STT-ARENA: A MORE REALISTIC ENVIRONMENT FOR TOOL-USING WITH SPATIO-TEMPORAL DYNAMICS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) deployed in real-world agentic applications must be capable of replanning and adapting when mid-task disruptions invalidate their prior decisions. Existing dynamic benchmarks primarily measure whether LLMs can detect temporal changes in a timely manner, leaving the complementary challenge of adaptive replanning under spatio-temporal dynamics largely unexplored. We introduce STT-Arena (Spatio-Temporal Tool-Use Arena), a benchmark of 227 high-quality interactive tasks spanning nine spatio-temporal conflict types and four solvability levels. Each task is grounded in a realistic, executable environment equipped with injected spatio-temporal triggers that can abruptly invalidate an ongoing plan, forcing the model to detect the state shift and construct a revised execution strategy. Extensive evaluation of frontier LLMs reveals that even the SOTA proprietary models, including Claude-4.6-Opus, achieves less than 40% overall accuracies, highlighting the fundamental difficulty of spatio-temporal dynamic reasoning. Systematic analysis of failure trajectories uncovers three recurring error modes of existing models: Stale-State Execution, Misdiagnosis of Dynamic Triggers, and Missing Post-Adaptation Verification. Guided by these findings, we propose an iterative trajectory refinement technique that eliminates these failure patterns from training data, and combine it with online RL to produce STT-Agent-4B which outperforms frontier LLMs on STT-Arena.

1 INTRODUCTION

Large language models (LLMs) based agents are increasingly deployed in real-world commercial applications, including airline reservation systems, clinical consultation services, etc (Team, 2025; 2026; GLM, 2026; Team & AI, 2025; Song et al., 2026; Sun et al., 2026). In these settings, LLMs must interact with external environments to retrieve information and execute multi-step operations. While early studies (Li et al., 2023; Qin et al., 2024; Yao et al., 2024; He et al., 2025) focus on static environment benchmarks with fixed interfaces and predictable outputs, more recent works such as GAIA-2 (Froger et al., 2026), Real-Time Reasoning Gym (Wen et al., 2025), and Timely Machine (Ma et al., 2026) introduce continuously evolving environments. These benchmarks emphasize real-time environmental change and measure how rapidly an LLM can respond to external dynamics, treating responsiveness as the primary indicator of competence under non-stationary conditions.

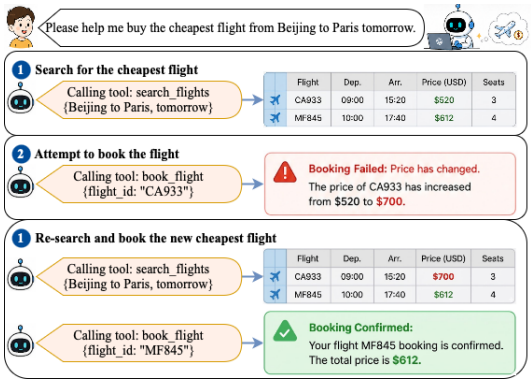

Figure 1: Adaptive replanning in spatio-temporal environments: A mid-task price change invalidates the plan, prompting detection of updated prices and reselection of the optimal flight.

Despite these advances, existing dynamic benchmarks focus on timely completion under continuously evolving environments. In this work, we address a complementary dimension: ***adaptive replanning and recovery***, the ability to abandon a failed plan and reconstruct an alternative multi-step strategy when a sudden spatio-temporal change invalidates prior decisions. As illustrated in

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, consider an LLM tasked with purchasing the cheapest available flight ticket. Midway through execution, ticket prices shift due to spatio-temporal dynamics. Beyond simply detecting the change, a capable LLM must reassess its prior decisions and construct a revised plan to complete the task correctly. We argue that the ability to replan and reconsider under mid-task environmental shifts is essential for trustworthy real-world deployment.

To systematically characterize the environmental dynamics that necessitate such replanning, we identify three fundamental axes along which real-world conditions evolve. ***Temporal evolution*** refers to state changes that unfold over time: for instance, seat availability changes continuously as passengers complete bookings during online check-in. ***Spatial dependency*** captures how environmental conditions vary with geographic context: delivery services, for example, are bound by predefined service zones, and a change in location may render certain operations inaccessible or alter operational constraints. ***Spatio-temporal dynamics*** arise when a task is jointly governed by both dimensions: during rush hour, traffic congestion propagates across different locations at different times, making it unreliable to plan a route without accounting for their coupled interaction.

Grounded in these three axes, we introduce **STT-Arena** (Spatio-Temporal Tool-Use Arena), a dynamic and interactive benchmark designed to evaluate the replanning and adaptive tool-use capabilities of LLMs under spatio-temporal conditions (Table 1 shows the comparison of existing work). Built upon a diverse collection of authentic scenarios, STT-Arena makes three core contributions:

- **A fine-grained, spatio-temporally centered task taxonomy.** We curate a comprehensive set of authentic tool-use tasks organized into 3 major categories and 9 subcategories that explicitly capture temporal evolution, spatial dependency, and their coupled interactions. Each task is paired with a dedicated executable environment that faithfully and controllably simulates real-world spatio-temporal dynamics across diverse domains. Tasks are further stratified by solvability: *solvable* tasks are divided into three difficulty levels (*Easy*, *Medium*, and *Hard*), where the LLM must identify a feasible action sequence under evolving conditions; *impossible* tasks require the LLM to recognize that no valid completion path exists and to correctly report task infeasibility. Detailed information about the nine subcategories and four levels can be found in Tables 11 and 12.
- **A scalable, interactive infrastructure for spatio-temporal environment simulation.** As shown in Figure 2, we introduce a dynamic simulation framework built around three core components: environment curation, spatio-temporal dynamic injection, and dual-agent assessment. This design enables the systematic evaluation and targeted improvement of model tool-use capabilities under dynamically changing conditions, offering a reproducible, extensible, and high-fidelity testbed that closely mirrors real-world deployment scenarios.
- **Extensive benchmarking results and an efficient training paradigm for spatio-temporal tool augmentation.** As shown in Figure 3, we evaluate closed-source and open-source LLMs at scale, exposing critical deficiencies in their ability to replan and operate under dynamic tool-use constraints. Guided by analyses of recurring failure modes, we introduce an iterative trajectory refinement technique that post-processes training trajectories by reordering, deleting, or modifying tool-call blocks to eliminate inefficient interaction patterns. Building on 2,212 refined trajectories for SFT and a set of verifiable dynamic tasks for online RL, we release **STT-Agent-4B**, which achieves 27.17% on STT-Arena, matching the performance of GLM-5.1.

2 Spatio-Temporal Tool-Use Arena

In this section, we present STT-Arena, a benchmark for evaluating LLMs in tool-use environments that evolve autonomously over time and space. Each task includes spatio-temporal triggers that modify the environment state or tool availability when certain conditions are met, forcing the model to detect changes and replan accordingly. Each task is created using a three-stage approach.

2.1 Task Formulation

STT-Arena formalizes each instance as a tuple $\mathcal{T} = (\mathcal{E}, \Phi, u, q, \mathrm{CL})$, where $\mathcal{E}$ denotes the environment, which encompasses a set of states $\mathcal{S}$ and available tools $\mathcal{A}$; $\Phi = (\phi, c_\phi, e_\phi)$ is spatio-temporal triggers, each with a condition $c_\phi : \mathcal{S} \times \mathcal{X} \to \{0, 1\}$ depending on the state and the spatio-temporal

Table 1: Feature comparison of STT-Arena against existing agentic tool-use benchmarks across three dimensions: evaluation protocol, environment type, and Training Data.

Benchmark	Evaluation Protocol			Environment Type						Training Data	
	Tool Matching	State Alignment	LLM Judgement	No Env.	Static Env.	Realistic Time	Temporal	Spatial	Spatio Temporal	SFT	RL
API-Bank	✓	✓	✗	✓	✗	✗	✗	✗	✗	✓	✗
ToolBench	✗	✓	✗	✓	✗	✗	✗	✗	✗	✓	✗
StableToolBench	✗	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗
MCP-Bench	✓	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗
MCPToolBench++	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗
ToolTalk	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗
BFCL-v4	✓	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗
τ-Bench	✗	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗
τ^2-Bench	✗	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗
ToolSandBox	✓	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗
ACEBench	✓	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗
ToolAthlon	✗	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗
VitaBench	✗	✗	✓	✗	✓	✗	✗	✗	✗	✗	✗
TCP	✗	✓	✗	✗	✗	✓	✗	✓	✗	✗	✗
Timely-Eval	✗	✓	✓	✗	✗	✓	✗	✗	✗	✓	✓
GAIA-2	✓	✗	✓	✓	✗	✓	✗	✗	✗	✗	✗
RTR Gym	✗	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗
STT-Arena (Ours)	✗	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓

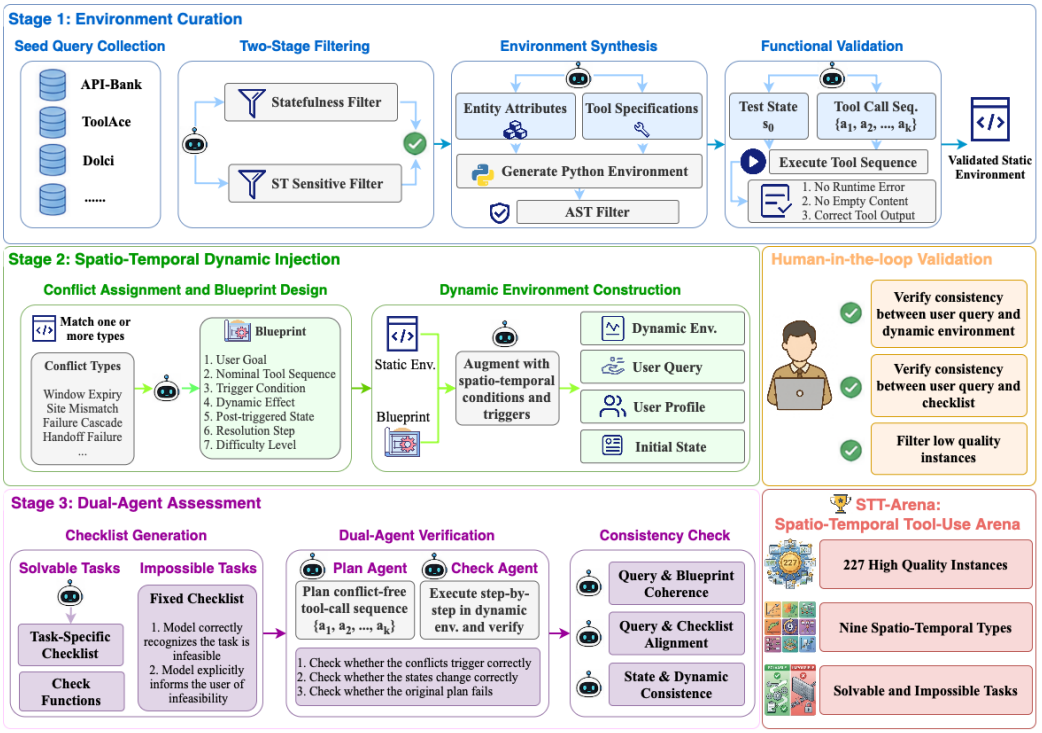

Figure 2: Overview of the STT-Arena construction pipeline. The pipeline consists of three stages: (1) Environment Curation, (2) Spatio-Temporal Dynamic Injection, and (3) Dual-Agent Assessment. Then we apply human-in-the-loop review to produce the final 227 benchmark instances.

context $\mathcal{X}$ and an effect $e_\phi : \mathcal{S} \to \mathcal{S}$ that modifies the state or tool availability when c_ϕ holds; u is the user profile encoding preferences and constraints; q is the user query; and CL is a checklist of necessary conditions for task success. The LLM receives q and issues a sequence of tool calls from $\mathcal{A}$. The model may also query a passive user simulator to clarify ambiguities or confirm state changes, but no additional information is provided beyond q and u. At any step, if the current state and context satisfy c_ϕ for some trigger $\phi \in \Phi$, the corresponding effect e_ϕ is applied autonomously. Such changes can invalidate the model's previous plan, forcing it to re-plan and adapt. After the model terminates, the final state is evaluated against CL to obtain the pass@1 rate of the task.

2.2 STT-ARENA CONSTRUCTION PIPELINE

To construct STT-Arena, we design a three-stage pipeline that systematically transforms real-world user requests into executable and rigorously validated spatio-temporal dynamic tasks. The pipeline begins by curating reliable static environments including entity states and tools. Then, it injects controlled spatio-temporal conflicts to create dynamic task instances with solvable and impossible categories across nine spatio-temporal types. Finally, we employ dual-agent assessment and human review to ensure that each instance is realistic, internally consistent, and evaluation-ready. The specific cases of each stage and step can be found in Appendix A.9.1.

2.2.1 STAGE 1: ENVIRONMENT CURATION.

The first stage, inspired by (Song et al., 2026), constructs a library $\mathcal{E}_{\text{static}}$ of validated, executable static environments that serve as the foundation for dynamic task generation.

Seed Query Collection and Filtering. We collect real-world user queries from three sources: API-Bank (Li et al., 2023), ToolAce (Liu et al., 2025), and Dolci (Olmo et al., 2025) to ensure that our benchmark is diverse and representative of real-world distributions. Each query passes through a two-stage filter using an LLM. The **statefulness filter** checks whether the query requires a persistent environment state across steps. The **spatio-temporal sensitivity filter** checks whether the outcome changes as a function of time or location. Only queries that pass both filters are retained. The detailed prompts can be found in Appendix A.10.2.

Environment Synthesis. For each qualifying query, we first prompt an LLM to infer the latent environmental information, including an environment summary and a detailed introduction. Based on this inferred information, we generate the corresponding entity attributes and tool specifications. The entity states and tool logic are then implemented separately as executable Python classes. Finally, we concatenate the entity attribute classes and the tool environment classes, and pass the combined code through an AST filter to eliminate unsafe or non-deterministic constructs, yielding a candidate environment $e \in \mathcal{E}_{\text{candidate}}$. The detailed prompts can be found in Appendix A.10.3.

Functional Validation. Each candidate environment e is validated by a tool-calling LLM. The LLM generates diverse test configurations, each consisting of an initial environment state s_0 and a sequence of tool calls $\{a_1, \ldots, a_k\}$. The environment executes this sequence; if any execution produces a runtime error, returns empty content, or otherwise fails to produce the expected tool outputs, the environment is discarded. Only environments that pass all test configurations without error are promoted to $\mathcal{E}_{\text{static}}$. The detailed prompts can be found in Appendix A.10.4.

2.2.2 STAGE 2: SPATIO-TEMPORAL DYNAMIC INJECTION

The second stage mainly construct spatio-temporal dynamic environments $\mathcal{E}_{\text{dynamic}}$ and tasks.

Conflict Assignment and Blueprint Design. For each $e \in \mathcal{E}_{\text{static}}$, we select one or more semantically compatible conflict types from a predefined set $\mathcal{C}$ of nine spatio-temporal dynamic types (Table 11). We then generate a blueprint B with difficulty level d (easy, medium, hard, and impossible) to generate conflict stories. Here, the blueprint acts as a generative contract that enforces internal consistency across all downstream modules, which includes: (1) User goal and user profile; (2) Nominal tool sequence $\{a_1, \ldots, a_m\}$ that succeeds in the absence of conflict; (3) Conflict trigger condition c_ϕ, which depends on the state and the spatio-temporal context $\mathcal{X}$; (4) Effect of conflict e_ϕ, which modifies the state or tool availability when condition c_ϕ satisfied; (5) Expected post-trigger state, characterizing the revised environmental states or tool availability that the model needs to recognize; (6) Required resolution steps that the model must execute to resolve the conflict and regain the objective. The detailed prompt templates can be found in Appendix A.11.1.

Dynamic Environment Construction. Given the blueprint B and a static environment $e \in \mathcal{E}_{\text{static}}$, our construction process involves three key steps: First, we prompt an LLM to augment e with the spatio-temporal conditions Φ specified in B, producing a dynamic environment e_{dynamic}. Second, we synthesize the user query q and user profile u based on the goals and constraints defined in B; Finally, we establish a realistic initial state $s_0 \in \mathcal{S}_e$ following the specified task plan in B. The detailed prompt templates can be found in Appendix A.11.2.

2.2.3 STAGE 3: DUAL-AGENT ASSESSMENT

The third stage focuses on validating the dynamic environments and tasks through dual-agent protocol.

Checklist Generation. For each dynamic instance, we generate an evaluation checklist CL based on the blueprint B and its difficulty level d. For feasible tasks (easy, medium, and hard), we prompt an LLM to produce a task-specific checklist $\mathrm{CL} = \{\mathrm{criterion}_1, \ldots, \mathrm{criterion}_p\}$ enumerating the necessary and sufficient conditions for task success, along with a set of rule-based check functions $\mathcal{F} = \{f_1, \ldots, f_p\}$, where each $f_j : \mathcal{S} \to \{0, 1\}$ evaluates the final environment state against each criterion. For the impossible category, the checklist is fixed and consists of two criteria: (i) the model correctly recognizes that the dynamic task cannot be completed given the current environment state, and (ii) the model explicitly informs the user simulator of this infeasibility. The detailed prompt templates can be found in Appendix A.12.1.

Dual-Agent Verification. To ensure data validity, each instance undergoes a dual-agent verification protocol. First, a planning agent formulates an original tool call sequence $\{a_1^{'} \ldots, a_m^{'}\}$ that assumes a conflict-free execution. Next, a checking agent executes this sequence in the dynamic environment e_{dynamic} , strictly verifying the process against three behavioral invariants: (i) the condition c_ϕ triggers exactly as scheduled in B; (ii) e_ϕ modifies the context or tool availability as specified; and (iii) the conflict successfully disrupts the original plan $\{a_1^{'} \ldots, a_m^{'}\}$, preventing goal achievement, e.g., a tool returns an error or an expected state change does not occur. Finally, spatio-temporal dynamic environments and tasks that pass this verification are curated for use as benchmark and training data. Please refer to Appendix A.12.2 for detailed prompt templates.

Consistency Check. Subsequent to dual-agent verification, we further perform a consistency check using LLMs and human annotations. Firstly, LLM-based auditor checks execution trajectory to verify that all artifacts are mutually coherent: the user query matches the blueprint, concrete mutations realize the claimed conflict semantics, the evaluation checklist covers key success and failure conditions, and the difficulty level aligns with the observed complexity. Prompt templates for this process can be found in Appendix A.12.3. Second, we manually filter candidate instances by validating the user query q, the checklist CL, and the alignment between the task description and environment dynamics. This human-in-the-loop validation results in the final STT-Arena, which consists of 227 high-quality instances. Detailed information and statistics are provided in Appendix A.8.2.

2.3 TASK EVALUATION

Each task is evaluated through interaction between the LLM and a passive user simulator. After the model terminates, the final state s_T is recorded. For feasible tasks (easy, medium, hard), each instance has a set of check functions $\mathcal{F} = \{f_1, \ldots, f_p\}$ where $f_j : \mathcal{S} \to \{0, 1\}$. The per-function outcome provides a fine-grained reward $R_{\text{fea}} = \frac{1}{p} \sum_{j=1}^{p} f_j(s_T)$. For impossible tasks, we use an LLM-as-a-judge with a fixed two-item checklist: (i) the model recognizes infeasibility, (ii) it communicates this to the user. The judge produces binary verdicts $v_1, v_2 \in 0, 1$ on the whole trajectory, yielding a binary reward $R_{\text{imp}} = \mathbf{1}[v_1 = 1 \wedge v_2 = 1]$.

3 EXPERIMENTS

3.1 EXPERIMENTAL SETUP

Compared LLMs. We benchmark the following LLMs on STT-Arena: (i) *Closed-Source LLMs*, including GPT-5.4, Gemini-3.1-Pro, CLaude-4.6-Opus, Claude-4.6-Sonnet, and Qwen-3.6-Plus; (ii) *Open-Source LLMs*, including GLM-5.1, GLM-5, Kimi-K2.5, MiniMax-M2.7, MiniMax-M2.5, Llama-3.3-70B, Qwen-3.5-397B-A17B, Deepseek-V3.2 and Deepseek-V4-Pro; (iii) *More Efficient LLMs*, including GPT-5.4-mini, Gemini-2.5-flash, Llama-3.1-8B, Qwen-3.5-9B, Qwen-3.5-35B-A3B, and Qwen-3-8B; (iv) *STT-Agent*, Qwen-3-4B (baseline) and STT-Agent with SFT and RL.

Evaluation Metrics. We adopt **Pass@1** as the primary evaluation metric. For the overall performance across solvable and impossible tasks, we compute a weighted average of the Pass@1 scores for each category, where the weight is proportional to the number of instances. Formally,

$$\text{Overall} = \alpha P_{\text{e}} + \beta P_{\text{m}} + \gamma P_{\text{h}} + \delta P_{\text{i}}, \tag{1}$$

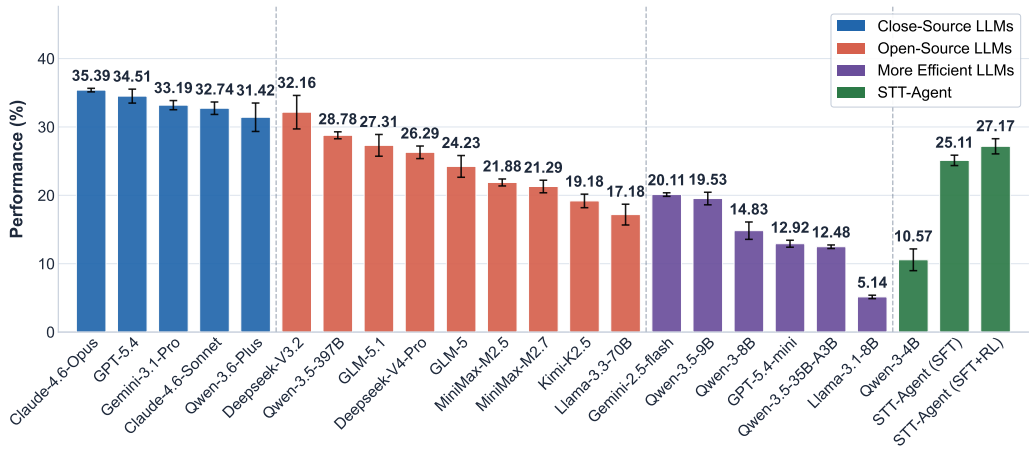

Figure 3: Overall Pass@1 performance of all evaluated models on STT-Arena. Results are grouped into four categories: closed-source LLMs, open-source LLMs, more efficient LLMs, and STT-Agent variants. Even the best-performing model, Claude-4.6-Opus, achieves only 35.39%, underscoring the fundamental difficulty of spatio-temporal dynamic reasoning. STT-Agent-4B, despite having only 4B parameters, outperforms many open-source frontier models. Detailed results can be found in Table 3.

where P_{e}, P_{m}, P_{h}, P_{i} denote the Pass@1 scores for each category (easy, medium, hard, impossible, respectively), and the weighting coefficients α, β, γ, δ are determined by sampling according to the corresponding number of instances in each level, with detailed values provided in Appendix A.8.2.

Evaluation Details. We set the maximum number of interaction turns to 50, perform three runs with a temperature of 0.7, and report the mean along with the standard deviation. We use Qwen-3.5-397B as the user simulator and the judgment model, with the temperature set to 0. Detailed system prompts used during evaluation can be found in the Appendix A.13.

3.2 Main Evaluation: STT-Arena Reveals Fundamental Gaps in Dynamic Reasoning

Figure 3 and Table 3 present the Pass@1 results across solvable and impossible tasks. Overall, current LLMs exhibit limited capabilities on STT-Arena, which can be summarized in three key observations:

Overall performance is limited, highlighting fundamental task difficulty. All evaluated models achieve limited performance on STT-Arena, with the best-performing model, Claude-4.6-Opus, reaching only 35.39% overall, highlighting the fundamental difficulty of spatio-temporal dynamic reasoning. Performance consistently degrades as task difficulty increases from Easy to Hard, with all models exhibiting notable drops at the Hard level, confirming that long-horizon replanning under intertwined spatio-temporal constraints remains an open challenge for current LLMs.

Closed-source models lead, while open-source models trail despite competitiveness. Among closed-source LLMs, the Claude and GPT series lead the rankings. Open-source models show competitive but consistently lower performance, with Deepseek-V3.2 (32.16%) being the strongest open-source contender yet still trailing the closed-source leaders by a non-trivial margin. This gap suggests that frontier closed-source models retain meaningful advantages in instruction following and adaptive decision-making under dynamic conditions.

Efficient LLMs perform substantially worse, underscoring the critical role of model scale. Efficient LLMs perform substantially worse than their frontier-scale counterparts. Models such as Llama-3.1-8B (5.14%), Qwen-3.5-35B-A3B (12.48%), and GPT-5.4-mini (12.92%) lag far behind, indicating that model scale plays a critical role in handling the complex replanning demands of STT-Arena, and that parameter-efficient architectures alone are insufficient to address spatio-temporal dynamic reasoning without targeted training.

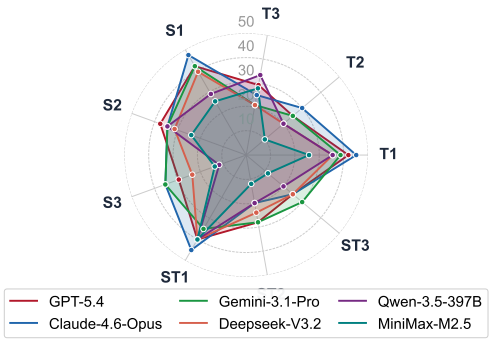

Figure 4: Pass@1 performance across the nine spatio-temporal conflict subtypes.

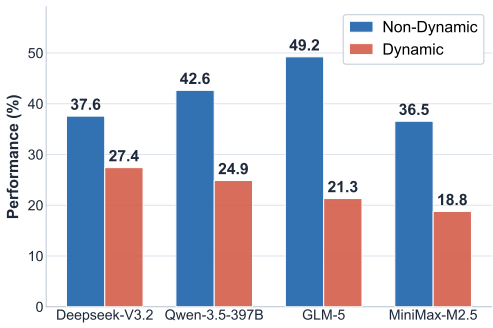

Figure 5: Performance gap between dynamic (STT-Arena) and non-dynamic environments.

3.3 Key Analyses: What Makes STT-Arena Hard?

To understand the origins of these performance gaps, we conduct fine-grained analyses along two dimensions, yielding the following insights:

Task structure governs failure modes. Figure 4 shows performance per conflict type defined in Table 11. Across all models, T1 (window expiry) and S1 (site mismatch) are consistently high, while T2 (priority reorder) and S3 (route restriction) are consistently low. This indicates that within pure temporal or spatial dynamics, tasks involving simple deadline tracking or location mismatch are manageable, but those requiring reordering of priorities or enforcement of route constraints pose universal difficulty. Strikingly, ST1 (resource shift) achieves stable and high performance across every model. In contrast, ST2 (failure cascade) and ST3 (handoff failure) drop sharply. This contrast reveals that spatio-temporal coupling is not inherently difficult: simple resource reallocation (ST1) is well handled, but once the coupling involves cascading dependencies (ST2) or misaligned handoffs across time and space (ST3), all current LLMs break down, exposing a fundamental blind spot in multi-step causal reasoning under intertwined dynamics.

Dynamics expose brittleness concealed in static evaluations. As shown in Figure 5, we compare model performance under a static benchmark constructed by removing spatio-temporal triggers and reconstructing the original checklist against STT-Arena. Across all four models, introducing dynamics consistently reduces performance, confirming that spatio-temporal evolution imposes a universal difficulty that current LLMs are not yet equipped to handle. More importantly, the relative ordering of models changes substantially between the two conditions, indicating that static evaluation alone is not sufficient for assessing robustness under realistic environmental shifts. These results suggest that high performance on conventional tool-use benchmarks may come at the cost of overfitting to fixed patterns, and that spatio-temporal stress testing is essential for assessing true deployment readiness.

3.4 Further Ablations: Probing Model Capabilities and Design Choices

We further investigate whether current limitations can be mitigated through algorithmic or architectural interventions, leading to three key findings:

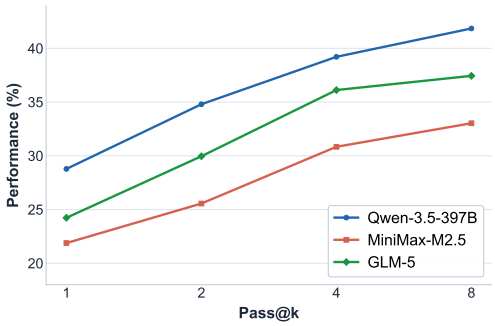

Figure 6: Test-time scaling via Pass@k rate.

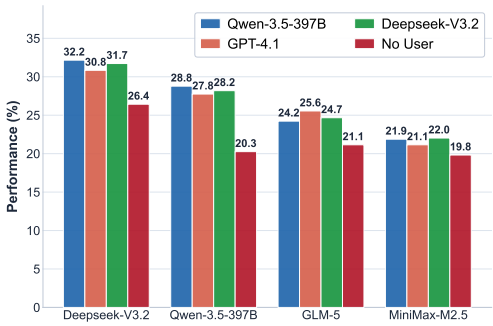

Figure 7: Ablation on the user simulators.

Test-time scaling partially mitigates uncertainty. We conduct test-time scaling via Pass@k in STT-Arena (Figure 6), where Pass@k means generating k independent attempts per task and considering the task successful if any single attempt succeeds, thus measuring the upper bound of each model when given more attempts under spatio-temporal dynamics. Across all three models, increasing k from 1 to 8 yields consistent and substantial gains, indicating that the inherent difficulty of dynamic tool use can be partially mitigated by broader sampling rather than relying solely on a single reasoning path. The gap between models narrows monotonically as k grows, implying that current limitations in handling spatio-temporal uncertainty are at least partly attributable to insufficient coverage of the solution space rather than fundamental architectural deficits. However, the performance gains gradually saturate as k increases to 8, with diminishing returns beyond moderate sampling. Even at Pass@8, the best model still falls short of 50% accuracy, underscoring that pure sampling alone cannot overcome the fundamental challenges posed by STT-Arena.

User simulators provide critical grounding. As shown in Figure 7, we compare three different user simulators (Qwen-3.5-397B, GPT-4.1, and Deepseek-V3.2) against a setting where the user simulator is completely removed from the evaluation loop (No User). In STT-Arena, spatio-temporal dynamics force LLMs to constantly re-plan and re-execute actions as the environment evolves. The user simulator provides information that helps models commit to better decisions and sustain task progression. When this guidance is absent, we observe a clear and consistent performance drop across all models, indicating that current LLMs lack a robust internal model of user intent and situational grounding. More importantly, without the additional information from a user simulator, LLMs become less confident during re-planning and frequently fall into local loops or repetitive failures, which severely hinders their ability to recover from dynamic shifts.

Reasoning helps but design matters. Figure 8 compares thinking and non-thinking modes across four models in STT-Arena. For Claude, Deepseek, and GLM, enabling thinking leads to substantial performance gains, confirming that explicit reasoning helps replan and adapt when the environment shifts. Qwen, however, shows nearly identical results between the two modes. This anomaly may arise because Qwen's thinking mode omits the final summary and only performs explicit reasoning, whereas its non-thinking mode includes a detailed summary that partially compensates for the lack of explicit reasoning. Consequently, the non-thinking mode in Qwen may inadvertently provide a form of structured guidance that mimics some benefits of reasoning, narrowing the gap. This counterexample reveals that the effectiveness of thinking modes depends not only on the presence of reasoning but also on how the model articulates and integrates its outputs.

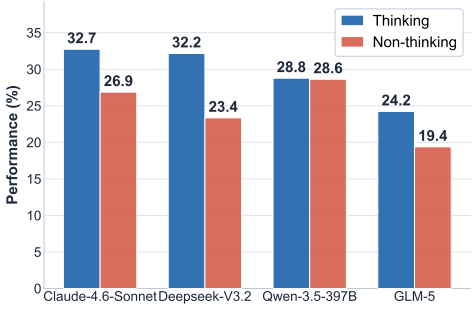

Figure 8: Effect of reasoning content on Pass@1 performance.

3.5 STT-Agent Results

We evaluate STT-Agent on STT-Arena and report results in Figure 3, Table 3, and Table 2. Implementation details are provided in Appendix A.8.4.

Table 2: Ablation results of STT-Agent, comparing the baseline Qwen-3-4B model, STT-Agent trained without trajectory refinement, and STT-Agent with refinement.

Models	Easy	Med.	Hard	Impossible	Overall	Avg. Calls
Qwen-3-4B (baseline)	18.31	9.46	2.82	10.00	10.57	7.63
STT-Agent (w/o refine)	28.17	16.92	11.86	47.01	23.10	32.70
STT-Agent	26.76	17.41	13.56	61.11	**25.11**	**15.30**

STT-Agent achieves strong performance despite its compact size. STT-Agent-4B achieves 27.17% overall Pass@1 on STT-Arena, outperforming many open-source frontier models with far more parameters. This result suggests that the performance gap observed in Section 3.2 stems not only from fundamental architectural limitations, but also from the absence of spatio-temporal dynamic reasoning in existing training pipelines.

Trajectory refinement is essential for both performance and efficiency. Trajectory refinement, discussed in detail in Section 4, plays a critical role in training quality. As shown in Table 2, STT-Agent trained on refined trajectories not only achieves higher overall Pass@1 (25.11% vs. 23.10%),

but also reduces average API calls substantially (15.30 vs. 32.70), indicating that training on refined trajectories leads to more decisive and efficient tool-use behavior with fewer redundant interactions.

4 DISCUSSION

4.1 FAILURE MODE ANALYSIS

We conduct a comparative analysis of successful and failed trajectory pairs on STT-Arena. The results reveal three recurring failure modes that distinguish robust LLMs from failing ones.

Stale-State Execution. A dominant failure mode is continuing to act on an outdated world state after the environment has already changed. LLMs persist with the pre-trigger plan and repeatedly invoke the same tools with similar arguments instead of first checking the environment state. This suggests that current LLMs overcommit to their initial reasoning trace and underutilize new observations returned by tools. In dynamic settings, valid actions depend on the latest state rather than earlier assumptions. Figure 19 illustrates a representative case where the LLM repeatedly retries an invalid route after a spatial blockage.

Misdiagnosis of Dynamic Triggers. Even when LLMs observe failures or abnormal tool outputs, they frequently misinterpret the underlying cause. For example, policy lockouts may be mistaken as parameter errors, missing identifiers as transient glitches, or hard infeasibility signals as recoverable obstacles. This indicates that LLMs treat tool feedback as surface-level content rather than evidence of deeper environmental transitions. Successful adaptation requires inferring why the state changed before deciding how to respond. Figure 20 presents an example where the agent mistakes a regulatory restriction for a technical failure and follows an incorrect recovery path.

Missing Post-Adaptation Verification. A third common pattern is that LLMs perform an adaptation step (e.g., rerouting, reassignment, or status update) but fail to verify whether the final state truly satisfies the updated constraints. They often stop once an intermediate action succeeds, even though capacity remains insufficient, dependencies are unresolved, or the task is only partially completed. This reveals a gap between action execution and outcome validation: tool success is incorrectly equated with task success. In spatio-temporal dynamic environments, adaptation is complete only when the resulting global state is feasible. Figure 21 shows a case where the LLM successfully reallocates resources but never checks that the full demand remains unmet.

4.2 ITERATIVE TRAJECTORY REFINEMENT

Motivated by these failure patterns, we propose an iterative trajectory refinement method that post-processes training trajectories. Even successfully solved trajectories often contain inefficient or fragile steps, such as blind retries after state changes, shallow misinterpretations of tool feedback, or premature termination without final verification. Our method cleans these trajectories by allowing an LLM to reorder, delete, or modify existing message blocks. Refinement proceeds in three sequential stages, each targeting one failure mode: first, **root-cause diagnosis after dynamic triggers**, preferring blocks that interpret the failure over blind retries; second, **state refresh before continuing execution**, enforcing re-sensing after spatio-temporal changes; third, **post-adaptation end-state verification**, ensuring that all constraints are satisfied before completion. As shown in Table 2, models trained on refined trajectories significantly outperform those trained on original trajectories, and the average number of tool-call rounds is substantially reduced, demonstrating that refined trajectories eliminate redundant steps and improve execution efficiency on STT-Arena.

5 CONCLUSION

We present STT-Arena, a benchmark of 227 tasks designed to evaluate the adaptive replanning capabilities of LLMs under spatio-temporal dynamics. Our evaluation of massive models demonstrates that spatio-temporal dynamics pose a fundamental and universal challenge, with even the strongest frontier model achieving only 35.39%. Through trajectory analysis, we identify three recurring failure modes that consistently separate robust models from failing ones, and propose an iterative trajectory refinement approach that targets each failure mode sequentially during training.

AI USE STATEMENT

In the preparation of this manuscript, we utilized artificial intelligence (AI) tools exclusively for language refinement, including grammar correction, proofreading, and stylistic polishing to enhance the overall readability of the text. These tools were strictly limited to linguistic assistance and were not employed for autonomous research, data analysis, literature generation, or the formulation of core scientific concepts and conclusions. We confirm that there was no over-reliance on AI throughout the research and writing process, and we take full responsibility for the originality, accuracy, and integrity of the final content.

REPRODUCIBILITY STATEMENT

We are committed to ensuring the full reproducibility of our benchmark. For the benchmark construction, we provide a comprehensive description in the main text and the supplementary materials (mainly for the codebase), enabling readers to reconstruct the benchmark from scratch. For the model evaluation, we detail the evaluation protocol, including prompt templates, decoding hyperparameters, inference configurations, and scoring metrics for all compared models; these details ensure that the reported results can be faithfully reproduced. We believe that the combination of the detailed descriptions in the main text, appendices, and the supplementary materials provides sufficient information to independently reproduce all results presented in this paper.

REFERENCES

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *CoRR*, abs/2506.07982, 2025. doi: 10.48550/ARXIV.2506.07982. URL https://doi.org/10.48550/arXiv.2506.07982.

Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Yuefeng Huang, Xiangcheng Liu, Xinzhi Wang, and Wu Liu. Acebench: A comprehensive evaluation of LLM tool usage. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025, Suzhou, China, November 4-9, 2025*, pp. 12970–12998. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.findings-emnlp.697/.

Zifeng Ding, Sikuan Yan, Moy Yuan, Xianglong Hu, Fangru Lin, and Andreas Vlachos. TCP: a benchmark for temporal constraint-based planning. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pp. 22452–22475. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.EMNLP-MAIN.1142. URL https://doi.org/10.18653/v1/2025.emnlp-main.1142.

Shiqing Fan, Xichen Ding, Liang Zhang, and Linjian Mo. Mcptoolbench++: A large scale AI agent model context protocol MCP tool use benchmark. *CoRR*, abs/2508.07575, 2025. doi: 10.48550/ARXIV.2508.07575. URL https://doi.org/10.48550/arXiv.2508.07575.

Nicholas Farn and Richard Shin. Tooltalk: Evaluating tool-usage in a conversational setting. *CoRR*, abs/2311.10775, 2023. doi: 10.48550/ARXIV.2311.10775. URL https://doi.org/10.48550/arXiv.2311.10775.

Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, Kunal Malkan, Dheeraj Mekala, Pierre Ménard, Gerard Moreno-Torres Bertran, Ulyana Piterbarg, Mikhail Plekhanov, Mathieu Rita, Andrey Rusakov, Vladislav Vorotilov, Mengjue Wang, Ian Yu, Amine Benhalloum, Grégoire Mialon, and Thomas Scialom. Gaia2: Benchmarking LLM agents on dynamic and asynchronous environments. *CoRR*, abs/2602.11964, 2026. doi: 10.48550/ARXIV.2602.11964. URL https://doi.org/10.48550/arXiv.2602.11964.

GLM. GLM-5: from vibe coding to agentic engineering. *CoRR*, abs/2602.15763, 2026. doi: 10.48550/ARXIV.2602.15763. URL https://doi.org/10.48550/arXiv.2602.15763.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, Findings of ACL, pp. 11143–11156. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.FINDINGS-ACL.664. URL https://doi.org/10.18653/v1/2024.findings-acl.664.

Wei He, Yueqing Sun, Hongyan Hao, Xueyuan Hao, Zhikang Xia, Qi Gu, Chengcheng Han, Dengchang Zhao, Hui Su, Kefeng Zhang, Man Gao, Xi Su, Xiaodong Cai, Xunliang Cai, Yu Yang, and Yunke Zhao. Vitabench: Benchmarking LLM agents with versatile interactive tasks in real-world applications. *CoRR*, abs/2509.26490, 2025. doi: 10.48550/ARXIV.2509.26490. URL https://doi.org/10.48550/arXiv.2509.26490.

Jian Hu, Jason Klein Liu, Haotian Xu, and Wei Shen. Reinforce++: Stabilizing critic-free policy optimization with global advantage normalization, 2025. URL https://arxiv.org/abs/2501.03262.

Junlong Li, Wenshuo Zhao, Jian Zhao, Weihao Zeng, Haoze Wu, Xiaochen Wang, Rui Ge, Yuxuan Cao, Yuzhen Huang, Wei Liu, Junteng Liu, Zhaochen Su, Yiyang Guo, Fan Zhou, Lueyang Zhang, Juan Michelini, Xingyao Wang, Xiang Yue, Shuyan Zhou, Graham Neubig, and Junxian He. The tool decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution. *CoRR*, abs/2510.25726, 2025. doi: 10.48550/ARXIV.2510.25726. URL https://doi.org/10.48550/arXiv.2510.25726.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pp. 3102–3116. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.EMNLP-MAIN.187. URL https://doi.org/10.18653/v1/2023.emnlp-main.187.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen. Toolace: Winning the points of LLM function calling. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=8EB8k6DdCU.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, Findings of ACL, pp. 1160–1183. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.FINDINGS-NAACL.65. URL https://doi.org/10.18653/v1/2025.findings-naacl.65.

Yichuan Ma, Linyang Li, Yongkang Chen, Peiji Li, Xiaozhe Li, Qipeng Guo, Dahua Lin, and Kai Chen. Timely machine: Awareness of time makes test-time scaling agentic. *CoRR*, abs/2601.16486, 2026. doi: 10.48550/ARXIV.2601.16486. URL https://doi.org/10.48550/arXiv.2601.16486.

Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, Robert Berry, Saumya Malik, Saurabh Shah, Scott Geng, Shane Arora, Shashank Gupta, Taira Anderson, Teng Xiao, Tyler Murray, Tyler Romero, Victoria Graf, Akari Asai, Akshita Bhagia, Alexander Wettig, Alisa Liu, Aman Rangapur, Chloe Anastasiades, Costa Huang, Dustin Schwenk, Harsh Trivedi, Ian Magnusson, Jaron Lochner, Jiacheng Liu, Lester

James V. Miranda, Maarten Sap, Malia Morgan, Michael Schmitz, Michal Guerquin, Michael Wilson, Regan Huff, Ronan Le Bras, Rui Xin, Rulin Shao, Sam Skjonsberg, Shannon Zejiang Shen, Shuyue Stella Li, Tucker Wilde, Valentina Pyatkin, Will Merrill, Yapei Chang, Yuling Gu, Zhiyuan Zeng, Ashish Sabharwal, Luke Zettlemoyer, Pang Wei Koh, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. Olmo 3, 2025. URL https://arxiv.org/abs/2512.13961.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): from tool use to agentic evaluation of large language models. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=dHng2O0Jjr.

Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Zhicheng Dou, and Ji-Rong Wen. Envscaler: Scaling tool-interactive environments for LLM agent via programmatic synthesis. *CoRR*, abs/2601.05808, 2026. doi: 10.48550/ARXIV.2601.05808. URL https://doi.org/10.48550/arXiv.2601.05808.

Zexu Sun, Bokai Ji, Hengyi Cai, Shuaiqiang Wang, Lei Wang, Guangxia Li, and Xu Chen. Agentskiller: Scaling generalist agent intelligence through semantically integrated cross-domain data synthesis. *CoRR*, abs/2602.09372, 2026. doi: 10.48550/ARXIV.2602.09372. URL https://doi.org/10.48550/arXiv.2602.09372.

Kimi Team. Kimi K2.5: visual agentic intelligence. *CoRR*, abs/2602.02276, 2026. doi: 10.48550/ARXIV.2602.02276. URL https://doi.org/10.48550/arXiv.2602.02276.

Ling Team and Inclusion AI. Every step evolves: Scaling reinforcement learning for trillion-scale thinking model. *CoRR*, abs/2510.18855, 2025. doi: 10.48550/ARXIV.2510.18855. URL https://doi.org/10.48550/arXiv.2510.18855.

Qwen Team. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025. doi: 10.48550/ARXIV.2505.09388. URL https://doi.org/10.48550/arXiv.2505.09388.

Weixun Wang, Shaopan Xiong, Gengru Chen, Wei Gao, Sheng Guo, Yancheng He, Ju Huang, Jiaheng Liu, Zhendong Li, Xiaoyang Li, et al. Reinforcement learning optimization for large-scale learning: An efficient and user-friendly scaling library. *arXiv preprint arXiv:2506.06122*, 2025a.

Zhenting Wang, Qi Chang, Hemani Patel, Shashank Biju, Cheng-En Wu, Quan Liu, Aolin Ding, Alireza Rezazadeh, Ankit Shah, Yujia Bao, and Eugene Siow. Mcp-bench: Benchmarking tool-using LLM agents with complex real-world tasks via MCP servers. *CoRR*, abs/2508.20453, 2025b. doi: 10.48550/ARXIV.2508.20453. URL https://doi.org/10.48550/arXiv.2508.20453.

Yule Wen, Yixin Ye, Yanzhe Zhang, Diyi Yang, and Hao Zhu. Real-time reasoning agents in evolving environments. *CoRR*, abs/2511.04898, 2025. doi: 10.48550/ARXIV.2511.04898. URL https://doi.org/10.48550/arXiv.2511.04898.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *CoRR*, abs/2406.12045, 2024. doi: 10.48550/ARXIV.2406.12045. URL https://doi.org/10.48550/arXiv.2406.12045.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand, 2024. Association for Computational Linguistics. URL http://arxiv.org/abs/2403.13372.

A Appendix

A.1 Related Work

No-Environment Benchmarks. Recent efforts evaluated agentic LLMs without a closed and self-contained environment, focusing purely on single-turn or multi-turn API calling accuracy. API-Bank (Li et al., 2023) and ToolBench (Qin et al., 2024) introduced plan-retrieve-call pipelines but treated tools as isolated functions without state or dependencies. StableToolBench (Guo et al., 2024) added a virtual API server to improve stability. ToolTalk (Farn & Shin, 2023) enabled multi-step tool execution through conversational interfaces, but relied on predefined trajectories that restrict agent autonomy. With the emergence of the Model Context Protocol (MCP), protocol-aligned benchmarks have become popular. MCP-Bench (Wang et al., 2025b) and MCPToolBench++ (Fan et al., 2025) scale to large numbers of servers and tools with fine-grained error taxonomies. All these benchmarks lack a closed, self-consistent environment for LLMs to interact with.

Static Environment Benchmarks. A second group of benchmarks features a closed, static environment where the world state evolves only when the agent calls and executes tools, with no time or location-driven changes. BFCL (Patil et al., 2025) extended evaluation to multi-turn dialogues but assembled conversations from fixed templates. τ-Bench (Yao et al., 2024) and τ^2-Bench (Barres et al., 2025) require LLMs to follow domain-specific rules while engaging with simulated users, yet focus on narrow scenarios. VitaBench (He et al., 2025) and ACEBench (Chen et al., 2025) provide diverse task scenarios but their dynamics remain limited to agent-driven transitions. ToolSandBox (Lu et al., 2025) pioneered stateful execution and tool interdependencies, while ToolAthlon (Li et al., 2025) offers a rich environment for tool use evaluation. All these benchmarks assume that the environment is static: no time-dependent changes, no spatial shifts, and no external triggers that alter the state without LLMs intervention.

Dynamic Environment Benchmarks. A few recent benchmarks have begun to address environmental dynamics. TCP (Ding et al., 2025) and Timely-Eval (Ma et al., 2026) focus on wall-clock time planning and reasoning, requiring LLMs to act under temporal constraints. GAIA-2 (Froger et al., 2026) introduces temporally evolving tasks where information becomes available or obsolete over time. Real-Time Reasoning Gym (RTR Gym) (Wen et al., 2025) evaluates how sensitive LLMs are to the passage of real-world time. However, these benchmarks emphasize gradual or predictable temporal changes (e.g., deadlines, streaming data) rather than abrupt state shifts that can occur at any step due to external triggers. Moreover, they do not require LLMs to detect, replan, and adapt in response to unexpected disruptions that involve both time and location.

In contrast to all of the above, STT-Arena systematically evaluates LLMs in spatiotemporally dynamic environments. The environment state can change abruptly at any step due to triggers activated by time, location, or their combination. These changes disrupt the original plan of models, forcing it to detect the shift, replan, and adapt. Table 1 reveal the detail comparisons among different benchmarks.

A.2 Limitations and Potential Society Impacts

Limitations. Although STT-Arena provides a rigorous and diverse benchmark for evaluating adaptive replanning under spatio-temporal dynamics, several limitations remain. First, the benchmark comprises 227 instances, which, while carefully curated through a three-stage pipeline with human-in-the-loop validation, may not exhaustively cover the full spectrum of real-world spatio-temporal conflict patterns. The nine conflict subtypes defined in our taxonomy represent a principled but potentially incomplete characterization of environmental dynamics encountered in deployment. Second, the construction pipeline relies heavily on Qwen-3.5-397B-A17B for environment synthesis, blueprint generation, and trajectory refinement, which may introduce systematic biases toward conflict patterns or linguistic styles that are more easily generated by this particular model family. Third, the evaluation protocol uses a fixed passive user simulator, which, although ablated across multiple backbone models, cannot fully replicate the diversity of real human communication behaviors. Finally, while STT-Agent-4B demonstrates that targeted training on spatio-temporal failure patterns is effective, the iterative trajectory refinement procedure introduces additional computational overhead, and its scalability to larger model families or more complex multi-agent settings has not been thoroughly investigated.

Potential Society Impacts. STT-Arena is intended to advance the development of LLM-based agents that are genuinely robust under the dynamic conditions of real-world deployment, with broad positive implications for safety-critical applications such as clinical consultation services, logistics coordination, and airline reservation systems. By exposing fundamental gaps in current models' ability to detect environmental shifts, replan, and verify task completion, this work encourages the community to prioritize reliability and graceful degradation over static benchmark performance, thereby reducing the risk of deploying brittle agents in high-stakes scenarios. On the negative side, improvements in adaptive replanning capabilities driven by benchmarks like STT-Arena could also lower the barrier for deploying autonomous agents in sensitive domains before sufficient alignment and safety guarantees are in place. Furthermore, the automated data synthesis pipeline, while designed for benchmark construction, could in principle be repurposed to generate adversarial environments that deliberately mislead or destabilize deployed agents. We encourage future work to pair advances in dynamic reasoning with corresponding progress in robustness certification and human oversight mechanisms.

Table 3: Full Pass@1 results (mean ± standard deviation over three runs) for all evaluated models on STT-Arena, broken down by solvable and impossible tasks. Performance degrades consistently as difficulty increases (Easy to Hard), with all models struggling most at the Hard level. STT-Agent-4B achieves xx% overall despite having only 4B parameters, surpassing many frontier models.

Models	**Easy**	**Medium**	**Hard**	**Impossible**	**Overall**
Closed-Source LLMs					
Qwen-3.6-Plus	35.21 (±2.82)	24.38 (±1.72)	23.16 (±2.59)	54.44 (±1.93)	31.42 (±2.08)
Claude-4.6-Sonnet	35.21 (±1.41)	29.85 (±1.49)	19.77 (±2.59)	58.89 (±1.92)	32.74 (±0.91)
Gemini-3.1-Pro	37.56 (±2.15)	30.35 (±2.28)	21.47 (±3.53)	52.22 (±1.92)	33.19 (±0.67)
GPT-5.4	39.91 (±0.81)	29.85 (±3.95)	24.29 (±0.98)	52.22 (±1.92)	34.51 (±1.02)
Claude-4.6-Opus	41.78 (±2.15)	31.34 (±2.59)	23.73 (±1.70)	52.22 (±3.85)	**35.39 (±0.26)**
Open-source LLMs					
Llama-3.3-70B	22.07 (±0.81)	11.44 (±2.28)	11.30 (±1.96)	30.00 (±3.33)	17.18 (±1.52)
Kimi-K2.5	25.35 (±2.82)	13.43 (±1.50)	14.12 (±4.26)	37.78 (±6.94)	19.18 (±0.99)
MiniMax-M2.7	26.29 (±2.94)	15.92 (±0.86)	13.56 (±1.70)	36.67 (±3.34)	21.29 (±0.92)
MiniMax-M2.5	26.76 (±2.44)	15.92 (±1.72)	12.43 (±3.53)	42.22 (±6.94)	21.88 (±0.51)
GLM-5	30.05 (±4.30)	21.39 (±1.73)	10.73 (±1.96)	43.33 (±3.34)	24.23 (±1.59)
Deepseek-V4-Pro	32.86 (±1.62)	20.40 (±2.28)	15.25 (±1.70)	45.56 (±8.39)	26.29 (±0.92)
GLM-5.1	35.21 (±1.41)	23.38 (±1.72)	15.25 (±1.70)	41.11 (±5.09)	27.31 (±1.59)
Qwen-3.5-397B-A17B	33.80 (±1.41)	21.89 (±3.11)	17.51 (±5.18)	54.45 (±3.85)	28.78 (±0.51)
Deepseek-V3.2	36.62 (±3.73)	27.86 (±2.28)	15.82 (±0.98)	63.33 (±5.77)	**32.16 (±2.45)**
More Efficient LLMs					
Llama-3.1-8B	7.98(±1.63)	3.98 (±0.86)	2.82 (±0.98)	5.56 (±1.93)	5.14 (±0.25)
Qwen-3.5-35B-A3B	19.72 (±1.41)	13.43 (±0.00)	7.34 (±0.98)	17.78 (±1.92)	12.48 (±0.26)
GPT-5.4-mini	19.72 (±1.41)	13.43 (±0.00)	7.34 (±0.98)	6.67 (±0.00)	12.92 (±0.51)
Qwen-3-8B	16.90 (±1.41)	13.43 (±1.50)	11.30 (±0.98)	20.00 (±3.33)	14.83 (±1.27)
Qwen-3.5-9B	27.23 (±0.81)	12.93 (±0.86)	11.30 (±0.98)	32.22 (±1.92)	19.53 (±0.92)
Gemini-2.5-flash	28.17 (±1.41)	13.43 (±1.50)	12.43 (±0.98)	31.11 (±1.92)	**20.11 (±0.26)**
STT-Agent					
Qwen-3-4B (baseline)	18.31 (±3.73)	9.46 (±0.86)	2.82 (±0.98)	10.00 (±3.33)	10.57 (±1.59)
STT-Agent-4B (SFT)	26.76 (±1.41)	17.41 (±0.86)	13.56 (±1.70)	61.11 (±1.92)	25.11 (±0.76)
STT-Agent-4B (SFT+RL)	29.11 (±1.63)	19.90 (±2.28)	14.12(±0.98)	64.44 (±1.93)	27.17 (±1.11)

A.3 Trigger Statistics

To better characterize the dynamic intensity of STT-Arena, we report supplementary statistics on trigger counts, trigger timing, and the magnitude of induced state changes. These statistics address concerns about whether the benchmark contains only a single disruption per task and whether the timing or intensity of triggers may confound the observed model performance.

A.3.1 TRIGGER COUNT DISTRIBUTION

Table 4 reports the number of triggers per task across all 227 benchmark instances.

Table 4: Distribution of trigger counts per task in STT-Arena.

Trigger Count	Number of Tasks	Percentage
1	209	92.1%
2	12	5.3%
3	6	2.6%

Approximately 92.1% of tasks contain a single mid-task trigger. The remaining tasks contain either two or three triggers, and these multi-trigger instances appear exclusively in the Hard difficulty category. This design choice ensures that the majority of tasks evaluate clean single-disruption replanning, while the most difficult subset additionally tests cascading or repeated recovery.

A.3.2 TRIGGER TIMING DISTRIBUTION

We divide each agent trajectory into two broad phases: **information gathering**, where the agent queries the environment to understand the initial state, and **task execution**, where the agent begins performing the main sequence of state-changing operations. Table 5 reports when triggers fire relative to these phases.

Table 5: Distribution of trigger timing relative to the execution phase.

Timing Phase	Number of Tasks	Percentage
Task Execution Phase	222	97.8%
Information Gathering Phase	5	2.2%

The overwhelming majority of triggers fire during the task execution phase. This means that the agent has usually already committed to an initial plan before the spatio-temporal disruption occurs. The small number of triggers occurring during information gathering appear only in the Easy category, where early detection is sufficient for recovery.

We further analyze trigger timing relative to the nominal tool sequence. Let m denote the number of tool calls in the nominal conflict-free plan, and let t denote the step at which the trigger fires. We define the relative trigger position as

$$r = \frac{t}{m}.$$

Table 6 reports the distribution of r.

Table 6: Distribution of relative trigger position $r = t/m$ within the nominal tool sequence.

Relative Position Range	Number of Tasks	Percentage
Early ($r \leq 0.33$)	41	18.1%
Middle ($0.33 < r \leq 0.66$)	112	49.3%
Late ($r > 0.66$)	74	32.6%

Most triggers occur in the middle portion of the nominal plan. This ensures that the agent has already made meaningful progress before the conflict appears. Late triggers are more common in Hard tasks, where the agent may need to abandon substantial completed work and reconstruct a global plan.

A.4 STAGE-LEVEL EVALUATION METRICS

Pass@1 measures whether a task is eventually solved, but it does not distinguish where the agent fails during adaptive replanning. To provide a more fine-grained diagnosis, we introduce four stage-level metrics.

A.4.1 METRIC DEFINITIONS

Trigger Detection Rate. The proportion of tasks in which the agent's execution path naturally reaches and activates the injected trigger. This metric measures whether the model follows a sufficiently correct execution path to encounter the designed conflict.

Recovery Length. The average number of tool-call steps taken from the moment the trigger fires until the agent resumes productive task execution or terminates. Lower values indicate more efficient recovery.

Diagnosis Rate. The proportion of triggered tasks in which the agent explicitly identifies the dynamic change, for example by querying the affected state, noting the discrepancy, or adjusting its reasoning about the cause of failure.

Adaptive Replan vs. Brute-force Retry. A classification of post-trigger behavior. "Adaptive Replan" indicates that the agent changes its strategy, such as selecting different tools, querying updated state, or targeting alternative entities. "Brute-force Retry" indicates that the agent repeats the same or superficially varied actions without addressing the underlying state change.

We use GPT-OSS-120B as an automatic trajectory annotator to compute these metrics. The annotator inspects the full execution trace and labels whether the trigger was reached, whether the agent diagnosed the change, and whether the post-trigger behavior reflects adaptive replanning or brute-force retry.

A.4.2 STAGE-LEVEL RESULTS

Table 7 reports the stage-level metrics for four representative open-source models. For reference, we also report the human baseline, which is described in Appendix **??**.

Table 7: Stage-level evaluation metrics on STT-Arena. Adaptive Replan / Brute-force Retry reports the number of analyzed triggered trajectories assigned to each category.

Model	Trigger Detection Rate (%)	Recovery Length	Diagnosis Rate (%)	Adaptive / Retry	Pass@1 (%)
Deepseek-V3.2	78.85	5.2	87.64	147 / 31	32.16
Qwen-3.5-397B	71.37	5.4	80.25	128 / 34	28.78
GLM-5	73.57	5.7	76.65	110 / 57	24.23
MiniMax-M2.5	63.44	6.1	70.83	87 / 57	21.88
Human Baseline	88.00	4.4	90.90	39 / 5	56.00

Several observations emerge from Table 7. First, trigger detection rate is positively correlated with overall Pass@1. Models that more reliably follow a correct execution path are also more likely to solve the task after the trigger fires. Second, recovery length is inversely correlated with performance: stronger models recover in fewer steps, while weaker models spend more steps on redundant exploration or repeated failures. Third, diagnosis rate separates more robust models from brittle ones. Models that correctly identify the cause of disruption are more likely to choose a valid recovery action. Finally, the ratio of adaptive replanning to brute-force retry is a strong indicator of robustness. Stronger models tend to revise their strategy after a trigger, whereas weaker models often repeat invalid actions.

A.5 GENERATOR MODEL ABLATION

A potential concern is that STT-Arena relies heavily on Qwen-3.5-397B during environment synthesis, blueprint generation, and task construction. To examine whether this introduces a systematic preference bias, we conducted an additional ablation using alternative generator models.

A.5.1 SETUP

We used Deepseek-V3.2, GLM-5, and MiniMax-M2.5 to synthesize 50 additional task instances each. These instances were generated from Stage 2 of the pipeline onward, using the same Stage 1 static environments as the original benchmark. We matched the difficulty and conflict-type distributions of these additional sets to the original STT-Arena as closely as possible.

These three additional sets were produced for ablation purposes and were not subjected to the full human-in-the-loop validation applied to the final 227 benchmark instances.

A.5.2 RESULTS

Table 8 reports the Pass@1 performance of four representative evaluated models on the original benchmark and on the three additional generator sets.

Table 8: Pass@1 (%) across different generator-model sets. Each additional set contains 50 tasks and is evaluated with a single run.

Evaluated Model	Origin STT-Arena	Deepseek-V3.2 Generated	GLM-5 Generated	MiniMax-M2.5 Generated
Deepseek-V3.2	32.16	30.00	24.00	36.00
Qwen-3.5-397B	28.78	24.00	20.00	30.00
GLM-5	24.23	20.00	18.00	24.00
MiniMax-M2.5	21.88	16.00	12.00	22.00

Although absolute Pass@1 values fluctuate across generator sets, the relative ranking of the evaluated models remains consistent. Across all four sets, Deepseek-V3.2 outperforms Qwen-3.5-397B, which outperforms GLM-5, which in turn outperforms MiniMax-M2.5. This indicates that the benchmark does not systematically favor the generator model family or any specific evaluated model.

We also emphasize that the executable environments in STT-Arena are implemented as deterministic Python state machines. The generator model designs the conflict semantics, narrative, and task structure, but the runtime environment behavior is determined by code and deterministic state transitions rather than by an LLM during evaluation.

A.6 JUDGE MODEL ABLATION

For Impossible tasks, evaluation uses an LLM-as-a-judge protocol to determine whether the agent recognized task infeasibility and communicated it to the user. To verify that the judge model does not bias the results, we evaluated three representative models on the Impossible subset using three different judge models.

Table 9 reports the results.

Table 9: Impossible-set Pass@1 (%) under different judge models.

Evaluated Model	Judge: Qwen-3.5-397B	Judge: GLM-5	Judge: MiniMax-M2.5
Deepseek-V3.2	63.33	63.33	60.00
Qwen-3.5-397B	54.45	52.22	52.22
GLM-5	43.33	43.33	41.11

The maximum variation across judge models is small and within the range of evaluation variance observed across repeated runs. This indicates that the choice of judge model does not materially affect the evaluation of Impossible tasks.

A.7 TRANSFERABILITY RESULTS ON BFCL V4

To examine whether training on STT-Arena generalizes beyond our benchmark, we evaluate STT-Agent-4B on the BFCL v4 multi-turn tool-use benchmark. We compare against two 4B-scale baselines trained for tool-interactive environments.

Table 10 reports the results.

STT-Agent-4B achieves competitive performance on BFCL v4, surpassing EnvScaler-4B and approaching EnvFactory-4B. This suggests that training on spatio-temporal dynamic tasks improves general tool-use and multi-turn interaction ability, rather than merely overfitting to STT-Arena-specific patterns.

Table 10: Performance on the BFCL v4 multi-turn benchmark.

Model	Base	Miss Func	Miss Param	Long Context	Overall
EnvScaler-4B	53.5	48.5	35.0	37.0	43.5
EnvFactory-4B	59.5	56.0	36.5	38.0	47.5
STT-Agent-4B	54.0	52.5	35.0	39.0	45.1

A.8 Detailed Information

A.8.1 Detailed information of Main Results

As shown in Table 3, we report the detailed results of Figure 3.

A.8.2 Detailed Information of STT-Arena

Construction Details. During the construction of STT-Arena, we utilize Qwen-3.5-397B to synthesize all benchmark data. After the automated pipeline, three annotators conduct a final verification of the produced benchmark instances, covering the following aspects: (1) consistency between the user query and the checklist, (2) correctness of the checklist, and (3) plausibility of the spatio-temporal dynamics. After manual validation, we obtain 227 final instances, spanning two categories (solvable and impossible) and nine spatio-temporal subtypes.

Statistic of STT-Arena. As shown in Figures 9 and 10, we report the distribution of data samples across the solvable and impossible tasks and the nine spatio-temporal subtypes. Detailed information and examples for each difficulty level and each subtype are provided in Tables 12 and 11, respectively.

Evaluation Details of STT-Arena. We utilize Pass@1 rate for evaluating STT-Arena and we calculate the overall performance through a weighted average of the four levels. Formally, Overall $= \alpha P_{\mathrm{e}} + \beta P_{\mathrm{m}} + \gamma P_{\mathrm{h}} + \delta P_{\mathrm{i}}$, where α, β, γ, and δ are equal to $71/227$, $67/227$, $59/227$, and $30/227$, respectively.

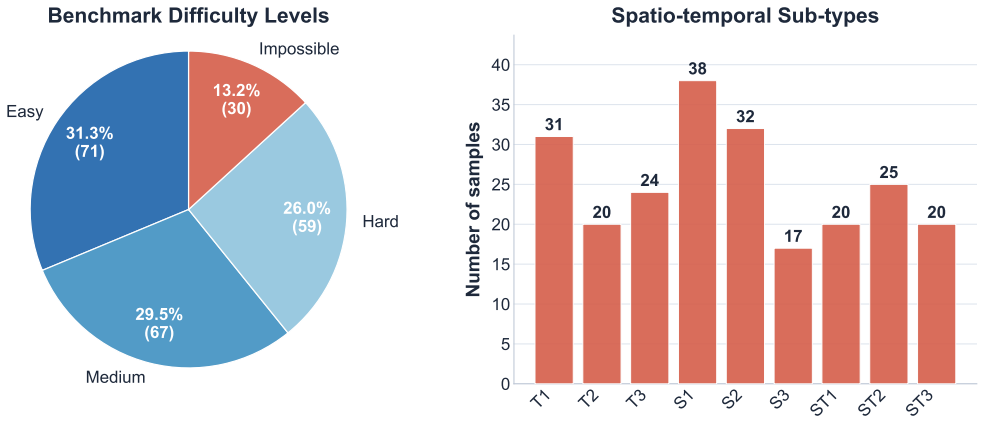

Figure 9: Distribution of STT-Arena instances across difficulty levels and spatio-temporal subtypes. (Left) The benchmark comprises 71 Easy (31.3%), 67 Medium (29.5%), 59 Hard (26.0%), and 30 Impossible (13.2%) instances. (Right) Instance counts per spatio-temporal subtype, with S1 being the most frequent (38 instances) and S3 the least (17 instances).

A.8.3 Detailed Information of Training Data

We generate SFT trajectories and RL tasks through our three-stage pipeline (same as the construction of STT-Arena, we also use Qwen-3.5-397B-A17B to generate training data). Specifically, as shown in Figures 11 and 12, we construct 2,212 validated trajectories for SFT and 8,119 instances for online RL, though we note that not all RL instances are used in RL training.

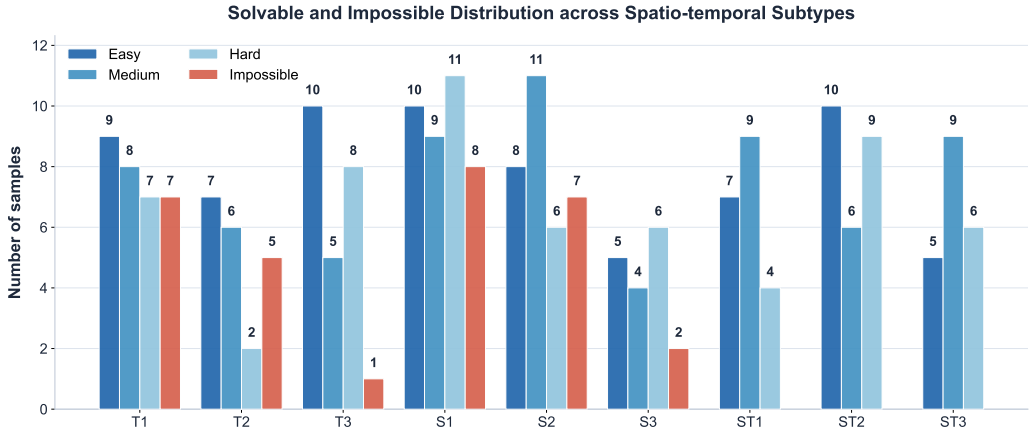

Figure 10: Solvable and impossible instance distribution across the nine spatio-temporal subtypes.

Table 11: Taxonomy of the nine spatio-temporal conflict types in STT-Arena, organized into three categories: Temporal (T1–T3), Spatial (S1–S3), and Spatio-Temporal (ST1–ST3). Each conflict type is defined by a distinct disruption mechanism and illustrated with a concrete real-world example, collectively covering the principal ways in which real-world conditions can invalidate an ongoing agent plan.

Category	ID	Conflict Type	Examples
Temporal	T1	Window Expiry	Hotel booking timeout during multi-step planning.
	T2	Priority Reorder	Flight rescheduled earlier, invalidating transfers.
	T3	Quota Reset	Promotional tickets expire at midnight mid-session.
Spatial	S1	Site Mismatch	Reserved vehicle relocated to a different depot.
	S2	Dependency Block	Warehouse lockdown halts all downstream dispatch.
	S3	Route Restriction	New customs rules block a cross-border corridor.
Spatio-temp.	ST1	Resource Shift	Peak demand moves cars away from residential zones.
	ST2	Failure Cascade	Hub outage disrupts regional warehouse inventory.
	ST3	Handoff Failure	Medical sample misses window due to clock drift.

Table 12: Description of the solvable and impossible tasks in STT-Arena. Levels range from Easy (single isolated conflict, one corrective action) to Hard (long-horizon cascading constraints requiring global replanning) and Impossible (no valid completion path exists; the correct response is to recognize and communicate infeasibility).

Level	Description
Easy	Tasks involve isolated, immediately observable conflicts with no cascading effects. Recovery requires at most one corrective action, with limited state dependency across tool calls.
Medium	Multi-step tasks with mild state dependency. Conflicts may be deferred and require awareness of prior context; recovery involves replanning over 2–3 steps.
Hard	Long-horizon tasks with interleaved spatiotemporal constraints. Conflicts may cascade, requiring the agent to detect implicit failures and replan globally across the full execution trajectory.
Impossible	Tasks for which no valid completion path exists given the injected conflict. The correct behavior is to identify infeasibility and communicate it to the user, rather than attempting resolution.

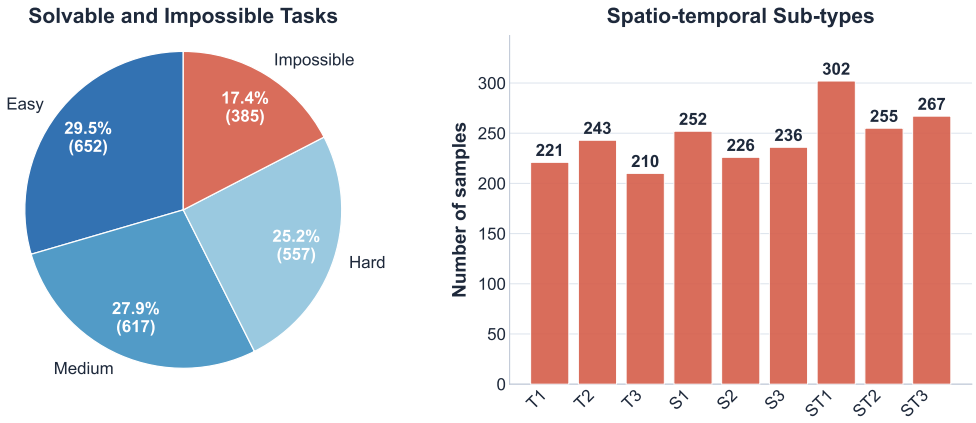

Figure 11: Distribution of SFT trajectories across difficulty levels and spatio-temporal subtypes.

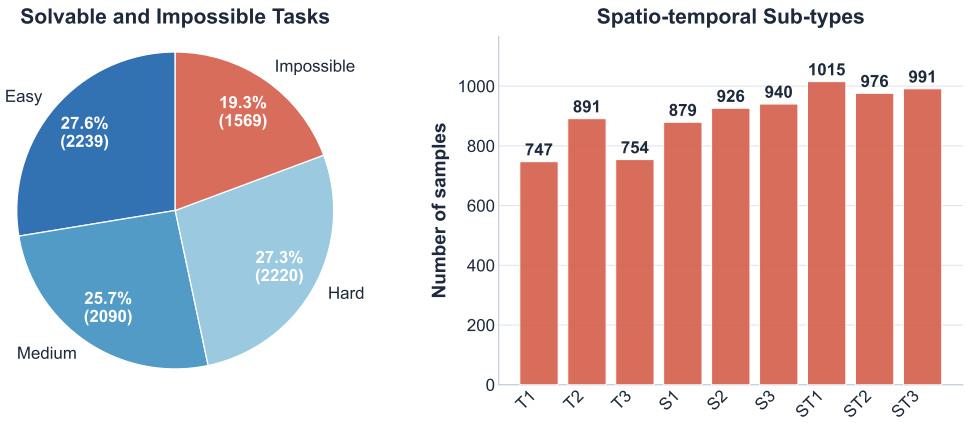

Figure 12: Distribution of RL tasks across difficulty levels and spatio-temporal subtypes.

A.8.4 Implementation Details of SFT and RL

SFT. We use Qwen-3-4B-Base as the backbone model and train with the LlamaFactory (Zheng et al., 2024) framework for 2 epochs. We adopt a cosine learning rate scheduler with a learning rate of 1.0×10^{-5} and a warmup ratio of 0.03. The maximum sequence length is set to 32K tokens, and the effective global batch size is 128.

RL. We further fine-tune the SFT checkpoint using the REINFORCE++ (Hu et al., 2025) algorithm within the ROLL (Wang et al., 2025a) framework. We retain a KL constraint with a coefficient of 0.1 and use a learning rate of 1.0×10^{-6}. In each training step, we sample 32 tasks and rollout 4 trajectories per task, for a total of up to 100 training steps. The maximum trajectory length is set to 32K tokens, and the maximum generation length per step is capped at 4K tokens.

Computational Resources. All the training experiments including SFT and online RL are conducted on 4 NVIDIA H200 GPUs.

A.9 Case Examples

A.9.1 Cases of STT-Arena Construction Pipeline

In this section, we provide some examples during STT-Arena construction pipeline including static environment, blueprint, user profile, checklist, check functions, and dynamic environment as shown in Figures 13, 14, 15, 16, 17, and 18.

Example of Static Environment

```
class ColdChainLogisticsEnv:
    def __init__(self, init_config: dict):
        self.distribution_hubs = init_config.get("distribution_hubs", {})
        self.depots = init_config.get("depots", {})
        self.vehicles = init_config.get("vehicles", {})
        self.cooling_units = init_config.get("cooling_units", {})
        self.cooling_unit_resets = init_config.get("cooling_unit_resets", {})
        self.shipments = init_config.get("shipments", {})
        self.goods_inventory = init_config.get("goods_inventory", {})
        self.route_stops = init_config.get("route_stops", {})
        self.clinics = init_config.get("clinics", {})

    # --------- INFORMATION QUERY OPERATIONS ---------

    def get_vehicle_status(self, vehicle_id: str): ...
    def list_depot_maintenance_schedule(self, depot_id: str,
    time_range: dict): ...
    def get_shipment_journey_status(self, shipment_id: str): ...
    def get_route_stop_details(self, vehicle_id: str): ...
    def get_goods_inventory_at_site(self, site_id: str): ...
    def list_clinic_immunization_schedule(self, site_id: str): ...
    def list_operational_hours(self, site_id: str): ...
    def get_cooling_unit_status(self, cooling_unit_id: str): ...

    # ---------ADMINISTRATIVE / STATE-CHANGE OPERATIONS ---------

    def assign_shipment_to_vehicle(self, shipment_id: str,
    vehicle_id: str): ...
    def record_cooling_unit_reset(self, reset_window_id: str, action: str,
                                  timestamp: str): ...
    def update_vehicle_status(self, vehicle_id: str, status: str,
                              current_site_id: str): ...
    def schedule_cooling_unit_reset(self, cooling_unit_id: str, depot_id: str,
                                    requested_window: dict): ...
    def add_or_update_route_plan(self, vehicle_id: str, route_plan: list): ...
    def update_route_stop_actual_times(self, route_stop_id: str,
                                       actual_arrival: str,
```

```
                                          actual_departure: str): ...
    def register_goods_receipt(self, shipment_id: str, site_id: str,
                               timestamp: str): ...
    def update_goods_inventory_status(self, goods_id: str, status: str,
                                      current_location_id: str): ...
    def create_new_shipment(self, goods_id: str, origin_hub_id: str,
                            destination_hub_id: str, departure_time: str): ...
```

Figure 13: Example of static environment, we implement the environment through Python class.

Example of Blueprint


```
[
  {
    "scenario_id": "ColdChainLogisticsEnv__T2__medium",
    "env_id": "ColdChainLogisticsEnv",
    "conflict_type": "T2",
    "difficulty": "medium",
    "user_goal": "Ship a batch of insulin (goods_id='INS-2024-001')
    from distribution hub 'HUB-NORTH' to clinic 'CLINIC-DOWNTOWN'
    with a required arrival time before 2024-08-15T08:00:00.
    The insulin must never exceed 8°C.",
    "normal_flow": [
      {
        "operation": "create_new_shipment",
        "purpose": "Create a new shipment record for the insulin batch."
      },
      {
        "operation": "list_operational_hours",
        "purpose": "Check clinic's receiving hours to schedule delivery."
      },
      {
        "operation": "assign_shipment_to_vehicle",
        "purpose": "Assign the shipment to a refrigerated vehicle."
      },
      {
        "operation": "add_or_update_route_plan",
        "purpose": "Define route stops
        including the clinic as final destination."
      },
      {
        "operation": "update_vehicle_status",
        "purpose": "Mark vehicle as 'in_transit' when dispatched."
      },
      {
        "operation": "register_goods_receipt",
        "purpose": "Confirm arrival and receipt at the clinic."
      }
    ],
    "conflict_design": {
      "activation_operation": "list_operational_hours",
      "trigger_mechanism": "conditional_guarded",
      "trigger_condition": "The vehicle has already been dispatched
      (status='in_transit') but the clinic's operating hours
      for the planned delivery day are overridden
      by an emergency closure order that was issued after dispatch.",
      "recovery_guard_condition": "The agent successfully cancels
      the current delivery assignment and reassigns
      the shipment to a different vehicle with a new route
      that arrives before the insulin expiry deadline.",
      "mutations": [
        {
```

```
            "state_path": "clinics['CLINIC-DOWNTOWN'].operational_hours",
            "change_description": "original: {'Monday-Friday': '08:00-17:00'}
            → new: {'Monday-Friday': '08:00-12:00'}"
          },
          {
            "state_path": "shipments['SHIP-INS-001'].delivery_window.end",
            "change_description": "original: '2024-08-15T17:00:00'
            → new: '2024-08-15T10:00:00' (because the clinic now closes at noon)"
          }
        ],
        "observable_via": "list_operational_hours"
      },
      "recovery_path": "1. `update_vehicle_status` |
      Change vehicle status back to 'idle' and clear current assignment.
      2. `update_goods_inventory_status` |
      Mark the insulin shipment as 'pending_reassignment'.
      3. `list_operational_hours` |
      Verify new clinic hours and identify alternative delivery window.
      4. `create_new_shipment` |
      Create a new shipment with accelerated delivery window.
      5. `assign_shipment_to_vehicle` |
      Assign to a different vehicle (with faster cooling unit).
      6. `add_or_update_route_plan` |
      Plan direct route without intermediate stops.
      7. `update_vehicle_status` |
      Dispatch new vehicle.
      8. `register_goods_receipt` |
      Complete delivery before 10:00.",
      "_raw": "## User Goal\nShip a batch of insulin ...
      ## Normal Flow\n1. `create_new_shipment` ...\n..."
    }
  ]
```

Figure 14: Example of blueprint, we design the blueprint based on the conflict types, difficulty levels, and environment information.

Example of User Profile

```
[
  {
    "scenario_id": "ColdChainLogisticsEnv__T2__medium",
    "user_profile": {
      "task_goal": "Ensure the insulin batch arrives at
      CLINIC-DOWNTOWN before 10:00.",
      "persona": "I am a logistics coordinator at the central hub,
      responsible for temperature-sensitive deliveries.",
      "context": "This shipment is critical for pediatric patients.
      The clinic's hours changed due to an emergency.",
      "tone": "Professional but slightly anxious about the deadline;
      responds concisely with relevant data.",
      "fallback_response": "I'm not sure what you mean.
      Please check the shipment ID and try again.",
      "decline_cancel_response": "We cannot cancel this shipment
      without a formal override.
      Please find a way.",
      "frustration_threshold": "medium",
      "flexibility": "high",
      "known_facts": [
        "The shipment ID is SHIP-INS-001.",
        "Clinic normally closes at 17:00 but today closes at 10:00.",
        "Vehicle V002 has a faster cooling unit than V001."
      ],
```

```
      "withhold_until_asked": [
        "The emergency closure order was issued after
        the first vehicle was dispatched.",
        "There is a secondary receiving bay that
        stays open until 12:00 but only for small packages."
      ],
      "fidelity_rules": [
        "Do not invent new vehicle IDs or shipment IDs
        beyond those in the scenario.",
        "Do not assume the clinic can extend hours."
      ],
      "helpfulness_rules": [
        "If the agent asks for operating hours,
        provide the current (mutated) hours.",
        "If the agent asks why the deadline changed,
        state the emergency closure but only after being asked."
      ],
      "persona_name": "I am a logistics coordinator at the central hub,
      responsible for temperature-sensitive deliveries.",
      "persona_background": "This shipment is critical for pediatric patients.
      The clinic's hours changed due to an emergency.",
      "communication_style": "Professional but slightly anxious
      about the deadline;
      responds concisely with relevant data.",
      "key_preferences": [
        "The shipment ID is SHIP-INS-001.",
        "Clinic normally closes at 17:00 but today closes at 10:00.",
        "Vehicle V002 has a faster cooling unit than V001."
      ],
      "clarification_responses": {
        "What is the deadline?": "The insulin must arrive before 10:00 today.",
        "Can we use a different clinic?":
        "No, it must be CLINIC-DOWNTOWN for the specific pediatric patients.",
        "What vehicles are available?":
        "V001 and V002 are idle now, but V002 is faster."
      }
    }
  }
]
```

Figure 15: Example of user profile, our user simulator is configured by the profile and interact with the tested LLMs.

Example of Checklist

```
Has the shipment record for the insulin batch been created?
Has the shipment been assigned to a refrigerated vehicle?
Has the route plan included the destination clinic?
Has the goods receipt been registered at the clinic before
the deadline (2024-08-15 10:00)?
Has the final delivery respected the mutated clinic
operating hours?
```

Figure 16: Example of checklist which is the evaluation mechanism of our tasks.

Example of Check Functions

```
def check_shipment_exists(env):
    shipments = getattr(env, 'shipments', {})
    for s in shipments.values():
```

```
        if s.get('goods_id') == 'INS-2024-001':
            return True, "Shipment record found."
    return False, "Shipment record missing."
```

Figure 17: Example Python function that validates one checklist item.

Example of Dynamic Environment

```
class ColdChainLogisticsEnv:
    def __init__(self, init_config: dict):
        # Top-level state mapping
        self.distribution_hubs =
        init_config.get("distribution_hubs", {})
        self.depots = init_config.get("depots", {})
        self.vehicles = init_config.get("vehicles", {})
        self.cooling_units =
        init_config.get("cooling_units", {})
        self.cooling_unit_resets =
        init_config.get("cooling_unit_resets", {})
        self.shipments = init_config.get("shipments", {})
        self.goods_inventory =
        init_config.get("goods_inventory", {})
        self.route_stops = init_config.get("route_stops", {})
        self.clinics = init_config.get("clinics", {})
        # Internal tracking for conflict triggering
        self._conflict_armed = False
        self._conflict_fired = False
        self._conflict_vehicle_id = "VH001"
        self._conflict_reset_window_id = "RST1001"
        self._conflict_depot_id = "DPT010"
        self._conflict_cooling_unit_id = "CU1001"
        self._operation_log = []

    # ------ INFORMATION QUERY OPERATIONS ------

    def get_vehicle_status(self, vehicle_id: str): ...
    def list_depot_maintenance_schedule(
        self, depot_id: str, time_range: dict
    ): ...
    def get_shipment_journey_status(self, shipment_id: str): ...
    def get_route_stop_details(self, vehicle_id: str): ...
    def get_goods_inventory_at_site(self, site_id: str): ...
    def list_clinic_immunization_schedule(self, site_id: str): ...
    def list_operational_hours(self, site_id: str): ...
    def get_cooling_unit_status(self, cooling_unit_id: str): ...

    # ------ ADMINISTRATIVE / STATE-CHANGE OPERATIONS ------

    def assign_shipment_to_vehicle(
        self, shipment_id: str, vehicle_id: str
    ): ...
    def record_cooling_unit_reset(
        self, reset_window_id: str, action: str, timestamp: str
    ): ...
    def update_vehicle_status(
        self, vehicle_id: str, status: str, current_site_id: str
    ): ...
    def schedule_cooling_unit_reset(
        self, cooling_unit_id: str,
        depot_id: str,
        requested_window: dict
```

```
    ): ...
    def add_or_update_route_plan(
        self, vehicle_id: str, route_plan: list
    ): ...
    def update_route_stop_actual_times(
        self, route_stop_id: str, actual_arrival: str,
        actual_departure: str
    ): ...
    def register_goods_receipt(
        self, shipment_id: str, site_id: str, timestamp: str
    ): ...
    def update_goods_inventory_status(
        self, goods_id: str, status: str,
        current_location_id: str
    ): ...
    def create_new_shipment(
        self, goods_id: str, origin_hub_id: str,
        destination_hub_id: str,
        departure_time: str
    ): ...
```

Figure 18: Example of dynamic environment which injects the spatio-temporal triggers into the static one.

A.9.2 Cases of Failure Mode

As shown in Figures 19, 20, and 21, we propose some cases about the failure modes in STT-Arena.

Failure Mode Case: Stale-State Execution

Task: Ensure that irrigation for Field Segment C begins at 07:00 with valve sequence V1, V2, V3, targeting 12,000 liters.

[Step 1] Agent queries get_field_segment_status("C") and retrieves the current segment state. Key finding: only **V3** is assigned to Segment C (not V1/V2/V3 as expected), and V3 is already open. The agent forms the initial plan: "an irrigation schedule likely exists; I need to find it."

[Step 2] Agent calls get_irrigation_schedule with schedule_id="SCHED_C".
Result: {"error": "IrrigationSchedule not found"}
↪ *Rather than updating its world model (''no schedule exists''), the agent retains its prior assumption and issues 20+ consecutive retry calls with guessed identifiers:*
schedule_id = "SCHED_C_1", "schedule_c", "1", "2", "IRR_C", "IRR_C_0700", "C_0700", "SC1", "SCHED_1", "farm7_c", "seg_c", "sched_seg_c", "tomato_south", "c_morning", "SCHED_C_AM_0700", "C_IRR", "C_daily", ...
(all return "IrrigationSchedule not found")

[Step 3 | After 20+ failed retries] Agent finally abandons schedule lookup and proceeds to *a different action without refreshing environmental state.* It assigns V1 and V2 to Segment C, then closes both V2 and V3 to ''prepare for the 07:00 start.''

Failure Analysis: The agent committed to a stale belief | that an irrigation schedule must exist | and exhausted over 20 tool calls retrying the same query with different ID guesses. At no point did it re-query the environment to verify whether a schedule was actually present, nor did it consider constructing a new schedule as the correct recovery action. *The agent executed against its*

initial assumptions rather than the evolving ground truth, a textbook instance of Stale-State Execution.

Figure 19: A representative Stale-State Execution failure: the agent makes 20+ blind retries on `get_irrigation_schedule` with guessed identifiers after the first failure, rather than refreshing its environment state and replanning.

Failure Mode Case: Misdiagnosis of Dynamic Triggers

Task: Book a connecting flight itinerary (PVG → CDG) with a layover in AMS, ensuring the total journey completes before 18:00 local time.

[Step 1] Agent calls search_flights("PVG", "AMS") and selects flight KL-891 (departs 08:30). It then calls search_flights("AMS", "CDG") and selects connecting flight AF-1234 (departs 14:00, ample layover). Plan looks feasible.

[Step 2 | Spatio-temporal trigger fires] A Schengen transit regulation update takes effect while the agent is mid-booking: passengers holding the user's passport category now require a transit visa for AMS, making KL-891 unavailable for booking.

[Step 3] Agent calls book_flight("KL-891", passenger_id="P-0042"). *Result:* {"error": "Booking rejected: passenger ineligible for this routing"}

↪ *The agent interprets the rejection as a **parameter formatting error** rather than a regulatory block. It proceeds to retry with surface-level variations:*

book_flight("KL-891", passenger_id="P-42") ×
book_flight("KL-891", passenger_id="0042") ×
book_flight("KL891", passenger_id="P-0042") ×
(all return the same eligibility rejection)

[Step 4] After exhausting ID format variants, the agent shifts its diagnosis: it now attributes the failure to a *transient seat-availability glitch* and calls get_flight_status("KL-891") to ''confirm the seat is still open.'' The status returns available, which the agent incorrectly treats as confirmation that the booking error was temporary. It retries book_flight("KL-891") once more.

[Step 5] Agent eventually attempts to reroute through a different hub (FRA), but selects LH-445 which also transits Schengen territory | the same regulatory block applies. Task fails.

Failure Analysis: The dynamic trigger (a regulatory restriction tied to passport category and transit location) produced a semantically distinct error signal | *passenger ineligibility* | that the agent consistently misread as either a parameter formatting issue or a transient system fault. The agent treated tool feedback as surface-level content to be retried rather than as evidence of a deeper environmental state change. Correct recovery required inferring *why* the booking was rejected before deciding *how* to respond (e.g., routing via a non-Schengen hub such as IST or DOH). This is a textbook instance of **Misdiagnosis of Dynamic Triggers**.

Figure 20: A representative Misdiagnosis of Dynamic Triggers failure: a Schengen transit restriction blocks the booking, but the agent repeatedly misattributes the rejection to parameter formatting errors or transient glitches, and subsequently selects an alternative route subject to the same undetected constraint.

Failure Mode Case: Missing Post-Adaptation Verification

Task: Fulfil a same-day vaccine delivery order of **500 doses** to Clinic-Downtown before 15:00. Doses are to be drawn from Warehouse-North (primary) with Warehouse-East as a fallback.

[Step 1] Agent calls get_inventory("Warehouse-North") and finds 500 doses available. It calls reserve_inventory("Warehouse-North", qty=500) | reservation succeeds. It then calls dispatch_vehicle("V-07", origin="Warehouse-North", destination="Clinic-Downtown").

[Step 2 | Spatio-temporal trigger fires] A hub outage at Warehouse-North causes its cold-chain system to fail, immediately releasing all reservations and marking its 500 doses as quarantined. Vehicle V-07's dispatch is cancelled automatically.

[Step 3] Agent observes the dispatch cancellation and correctly identifies that a disruption has occurred. It calls get_inventory("Warehouse-East") and finds **320 doses** available | a partial quantity. The agent calls reserve_inventory("Warehouse-East", qty=320).
Result: {"success": true, "reserved": 320}

[Step 4 | Adaptation step] Agent calls dispatch_vehicle("V-12", origin="Warehouse-East", destination="Clinic-Downtown", qty=320).
Result: {"success": true, "eta": "14:35"}
↪ *The dispatch tool returns success. The agent equates this intermediate action success with overall task completion and terminates:*

> Task Completed

[Ground truth check] Checklist evaluation reveals:

- [PASS] Disruption detected and primary route abandoned.
- [PASS] Fallback warehouse engaged within the time window.
- [FAIL] Total doses delivered: **320 / 500.** Shortfall of 180 doses unaddressed.
- [FAIL] Agent never queried remaining inventory at other depots (Warehouse-South holds 200 doses) nor attempted a supplementary dispatch to cover the deficit.

Failure Analysis: The agent correctly detected the dynamic trigger and executed a valid adaptation step (rerouting to Warehouse-East). However, it never verified whether the post-adaptation global state actually satisfied the original task constraint (500 doses delivered). Tool-level success | dispatch_vehicle returning success | was incorrectly equated with task-level success. In spatio-temporal dynamic environments, adaptation is complete only when the *resulting global state* is feasible with respect to all original requirements. This is a textbook instance of **Missing Post-Adaptation Verification.**

Figure 21: A representative Missing Post-Adaptation Verification failure: after a warehouse outage, the agent correctly reroutes to a fallback depot but terminates upon a successful dispatch call without verifying that the delivered quantity (320) meets the original requirement (500), leaving a 180-dose shortfall unresolved.

A.10 Prompt Templates

A.10.1 Prompt Templates of Stage 1: Environment Curation

In this section, we propose system prompts for environment curation stage.

A.10.2 TWO-STAGE FILTERING PROMPT TEMPLATE

In Environment Curation Stage, we first collect real-world user queries and conduct two-stage filtering to make sure STT-Arena is more diverse and suitable for spatio-temporal dynamic tasks. Figure 22 and Figure 23 show the system prompt template during two-stage filtering.

System Prompt for Stateful Task Filtering

You are a system that filters natural language tasks to determine whether they are **state-dependent, actionable requests** within a **persistent, domain-specific environment.**

Core Definition
We are ONLY looking for tasks that meet **all** of the following criteria:

1. Persistent Environment --- The query is about a domain where:

- There is a live, ongoing state that can be read or changed
- The environment supports both:
  - Information queries about current state (read operations)
  - Explicit state-changing actions (create, update, delete, move, cancel, etc.)

2. State Dependency --- The task cannot be answered correctly without:

- Inspecting the actual current data or configuration in the environment, and/or
- Executing an operation that modifies that data

3. Domain Specificity --- The environment is not general-purpose knowledge; it is a structured system such as:

- File management system with stored files/folders
- Order/logistics tracking system
- Calendar/scheduling system
- CRM, inventory, ticketing, project management tools
- Other specialized platforms with records that persist over time

4. Actionability in Context --- The query must correspond to an actionable operation or status check within the **actual environment** (not hypothetical).

Eligible Task Types

- **State queries**: ``Is invoice #1024 paid?'' / ``What meetings are scheduled for Wednesday?''
- **State modification operations**: ``Upload the proposal.pdf to the project folder'' / ``Cancel order #4512'' / ``Move meeting to 3 PM''

Explicit Exclusions
A request is **NOT eligible** if it is:

- Open-domain factual Q&A unrelated to a live state
- Casual conversation
- Content creation
- Pure hypothetical without actual environment interaction

- Isolated reasoning or calculations without accessing persisted state

Judgment Rule --- Be strict
Choose **YES** only if:

- The query cannot be answered from general knowledge alone
- AND it requires real-time access to persistent state in a domain-specific environment
- AND it targets an actionable operation (either a read or a write)
- AND the environment has the capability for both queries and modifications

If any criterion is missing → **NO.**

Task
Given a query, first analyze whether it implies or requires:

- A domain-specific environment with both query and modification capabilities
- Accessing or updating persistent state
- Performing a concrete, actionable operation

Then give your final judgment.

Output Format (Strictly enforce)

```
# Analysis
<Detailed reasoning whether this query depends on persistent state,
involves a stateful operation, and needs a capable environment as
defined>
# Answer
YES   (only if all strict criteria are met)
NO   (otherwise)
```

Figure 22: System prompt for the stateful task filter (Stage 1, Step 1). This prompt instructs the LLM to retain only queries that require a persistent, domain-specific environment with both read and write operations, discarding open-domain or hypothetical requests.

Spatio-Temporal Sensitive Filter

You are an expert judge for spatiotemporal dependency and multi-API conflict analysis on stateful tasks.
Your job is to judge whether a task should be selected as a benchmark candidate for the Step 3 conflict injection pipeline.

Conflict Taxonomy
A task should be kept only if a competent agent solving it could naturally encounter at least one realistic conflict mechanism during a normal multi-step workflow:

- **T1: State drift** --- resource mutates between observation and action.
- **T2: Validity expiration** --- a time-limited artefact expires between acquisition and use.
- **T3: Schedule/window violation** --- an operation becomes invalid outside a permitted time window.
- **S1: Resource locality mismatch** --- the target is bound to the wrong location / branch / node.

- **S2: Jurisdictional barrier** --- a policy, licensing, or regulatory boundary blocks the operation.
- **S3: Topology disruption** --- a physical or logical path becomes blocked or re-routed.
- **ST1: Dynamic spatial impact** --- a temporal event reshapes the spatial landscape.
- **ST2: Cascading dependency** --- a temporal failure propagates spatially to downstream resources.
- **ST3: Moving-window resource** --- a resource is only available within a joint time + location window.

Evaluation Dimensions
Evaluate the provided task using these dimensions:

1. Whether the task has real temporal affinity for T1/T2/T3 conflicts.
2. Whether the task has real spatial affinity for S1/S2/S3 conflicts.
3. Whether the task has joint spatiotemporal or strict dependent workflow structure supporting ST1/ST2/ST3 conflicts.
4. Whether the conflict lies on a normal, competent execution path rather than an unnatural contrived setup.
5. Whether the task has enough multi-step dependency that an injected conflict would be meaningful, observable, and benchmark-worthy.

Judgment Rule

- Answer **YES** only if the task naturally supports at least one concrete conflict code.
- Answer **NO** if the task is stateful but does not clearly support any realistic conflict mechanism.

Be Strict

- Prefer NO for simple CRUD-like stateful tasks.
- Prefer NO when time/space language is superficial.
- Prefer YES only when conflict opportunity is central to correct execution.
- Multi-step logic without plausible conflict codes should still be NO.

User Prompt
Analyze the following stateful task for spatiotemporal dependencies and multi-API call conflicts.
Task: {query}
Decide whether this task should be kept for Step 3 conflict injection, based on whether it supports one or more concrete conflict codes.

Output Format
Analysis
<detailed reasoning grounded in the conflict taxonomy>
Dependency Type
<Temporal / Spatial / Joint / Sequential / None>

```
# Conflict Codes
<comma-separated codes such as T1, ST2, S3, or None>
# Answer
YES or NO
```

Figure 23: System prompt for the spatio-temporal sensitivity filter (Stage 1, Step 1). This prompt evaluates whether a stateful query naturally supports at least one concrete spatio-temporal conflict mechanism from the nine-type taxonomy, filtering out tasks with only superficial temporal or spatial language.

A.10.3 Prompt Template for Environment Synthesis

We then generate static executable environments based on the filtered queries. First, we prompt an LLM to infer the environment information as the system prompt template is shown in Figure 24. Then we generate the entity attributes and tool specifications as illustrated in Figures 25 and 26. Finally, as shown in Figures 27 and 28, we prompt an LLM to implement the entity attributes and tool specifications to Python classes and concatenate them to a complete static environment.

Infer Environment Information

You are a Task Analyst.
Given a raw task description, your objective is to identify the most plausible stateful and domain-specific environment in which this task would naturally occur.
The chosen environment should strike a balance: not so broad as to be meaningless, and not so narrow as to apply only to a single, highly specific case. It should be scoped such that this task, along with similar related tasks, can be executed meaningfully.

Guidelines

- If multiple environments seem equally plausible, select one at random rather than listing all possibilities.
- Example: if a task could occur in a Linux, Windows, or macOS filesystem, randomly choose one instead of remaining indecisive.

Required Response Sections

Your response must include the following sections:

1. **# Analysis**
   - Explain the reasoning process used to connect the task to the chosen environment.
   - Note any relevant entities, constraints, relationships, or dynamics implied by the task.
2. **# Environment Summary**
   - Provide a concise label for the environment type.
   - Examples: Linux filesystem, E-commerce order management system, Airline booking system.
3. **# Environment Introduction**
   - Introduce the environment itself, without referring to the current task.
   - Focus on its inherent structure, the nature of the state it maintains, typical operations it supports, and its general real-world scope.
   - Limit to approximately three sentences.
4. **# Metrics**

- **Usefulness (1--10):** how broadly applicable and valuable this environment is in real-world scenarios.
- **Modelability (1--10):** how straightforward it would be to represent this environment using a single Python class with stateful attributes and operational methods.

Output Format

```
# Analysis
<Your analysis>
# Environment Summary
<Your environment summary>
# Environment Introduction
<Your environment introduction>
# Metrics
Usefulness: [1--10]
Modelability: [1--10]
No additional text or commentary.
```

Figure 24: System prompt for inferring the latent environment context from a seed query (Stage 1, Step 2). The LLM identifies the most plausible domain-specific environment and provides a structured summary, introduction, and feasibility metrics to guide subsequent synthesis.

Infer Entity Attributes

You are an expert task and environment analyst.
Given an environment description and an example task in this environment, infer the set of state variables (state space) maintained by the environment.
The state should not be too broad (e.g., all possible data in an e-commerce system), nor too narrow (only for this single task).
It should be reasonably designed to support this task and similar tasks in the same environment.

Input Format

```
# Environment Summary
<Environment summary>
# Environment Introduction
<Environment introduction>
# A Example Task in This Environment
<Example task>
```

Required Output Sections

1. **# Analysis**
   - Explain what states are involved in the environment.
   - Identify what entities and attributes need to be tracked.
   - Note relevant constraints, operational rules, and dependencies.
2. **# State Space Definition**
   - Define the major entities maintained by the environment.
   - For each entity, specify attributes and describe its functional role.
3. **# Constraints & Rules**
   - Summarize core consistency rules, domain constraints, permissions, capacities, temporal rules, or structural restrictions.

Output Format

```
# Analysis
<Your thought process>
# State Space Definition
- Entity: EntityName1
  - Attributes: Attribute1, Attribute2, ...
  - Description: The role of this entity in the environment
- Entity: EntityName2
  - Attributes: ...
  - Description: ...
# Constraints & Rules
- Constraint 1
- Constraint 2
...
Do not include any additional text.
```

Figure 25: System prompt for inferring entity attributes and the state space definition (Stage 1, Step 2). Given an environment summary and an example task, the LLM enumerates the major entities, their attributes, and domain constraints that must be tracked across multi-step tool interactions.

Infer Tool Specification

You are an expert in building and analyzing agent environments. Given an environment summary, introduction, state space definition, constraint rules, Python base class definition, and example task, your goal is to analyze the current environment and generate the list of operations needed to support the task in this environment. Each operation will later be converted into a callable class function for the agent.

Key Points

- Operations are divided into two categories:
  - **Information Query Class**
  - **State Change Class**
- Each operation must include:
  - Operation name
  - Brief description
- Before output, first write **# Analysis**:
  - Explain task logic
  - Determine which operations are query operations
  - Determine which are state-changing operations
  - Explain how environment constraints affect operation design

Input Format

Based on the following environment specification, produce the operation list.

```
{
"environment_summary": "...",
"environment_introduction": "...",
"state_space_definition": [...],
"constraints_rules": [...],
"environment_class_definition": "...",
"environment_example_task": "..."
}
```

Output Format

```
# Analysis
<Explain operation requirements, classification logic, and how
constraints affect the operation set>
# Operation List
## Information Query Class
- Operation:  OperationName   Description:  xxxx
- Operation:  OperationName   Description:  xxxx
- ...
## State Change Class
- Operation:  OperationName   Description:  xxxx
- Operation:  OperationName   Description:  xxxx
- ...
Strictly follow this format.
```

Figure 26: System prompt for inferring the tool operation list (Stage 1, Step 2). The LLM generates a categorized list of query and state-change operations required to support task execution within the synthesized environment.

Infer Entity Attributes Python Code

You are an AI coding assistant.
Your job is to translate an environment specification into a Python environment class definition.
The class should simulate the stateful environment structure (without methods yet).
You should first analyze the specification and then generate code.

Rules of Analysis

- Determine the environment class name using the environment summary or an appropriate adaptation. Example: LinuxFileSystem, EcommerceOrderSystem.
- Extract attribute names from each entity in state_space_definition.
- If needed, generate corresponding TypedDict definitions.
- Infer attribute value types using Python primitive types:
  - id, name, category → str
  - price, size → float / int
  - quantity → int
  - status → str
  - timestamps → str / float
- constraints_rules should be preserved as comments.

Rules of Code

- Generate each TypedDict definition if needed.
- Generate the environment class with only:
  - __init__
  - state attributes
- Use attributes of type Dict<ID, TypedDict> where appropriate.
- Add comments mapping attributes back to state-space entities.
- Annotate constraints as code comments.

- Do not implement business logic or methods yet.

Input Format

```
# Environment Summary
<short label>
# Environment Introduction
<paragraph intro>
# State Space Definition
[ { "entity":  "EntityName",
"attributes":  "attr1, attr2, ...",
"description":  "short description" }, ...  ]
# constraints_rules
constraint 1 ...
constraint 2 ...
```

Output Format

```
# Analysis
<Explain class naming, entity-to-structure mapping, stored fields,
dict/list design, and constraint annotations>
# Class Definition
'''python
<Python environment class definition>
''' Do not include any additional text.
```

Figure 27: System prompt for generating the entity attribute Python class (Stage 1, Step 2). The LLM translates the state space definition into a typed Python class with only __init__ and state attributes, annotating constraints as code comments.

Infer Tool Specification Python Code

You are a code generation assistant.
Given an agent environment (including environment summary, introduction, state space definition, constraint rules, base class definition, and operation list), your task is to implement a single target operation as a Python method within the environment class.

Operation Types

- Information Query Operations
- State Modification Operations

Core Requirements

For a given Target Operation:

1. **# Analysis**
   - Identify involved entities and attributes
   - Determine required parameters
   - Define expected outputs (query vs modification)
   - Identify edge cases (invalid input, missing state, permission issues)
   - Consider relevant environment constraints
2. **# Code**
   - Implement method as def operation_name(self, ...)
   - Method must be inside an existing environment class (not standalone)
   - Use type hints
   - Include docstring (inputs, outputs, constraints)
   - **Do not raise exceptions**

```
        • Return structured dictionaries:
          - Success: {"success": True, "data": ...} (query)
          - Success: {"success": True, "message": ...} (state
            change)
          - Failure: {"success": False, "error": "..."}

Input Format
# Environment Summary
<environment_summary>
# Environment Introduction
<environment_introduction>
# State Space Definition
<state_space_definition>
# Constraints Rules
<constraints_rules>
# Class Definition
```python
<class_definition>
```
Operation List
operation_list
Target Operation

"operation_name": "<operation_name>",
"operation_description": "<operation_description>",
"operation_type": "<query_or_state_change>"

Output Format
Analysis
<Reasoning: inputs, outputs, entities, attributes, constraints,
edge cases>
 # Code ```python
def <operation_name>(self, ...):
"""
<docstring describing inputs, outputs, constraints>
"""
Implementation
```
```

Figure 28: System prompt for implementing individual tool methods as Python code (Stage 1, Step 2). Each operation from the tool list is implemented as a class method returning structured success/failure dictionaries, with no exception raising.

A.10.4 Functional Validation Prompt Template

We conduct functional validation when obtain the candidate static environments. First, we prompt the tool-calling LLM to generate test configurations and the system prompt is shown in Figure 29. Then, as shown in Figure 30, we prompt the tool-calling LLM to generate the sequence of validation tool calls to filter the candidate static environments.

Test Config Generation

```
You are an AI assistant.
You will be given the complete definition of a Python class.
This class represents an environment state in a specific domain
and contains various attributes (such as dictionaries, lists,
TypedDict objects, dataclasses, etc.) used to manage entities
and their relationships within the system.
```
```

Based on the class definition, generate a JSON object that can serve directly as the class initialization configuration (config), following the rules below.

1. **Structure and Type Matching**
   - The JSON must strictly follow the attribute structure and data types required by the class.
   - Field names, nesting levels, and value types must match the class definition exactly.
2. **Respect Constraints**
   - Read class methods and docstrings to identify constraints (e.g., valid status values, required fields, ID reference rules).
   - Ensure all generated data complies with these constraints.
   - All cross-entity references must be valid and consistent.
3. **Richness of Data**
   - Each major dictionary-like attribute should contain 3--5 diverse entries.
   - Cover different valid states and value ranges.
   - Dates should be distributed over a reasonable time span.
   - Numerical fields should vary realistically.
4. **Realistic Simulation of Data**
   - Use natural fictional names (e.g., Alice Chan, Central City District).
   - Avoid placeholder-like values (e.g., name1, user001).
   - Dates must be in ISO format (YYYY-MM-DD) or timestamps.
   - IDs should be unique and may mix short codes and UUIDs.
   - All data must be fictitious and non-sensitive.
5. **Output Format**
   - Output only JSON (no explanations outside required sections).
   - Must be directly usable as class initialization config.

Input
Env Class Definition
```python
{env_class_code}
```
All Containers
{all_containers} **Output Format**

```
# Analysis
<Reasoning:  containers, fields, constraints, and data
construction strategy>
 # Init Config
{
...
}
```

Figure 29: System prompt for generating test configurations for functional validation (Stage 1, Step 3). The LLM produces a realistic, constraint-compliant JSON initialization config with 3–5 entries per major entity to support diverse validation scenarios.

Sequence of Tool Calls Generation

You are an experienced testing engineer, performing comprehensive exploratory testing on all tool interfaces (methods) of a simulated environment class.
Your goal is to verify the behavior of each method under different types of inputs, aiming to uncover potential errors, exceptions, and state inconsistencies.
In each testing round, you will generate one tool invocation as a test case. After execution, you will receive the environment's return information along with a backend evaluation (pass, warning, fail).

Environment Introduction
{env_introduction}

Available Tool Interface List
{tool_info}

Testing Strategy

- **Positive case testing:** use valid parameters that comply with interface definitions.
- **Negative case testing:** use invalid, missing, or malformed parameters (wrong types, non-existent IDs, out-of-range values, etc.).
- **Special case testing:** include boundary values, null/empty inputs, extreme values, and special characters.
- Ensure all tool interfaces are covered; balance both breadth and depth of testing.
- No need to maintain a consistent task objective; exploration is encouraged.

Testing Rules

- Only one tool invocation per round.
- Parameters must be in dictionary format.
- Keys must be valid; values may be intentionally invalid for robustness testing.
- Do not call methods outside the provided tool list.

Output Format
Strictly follow:

 # Thought
<Brief explanation of chosen method and parameter strategy>

 # Selected Function
<Method name>

 # Parameters Dictionary
<Parameter dictionary>

Figure 30: System prompt for generating the sequence of validation tool calls (Stage 1, Step 3). A tool-calling LLM performs exploratory testing across positive, negative, and boundary cases to verify that all tool interfaces execute correctly before an environment is admitted to $\mathcal{E}_{\text{static}}$.

A.11 Prompt Templates of Stage 2: Spatio-Temporal Dynamic Injection

In this section, we propose system prompts for spatio-temporal dynamic injection stage.

A.11.1 Conflict Assignment and Blueprint Generation Prompt Template

During Stage 2, we first assign several proper conflict types to a static environment, the system prompt can be found in Figure 31. Then, based on the conflict types and environment information, we generate a blueprint which serves as a generative contract ensuring internal consistency across all downstream components. The system prompt is shown in Figure 32.

Conflict Assignment

You are a benchmark designer for spatiotemporal AI agent evaluation.

Task
Identify which conflict types from the taxonomy are feasible for the given environment.
A conflict type is considered feasible only if the environment's state space and operations can naturally support:

- a valid activation trigger,
- a conflict firing mechanism,
- and a resulting observable state update.

Conflict Taxonomy
{taxonomy_block}

Environment
{env_summary}

Output Format
Analysis
<Reasoning: which conflict types are supported by the environment, and why others are not>
Matched Conflicts

- **Conflict:** <T1 | T2 | T3 | S1 | S2 | S3 | ST1 | ST2 | ST3>
  Rationale: <1--2 sentence explanation>
  Activation Operations: <comma-separated operation names from environment>
  Observation Operations: <comma-separated operation names from environment>
- **Conflict:** <type>
  Rationale: <explanation>
  Activation Operations: <...>
  Observation Operations: <...>

Rules

- Activation operations must be those that can trigger the conflict.
- Observation operations must reveal the resulting state change.
- Only use real operation names from the environment.
- Prefer conflicts that are genuinely supported by both state and operations.
- Include only 3--6 feasible conflict types.

Do not output anything outside the required format.

Figure 31: System prompt for conflict type assignment (Stage 2, Step 1). Given a static environment's state space and operations, the LLM identifies which of the nine spatio-temporal conflict types are naturally supportable, specifying the relevant activation and observation operations.

Blueprint Design

You are a benchmark scenario designer for AI agent evaluation.

Task
Design a **CONCRETE conflict scenario** for the given environment.
Your output must specify:

- exact operation names,
- state container names,
- field-level mutation descriptions.

Abstract or vague descriptions are not acceptable.

Environment
{render_json(env_summary)}

Conflict Type

- **Type:** {conflict_type} --- {CONFLICT_TAXONOMY[conflict_type]}
- **Event pattern:** {render_json(EVENT_TEMPLATE_PATTERNS[conflict_type])}
- **Match rationale:** {match_rationale}

Difficulty
{difficulty}
{diff_instructions}

Design Rules

1. Normal flow must use ONLY operations from the environment's operation list.
2. Activation operation must be a **READ/QUERY** operation where the conflict becomes visible during normal execution.
3. Mutations must reference REAL state container names from state_space_definition.
4. Mutations must describe **field-level changes**, not vague availability changes.
5. The conflict must match its taxonomy semantics (e.g., T2 requires temporal ordering changes).
6. Observable_via must be an operation that reveals the mutated state.
7. Trigger must depend on TASK or STATE CONDITIONS, not call order.
8. Trigger type must be:
   - always_once: fires once when condition is met
   - conditional_guarded: keeps affecting execution until recovery

Output Format
User Goal
<1--2 sentences describing the user's objective>
Normal Flow

- <operation_name> --- <purpose>
- <operation_name> --- <purpose>
- ...

Conflict Design

- **Activation Operation:** <exact operation name>
- **Trigger Mechanism:** <always_once | conditional_guarded>
- **Trigger Condition:** <concrete condition that causes conflict>
- **Recovery Guard Condition:** <or "none">
- **Mutations:**
  - State: <state_container.field_path> | Change: <old → new>
  - State: <state_container.field_path> | Change: <old → new>
- **Observable Via:** <operation name>

{Recovery Path if diff_info["task_solvable"] else "Impossible Rationale"}

- If solvable: step-by-step recovery using real operations
- If not solvable: explain violated constraints

Output ONLY the sections above.

Figure 32: System prompt for conflict blueprint design (Stage 2, Step 1). The LLM generates a structured, field-level conflict scenario blueprint encoding the user goal, nominal tool sequence, trigger condition, state mutations, recovery path, and assigned difficulty level.

A.11.2 Dynamic Environment Construction Prompt Template

Based on the blueprint, we prompt an LLM to augment the static environment to a spatio-temporal dynamic one. The system prompt can be found in Figure 33. Then, we generate the user query, initial configuration, and user profile as shown in Figures 34 and 35 to obtain the complete dynamic tasks.

Spatio-Temporal Dynamic Trigger Injection

You are an expert Python developer.
Your task is to modify an environment class to inject a deterministic conflict trigger.

Original Environment Code

```python
{env.get("env_class_code", "")}
```

Conflict Specification

- **Type:** {conflict_type} --- {CONFLICT_TAXONOMY[conflict_type]}
- **Activation Operation:** {activation_op} (method that triggers the conflict)
- **Trigger Mechanism:** {trigger_mechanism}
- **Trigger Condition:** {trigger_condition}
- **Recovery Guard Condition:** {recovery_guard_condition}
- **Observable Via:** {observable_op}

Exact Mutations to Apply

When the trigger condition is met inside {activation_op}, apply the following mutations:
{mutation_block}

Injection Rules

1. Add self._conflict_triggered = False in __init__
2. Trigger must depend on **real state or input conditions**, not call counts
3. Implement trigger logic inside {activation_op}
4. If **always_once**:
   - Fire only once when condition is first satisfied
   - Apply exact mutations
   - Set self._conflict_triggered = True
5. If **conditional_guarded**:
   - Fire whenever condition holds AND recovery condition is not satisfied
   - _conflict_triggered is bookkeeping only
   - Stop firing once recovery condition is satisfied
6. Normal operation must always return valid results (no error injection)
7. Mutated state must remain fully usable by all methods
8. Do not introduce artificial error handling logic
9. {observable_op} requires no modification unless strictly necessary
10. Output the full modified class only

Output Format

```
# Conflict Environment Code
```python
<complete modified Python class>
```
```

Output ONLY the required section.

Figure 33: System prompt for injecting spatio-temporal triggers into a static environment (Stage 2, Step 2). The LLM augments the Python environment class with a deterministic conflict trigger inside the designated activation operation, supporting both always_once and conditional_guarded firing mechanisms.

User Query and Initial Configuration Generation

You are a benchmark data engineer. Given an environment and an abstract scenario design, produce concrete data that makes the scenario executable.

Environment
{render_json(env_info)}

Abstract Scenario Design
{scenario}

Difficulty
{item['difficulty']}
- Task solvable after conflict: {diff_info['task_solvable']}
- Recovery complexity: {diff_info['recovery_complexity']}

Your Task
Generate THREE components:

1. User Query (2--5 sentences)

- Must contain all necessary entity names, dates, locations, and identifiers
- Must match the init configuration exactly
- Must NOT reveal the existence of any conflict
- Must sound like a natural user request

2. Init Config (JSON)

- Must cover all state containers in the environment
- Must include sufficient entities for both normal and recovery paths
- Must ensure full referential consistency
- Must include at least one alternative entity for recovery

3. Concrete Mutations (JSON array)

- Map abstract mutations to exact entity IDs and fields
- Each entry includes: state container, entity ID, field, old value, new value
- old value must match init config exactly
- new value must induce intended conflict

Output Format

```
# User Query
<2--5 sentence task instruction>
# Init Config
```json
{ }
```
Concrete Mutations
```json
[ ]
```
```

CRITICAL
The init config and concrete mutations must be fully consistent. All old values must exactly match those in the init config.

Figure 34: System prompt for generating the user query, initial configuration, and concrete mutations (Stage 2, Step 2). The LLM grounds the abstract blueprint into a fully executable task instance with consistent entity IDs, field-level mutation values, and a natural-sounding user request.

User Profile Generation

You are a user persona designer for an AI agent evaluation benchmark.

Context
An AI agent will interact with a simulated user to complete a task in a dynamic environment.

User Query
{user_query}

Scenario

- **Environment:** {item.get("environment_summary", "")}

- **Difficulty:** {item["difficulty"]}
- **Task solvable:** {diff_info["task_solvable"]}

User Profile Requirements
The profile helps the simulator:

- Respond naturally to agent questions
- Provide clarification when asked
- NOT proactively reveal conflict information
- Express preferences that guide the agent's decision-making
- { "Accept reasonable alternatives if the original plan fails" if diff_info["task_solvable"] else "Insist on the original requirements; alternatives are NOT acceptable (task is impossible)" }

Output Format
User Profile

- **Name:** <realistic name>
- **Background:** <1--2 sentences>
- **Communication Style:** <brief | verbose | formal | casual | technical | non-technical>
- **Frustration Threshold:** <low | medium | high>
- **Flexibility:** <medium or high / low>

Key Preferences

- <preference>
- ...

Clarification Responses

- **Topic:** <likely agent question>
  Response: <how user would respond>
- ...

Figure 35: System prompt for generating the user profile (Stage 2, Step 2). The LLM creates a persona for the user simulator, specifying communication style, flexibility, and clarification responses tailored to whether the task is solvable or impossible.

A.12 Prompt Templates of Stage 3: Dual-Agent Assessment

In this section, we propose system prompts for dual-agent assessment stage.

A.12.1 Checklist Generation Prompt Templates

We prompt an LLM to generate checklist and check functions according to the user query and blueprint. Figure 36 shows the prompt template of solvable tasks.

Checklist Generation of Solvable Tasks

You are a benchmark verification designer. Generate a checklist and Python function verifying correct CONFLICT-AWARE outcome.

User Query
{item.get("user_query", "")}

```
Init Config
{render_json(item.get("init_config", {}))}
Conflict Type
- {item["conflict_type"]} | {CONFLICT_TAXONOMY[item["conflict_type"]]}
Concrete Mutations (state changes from conflict)
{render_json(item.get("concrete_mutations", []))}
Difficulty: {item["difficulty"]} | {diff_info["description"]} -
Recovery complexity: {diff_info["recovery_complexity"]}
Environment
{render_json(env_info)}
Checklist Requirements
    • Verify the conflict was detected (agent re-queried and saw
      changed state)
    • Verify the recovery/alternative path was executed correctly
    • Verify the final state satisfies the user goal via the
      alternative path
    • DO NOT check for conflict_triggered (that's internal)
check_conflict(env) Requirements
    • Inspects env.<state_container> attributes directly
    • Returns: {{"passed": bool, "details": [{"check_id":
      "cc_1", "passed": bool, "message": str}, ...]}}
    • No external imports
Output Format
# Conflict Checklist
- Check: cc_1
  - Description: <specific>
  - Target State: <container name>
  - Expected Condition: <testable>
...
# Conflict Check Function
def check_conflict(env) -> dict:
    ...
```

Figure 36: System prompt for generating the evaluation checklist and check functions for solvable tasks (Stage 3, Step 1). The LLM produces a set of conflict-aware success criteria and corresponding Python check functions that verify conflict detection, recovery path execution, and final state validity.

A.12.2 Dual-Agent Validation Prompt Templates

We conduct dual-agent validation to obtain executable spatio-temporal dynamic environments and tasks. As shown in Figure 37, plan-agent first plans a sequence of tool calls and check-agent then evaluates the trajectories of the execution results of planning tool calls as Figure 38 illustrates.

Plan-Agent System Prompt

```
You are the plan agent for conflict-environment validation.
Your job is to produce a COMPLETE tool-call plan that executes the
task from the original user request, follows the designed story,
naturally reaches the conflict, and then continues far enough to
verify whether the conflict environment behaves correctly.
User Query
{item.get('user_query', '')}
```

Scenario Story
{render_json(story)}

Full Init Config
{render_json(item.get('init_config', {}))}

Available Tools
{render_json(_build_tool_reference(item))}

Requirements

- Output a COMPLETE executable plan for the task, not a minimal trigger-only probe.
- The plan should reflect the user_query plus the normal_flow story in order.
- Use ONLY real tool names from the available tools.
- Use ONLY concrete argument values grounded in the user_query and init_config summary.
- The plan must naturally reach the conflict through the original intended workflow, not through a synthetic shortcut.
- After the conflict is triggered, include the necessary follow-up calls to observe the changed state and verify whether task progress behaves as expected.
- Do not add explanations outside the requested section.

Output Format

```
# Test Plan
[
  {"method": "<method_name>", "arguments": {...}},
  ...
]
```

Output ONLY the section above.

Figure 37: System prompt for the plan agent in dual-agent verification (Stage 3, Step 2). The planning agent produces a complete, blueprint-guided tool-call sequence that naturally reaches the injected conflict and includes follow-up calls to observe post-trigger state changes.

Check-Agent System Prompt

You are the check agent for conflict-environment validation.
Judge whether the executed path is valid and whether the spatiotemporal conflict is triggered in the intended workflow.

Scenario Story
{render_json(story)}

Conflict Story
{render_json(conflict_story)}

Tool Calls And Results
{render_json(validation.get('trajectory', []))}

Decision Rules

- Verify whether the tool-call path follows the intended normal workflow in the story.
- Verify whether the called tools are appropriate and whether the calls execute normally for the intended path.
- Verify whether there are unexpected errors that should not appear in a valid execution path.

- Verify whether the spatiotemporal conflict is triggered at the intended point in the path according to the conflict story.
- Return Pass if the path is correct, the tool executions are normal enough for the intended workflow, no inappropriate errors appear, and the intended conflict trigger is correctly evidenced.
- Return Fail otherwise.
- Return Warning only if the main path is mostly correct but the evidence is slightly incomplete.

Output Format
Analysis <step-by-step reasoning>
Result Pass
Error Reason <No error, or the concrete reason>
Output ONLY the three sections above.

Figure 38: System prompt for the check agent in dual-agent verification (Stage 3, Step 2). The checking agent evaluates the executed trajectory against three behavioral invariants: correct trigger timing, correct state mutation, and failure of the original plan after the conflict fires.

A.12.3 Consistency Check Prompt Templates

Finally, we conduct consistency check to make sure our spatio-temporal dynamic environments and tasks are consistency and correct. The system prompt is shown in Figure 39.

Consistency Check System Prompt

You are a benchmark quality auditor. Judge whether this scenario is INTERNALLY CONSISTENT across all artifacts.

Scenario ID: {item.get("scenario_id","")}

Story
{render_json(story)}

User Query
{item.get("user_query", "")}

Full Init Config
{render_json(item.get("init_config", {}))}

Conflict Type: {item.get("conflict_type","")} | {conflict_desc}
Difficulty: {item.get("difficulty","")}

Concrete Mutations
{render_json(item.get("concrete_mutations", []))}

Conflict Design
{render_json(conflict_design)}

Evaluation Checklists
{checklist_text}

Your Job
Check whether the artifacts are mutually consistent. Focus on concrete contradictions, missing links, or implausible transitions.
You must judge at least these dimensions:
- Query{Config Coherence: does the user_query match the entities, values, dates, IDs, and constraints in init_config?
- Story{Query Alignment: do user_goal, normal_flow, and recovery/impossible path actually support the user_query?

- Conflict{Mutation Alignment: do the concrete_mutations and conflict_design really implement the claimed conflict semantics?
- Checklist Coverage: do the evaluation checklists test the key success/failure conditions implied by the query and story?
- Difficulty Consistency: is the scenario complexity and recovery/impossibility behavior consistent with the labeled difficulty?
- Overall Narrative Consistency: do all artifacts tell one coherent story without contradictions?

Treat a contradiction as fatal when it would make the benchmark invalid or misleading.

Output Rules

- Do NOT score.
- Do NOT summarize vaguely.
- Return 'pass' only if there are no fatal consistency issues.
- Return 'fail' if any fatal issue exists.
- Keep each issue concrete and artifact-grounded.

Output Format (strict JSON only)

```
{
    "verdict": "pass" | "fail",
    "checks": {
        "query_config_coherence": {"passed": true, "reason": "..."},
        "story_query_alignment": {"passed": true, "reason": "..."},
        "conflict_mutation_alignment": {"passed": true, "reason": "..."},
        "checklist_coverage": {"passed": true, "reason": "..."},
        "difficulty_consistency": {"passed": true, "reason": "..."},
        "narrative_consistency": {"passed": true, "reason": "..."}
    },
    "fatal_issues": ["<issue1>", ...],
    "minor_issues": ["<issue1>", ...],
    "suggestions": ["<suggestion1>", ...]
}
```

Figure 39: System prompt for the consistency auditor (Stage 3, Step 3). An LLM-based auditor verifies mutual coherence across all task artifacts—user query, init config, conflict design, concrete mutations, checklist, and difficulty label—returning a structured JSON verdict with issue descriptions.

A.13 Prompt Templates of STT-Arena Evaluation

In this section, we propose the system prompts during evaluation of STT-Arena. Figures 40, 41, and 42 show the system prompt for tested LLMs, user simulators, and LLM-as-a-judge.

Tested LLM System Prompt

You are an assistant operating inside a tool-based benchmark environment.
Rules:

- Each turn must contain exactly one tool call.
- Never make parallel tool calls or return multiple tool calls in the same turn.

- Do not output a normal text response without a tool call, unless you are responding with exactly: Task Completed
- Prefer querying the environment before making changes.
- If user clarification is needed and the tool is available, use chat_with_user.
- When the task is complete, respond with exactly: Task Completed
- Do not invent tool results or hidden state.

Figure 40: System prompt for the evaluated LLM during STT-Arena benchmarking. The model is instructed to issue exactly one tool call per turn, prefer environment queries before state changes, and terminate by outputting "Task Completed".

User Simulator System Prompt

```
You are simulating the benchmark user.
Rules:
```

- Reply as the user in plain natural language only.
- Stay consistent with the provided profile and previously revealed facts.
- Do not invent facts that are not grounded in the profile.
- Reveal withheld details only when the assistant asks for them directly.
- If the task is fully satisfied, reply with the completion signal exactly and nothing else.

Figure 41: System prompt for the passive user simulator during evaluation. The simulator responds in natural language consistent with the user profile, reveals withheld details only upon direct query, and emits a fixed completion signal when the task is fully satisfied.

LLM-as-a-Judge System Prompt (Impossible Task)

```
You are judging whether an agent handled an impossible task
correctly.
Ground truth: this task has no compliant completion path.
Mark the trajectory correct only if the agent recognizes that the
task cannot be completed compliantly and communicates that blocked
state back to the user.
Return JSON only:
{
  "passed": true,
  "recognized_impossibility": true,
  "fed_back_to_user": true,
  "evidence": ["short evidence"],
  "verdict": "short explanation"
}
```

Figure 42: System prompt for the LLM-as-a-judge on impossible tasks. The judge determines whether the agent correctly recognized task infeasibility and communicated this to the user, returning a structured JSON verdict with binary verdicts and supporting evidence.