Luo, Naman Jain, Jaskirat Singh, Sijun Tan, Ameen Patel, Qingyang Wu, Alpay Ariyak, Colin Cai, Tarun Venkat, Shang Zhu, Ben Athiwaratkun, Manan Roongta, Ce Zhang, Li Erran Li, Raluca Ada Popa, Koushik Sen, and Ion Stoica. Deepswe: Training a state-of-the-art coding agent from scratch by scaling rl, 2025. Notion Blog.

MAA. American invitational mathematics examination - aime, 2025. URL <https://maa.org/>.

MiniMax. Minimax m2 & agent: Ingenious in simplicity, 2025. URL <https://www.minimax.io/news/minimax-m2>.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with swe-gym. *arXiv preprint arXiv:2412.21139*, 2024.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. Generalizing verifiable instruction following. *arXiv preprint arXiv:2507.02833*, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

-
- Sijun Tan, Michael Luo, Colin Cai, Tarun Venkat, Kyle Montgomery, Aaron Hao, Tianhao Wu, Arnav Balyan, Manan Roongta, Chenguang Wang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. rllm: A framework for post-training language agents, 2025. Notion Blog.
- Artificial Analysis Team. Artificial analysis long context reasoning benchmark(lcr), 2025.
- Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.
- Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
- 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. *arXiv preprint arXiv:2502.18449*, 2025.
- Xin Xu, Qiyun Xu, Tong Xiao, Tianhao Chen, Yuchen Yan, Jiaxin ZHANG, Shizhe Diao, Can Yang, and Yang Wang. Ugphysics: A comprehensive benchmark for undergraduate physics reasoning with large language models. In *Forty-second International Conference on Machine Learning*.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengan Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.
- John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.
- John Yang, Kilian Lieret, Carlos E Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents. *arXiv preprint arXiv:2504.21798*, 2025b.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.
- Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.
- Di Zhang, Wei Liu, Qian Tan, Jingdan Chen, Hang Yan, Yuliang Yan, Jiatong Li, Weiran Huang, Xiangyu Yue, Wanli Ouyang, et al. Chemllm: A chemical large language model. *arXiv preprint arXiv:2402.06852*, 2024.
- Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.
- Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sqlang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

Yuxin Zuo, Shang Qu, Yifei Li, Zhang-Ren Chen, Xuekai Zhu, Ermo Hua, Kaiyan Zhang, Ning Ding, and Bowen Zhou. Medxpertqa: Benchmarking expert-level medical reasoning and understanding. In *Forty-second International Conference on Machine Learning*.

A Sandbox Implementation

We build upon the sandbox framework of R2E-Gym (Jain et al., 2025), adapting it for general-purpose exploration across diverse non-code domains. Complete specifications of the tools available within the sandbox are detailed below.

execute_bash

Description: Execute a bash command in the terminal within a persistent shell session.

- *One command at a time*: You can only execute one bash command at a time. If you need to run multiple commands sequentially, use `&&` or `;` to chain them together.
- *Persistent session*: Commands execute in a persistent shell session where environment variables, virtual environments, and working directory persist between commands.
- *Soft timeout*: Commands have a soft timeout of 10 seconds, once that's reached, you have the option to continue or interrupt the command.
- *Output truncation*: If the output exceeds a maximum length, it will be truncated before being returned.

Parameters:

- `command` (string, required): The bash command to execute. For example: `python my_script.py`.

str_replace_editor

Description: Custom editing tool for viewing, creating and editing files.

- State is persistent across command calls and discussions with the user.
- If `path` is a file, `view` displays the result of applying `cat -n`. If `path` is a directory, `view` lists non-hidden files and directories up to 2 levels deep.
- The `create` command cannot be used if the specified path already exists as a file.
- For the `str_replace` command, the `old_str` parameter should match EXACTLY one or more consecutive lines from the original file.

Parameters:

- `command` (string, required): The command to run. Allowed options are: `view`, `create`, `str_replace`, `insert`.
- `path` (string, required): Absolute path to file or directory.
- `file_text` (string, optional): Required for `create` command.
- `old_str` (string, optional): Required for `str_replace` command.
- `new_str` (string, optional): The replacement string for `str_replace`, or the string to insert for `insert`.
- `insert_line` (integer, optional): Required for `insert` command.
- `view_range` (array, optional): Line range for `view` command, e.g., [11, 12].

submit

Description: Finish the interaction when the task is complete OR if the assistant cannot proceed further with the task.

Parameters: No parameters are required for this function.

B Model Configurations

Table 13 summarizes the inference configurations used for each model. The maximum turn is set as 100. The maximum generation length per turn is set to 65,536 tokens, except for Claude-Sonnet-4.5-Think which is limited to 64,000 tokens due to API constraints. For vanilla LLM mode, this represents the maximum tokens generated in a single response given the prompt. For LLM-in-Sandbox, this limit applies to each turn, and the total trajectory length (including prompt, model output and environment output) is also capped at the same value, except for the long-context understanding task where we set the trajectory limit to 131,072 tokens to accommodate the longer context.

Model	Temperature	Top_p	Min_p	Top_k	Rep. Penalty	Backend
Claude-Sonnet-4.5-Think	1.0	-	-	-	-	API
GPT-5	1.0	-	-	-	-	API
DeepSeek-V3.2-Thinking	1.0	0.95	-	-	-	SGLang
MiniMax-M2	1.0	0.95	-	40	-	vLLM
Kimi-K2-Thinking	1.0	-	-	-	-	SGLang
Qwen3-Coder-30B-A3B	0.7	0.80	0.0	20	1.05	vLLM
Qwen3-4B-Instruct-2507	0.7	0.80	0.0	20	-	vLLM

Table 13: Inference configurations for each model. We use the recommended sampling parameters by each model supplier. “-” indicates the parameter is not applicable or uses the default value. The thinking budget for Claude is 60,000 tokens.

C Evaluation Details

We evaluate on six diverse non-code domains, summarized in Table 14. The system prompt largely follows the one in Appendix F, with minor domain-specific adjustments and final answer formatting instructions. Please refer to our released code for the exact prompts.

Domain	Benchmark	# Problems	Evaluation
Mathematics	AIME25 (MAA, 2025)	30×16	Math-Verify
Physics	UGPhysics (Xu et al.)	650	LLM Judge
Chemistry	ChemBench (Zhang et al., 2024)	450	Exact Match
Biomedicine	MedXpertQA (Zuo et al.)	500	Exact Match
Long-Context	AA-LCR (Team, 2025)	100×4	LLM Equality Checker
Instruction Following	IFBench (Pyatkin et al., 2025)	300	Rule-based (Loose)
Software Engineering	SWE-bench Verified (Jimenez et al., 2023)	500	Rule-based

Table 14: Summary of evaluation benchmarks. “ $\times N$ ” indicates each problem is repeated N times.

Mathematics. We use all 30 problems from the 2025 American Invitational Mathematics Examination (AIME25), which tests olympiad-level mathematical reasoning. Given the small dataset size, we repeat each problem 16 times and report average accuracy. The prompt includes “Please reason step by step, and put your final answer within `\boxed{}`.” and we use Math-Verify ([HuggingFace, 2025](#)) for evaluation.

Physics. UGPhysics is a comprehensive benchmark for evaluating physics problem-solving at the undergraduate level, spanning 13 core subjects. We sample 50 problems from each subject, yielding 650 problems in total. Responses are evaluated using an LLM-based judge ([Zheng et al., 2023](#)) with Qwen3-30B-A3B-Instruct-2507 ([Yang et al., 2025a](#)).

Chemistry. ChemBench assesses chemistry competency through single-choice questions across nine core tasks. We sample 50 problems from each sub-domain (450 total) and use exact match for evaluation.

Biomedicine. MedXpertQA is designed to evaluate expert-level medical knowledge and reasoning through multiple-choice questions. We use only the text-based questions, sampling 500 instances, and evaluate via exact match.

Long-Context Understanding. AA-LCR contains 100 challenging questions requiring multi-document reasoning, with each document set averaging approximately 100K tokens. Answers must be derived through reasoning rather than direct retrieval. In LLM-in-Sandbox mode, each problem is initialized with its own sandbox environment, where all related documents are stored as text files in `/testbed/documents/`, each named after its

original title. We repeat each problem 4 times and report average accuracy. Following the original work, we use an LLM-based equality checker for evaluation with Qwen3-235B-A22B-Instruct-2507 (Yang et al., 2025a).

Instruction Following. IFFBench tests precise instruction-following across 58 diverse, verifiable constraints. We use the single-turn subset (300 questions) and evaluate with the official code in loose mode, which handles formatting variations by checking multiple output forms.

Software Engineering. SWE-bench is a comprehensive benchmark for software engineering tasks, including code generation, debugging, and comprehension. We use the verified subset (500 problems) and evaluate using the official rule-based evaluation script. We leverage the SWE-bench sandbox setup from R2E-Gym (Jain et al., 2025) for code execution. The system prompt follows OpenHands (Wang et al., 2024), and the toolset is the same as described in Section 2.2.

D Sandbox Capability Classification

We classify model actions into three capability categories through pattern matching on bash commands and Python code. Table 15 summarizes the classification patterns for each category.

Category	Pattern Type	Examples
External Resources	Package installation HTTP requests Web scraping Domain libraries	pip install, apt-get install requests.get, curl, wget BeautifulSoup, selenium rdkit, biopython, pubchempy
File Management	Python file I/O Shell file commands Path operations Data serialization	open(), json.load, pd.read_csv cat, grep, find, head/tail os.path, pathlib, glob pickle.load, np.load/save
Computation	Numerical solvers Integration Iterative algorithms Combinatorics	scipy.optimize, fsolve, minimize odeint, solve_ivp, quad Large loops (range(N) where $N > 100$), while loops itertools.permutations/combinations

Table 15: Pattern matching rules for detecting sandbox capability usage.

For each trajectory, we extract all code blocks from model actions (both Python scripts and bash commands) and apply these patterns. The capability usage rate is computed as the number of turns containing at least one matched pattern divided by the total number of interaction turns.

E LLM-in-Sandbox-RL Training Details

We train LLM-in-Sandbox-RL following the training framework of DeepSWE (Luo et al., 2025) in rLLM (Tan et al., 2025). Based on this, we penalize excessively long trajectories: if the model exceeds maximum turns/tokens without submitting an answer, the episode is terminated with zero reward. The sandbox configuration is identical to that described in Section 2. Table 16 summarizes the key hyperparameters.

Reward Design. We use rule-based reward functions tailored to each task type: (1) for multiple-choice tasks, we assign a positive reward for selecting the correct option and 0 otherwise; when multiple correct options exist, we use F1 score as the reward; (2) for free-form generation tasks, we use ROUGE-L (Lin, 2004) score; (3) for tasks with binary

correctness (e.g., math problems), we use a simple binary reward (+1 for correct, 0 for incorrect).

Hyperparameter	Qwen3-4B-Instruct-2507	Qwen3-Coder-30B-A3B
RL Algorithm	GRPO++	GRPO++
Learning Rate	1e-6	1e-6
Train (Prompt) Batch Size	8	8
Update mini batch size	8	8
Rollouts per Prompt	8	8
Train Steps	150	50
Max Turns	100	100
KL Reward/Loss	None	None
Rollout Temperature	1.0	1.0
Rollout Top_p	0.8	0.8
Rollout Top_k	20	20
Max Response Length	65,536 Tokens	65,536 Tokens

Table 16: Hyperparameters for LLM-in-Sandbox-RL training. GRPO++ refers to the variants of GRPO (Shao et al., 2024) used in DeepSWE (Luo et al., 2025). Update mini batch size indicates the batch size used for each policy update step, we set it as the same as the train batch size, meaning we perform one update per batch (i.e., on-policy).

F Prompt for LLM-in-Sandbox

The prompts used in our experiments are shown in Figure 6 and Figure 7. This represents a minimal baseline prompt that establishes the core sandbox interaction protocol. The prompt can be easily adapted to different use cases by modifying the input/output format specifications (e.g., changing the output file path or format), adding domain-specific instructions, or incorporating additional tools.

Instance Prompt Template for LLM-in-Sandbox

```

<problem>
{problem_statement}
</problem>

Please solve this problem.

<OUTPUT_INSTRUCTIONS>
- If the task requires a specific answer (e.g., a number, text, or computation result):
  write the final answer to /testbed/output/answer.txt (plain text, answer only, no
  explanations)
- If the task requires creating a project, code, or multiple files: save all files
  directly to /testbed/output/
</OUTPUT_INSTRUCTIONS>

Working directory: /testbed
Input files (if any): /testbed/input
Output directory: /testbed/output

```

Figure 6: The Instance Prompt Template. The {problem_statement} placeholder is filled with the actual problem description for each task instance.

System Prompt for LLM-in-Sandbox

You are an expert specializing in solving complex problems using code.

<TASK>

You need to complete the given task by following the instructions precisely.
</TASK>

<DIRECTORIES>

- Working directory: /testbed
 - Input directory: /testbed/input <-- User-provided input files/assets are here
 - Output directory: /testbed/output <-- Put ALL your outputs here
- </DIRECTORIES>**

<WORKFLOW>

1. Read the problem carefully
 2. Check /testbed/input for any input files if the task mentions them
 3. Analyze the problem and determine the solution approach
 4. **MUST write code to a file and execute it** - DO NOT just think about the answer, use print statements directly, or hardcode answers
 5. Save all outputs to /testbed/output directory
- </WORKFLOW>**

<IMPORTANT_NOTES>

- Use execute_bash to run scripts or commands
 - Use str_replace_editor to view, create and edit files
 - Use submit to finish once you have completed the task
- </IMPORTANT_NOTES>**

<ENCOURAGED_APPROACHES>

- You have full access to this isolated environment - feel free to install packages, create files, run experiments, etc.
 - Explore diverse problem-solving approaches: use libraries, tools, external data, computations, simulations, or any method that helps
 - The environment is sandboxed for your use - be creative and try different computational strategies
 - The more comprehensive and computational your approach, the better
- </ENCOURAGED_APPROACHES>**

<ANTI_HARDCODING>

STRICTLY PROHIBITED: Do not use large comment blocks or print statements for natural language thinking (e.g., '# Let me think step by step...' or 'print("First, I need to consider...")'). Do not hardcode answers like 'answer = "A"' or 'return 'A''. You must derive the final result through actual computational logic, mathematical operations, and programmatic analysis of the problem data.

</ANTI_HARDCODING>

Figure 7: The System Prompt used for LLM-in-Sandbox.