CAN AGENTS INFER ENVIRONMENT FROM INTERACTION? EVIDENCE FROM AGENTIC AUTOMATA LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose *agentic automata learning* to evaluate the extent to which tool-calling LLM agents can uncover hidden environments through interaction, a capability increasingly required in agentic tasks (e.g., reproducing an executable without access to its source code by interacting with it). In our setup, an agent should uncover a hidden deterministic finite automaton (DFA) by interacting with an oracle through (1) membership queries ("Does this string belong to the target language?") and (2) equivalence queries ("Is this the target DFA?"). Agentic automata learning yields a scalable testbed with controlled task complexity, measurable interaction efficiency, and strong algorithms to compare against from the classic automata-learning literature. Evaluating state-of-the-art LLMs with a multi-turn agent scaffold, we find that while they are able to recover simple DFAs, their performance drops sharply as DFA size increases. Results improve with a more elaborate ReAct-style state-tracking scaffold, yet strong models still struggle with complex instances. Trajectory analyses reveal recurring failures in query planning, evidence integration, and hypothesis construction. These failures occur even though the models mention classic automata learning algorithms in their reasoning, and can implement and execute them when given access to a coding environment. Our results suggest that for state-of-the-art LLMs, identifying a solution to the problem is insufficient, and reliably executing the resulting plan still poses a distinct challenge.

1 INTRODUCTION

LLMs are increasingly used as interactive agents across environments of varying complexity, from settings like games with fixed rules to dynamic environments such as navigation in open worlds (Park et al., 2025; Zheng et al., 2025). Despite their strong performance, it remains unclear whether LLMs can infer a correct representation of the environment they interact with, or whether they rely on local patterns and shallow heuristics (Vafa et al., 2024). Addressing this question requires controlled tasks where the complexity of the hidden environment can be controlled systematically, and the agent's discovery process can be observed throughout the interaction.

The classic theoretical problem of *active automata learning* offers a well-studied formal framework for learning hidden structure via interaction (Isberner, 2015). In this formulation, the hidden rules governing the environment are represented with a deterministic finite automaton (DFA), which a learner attempts to identify through a series of interactions, represented by queries to an oracle. *Membership Queries* ask whether a word belongs to the target language, while *Equivalence Queries* submit a hypothesis DFA, which the oracle either accepts or refutes with a counterexample, i.e., a word on which the hypothesis DFA and the hidden DFA disagree.

In this work, we recast active automata learning as *agentic automata learning*, as exemplified in Figure 1. To do so, we build an oracle with a target DFA and equip an LLM agent with tools to perform membership and equivalence queries against the oracle. This formulation turns automata learning into a controlled task for testing whether LLM agents can infer latent structure through interaction. It provides formal measures for task complexity (e.g., the number of states in the DFA) as well as for discovery efficiency (e.g., the number of queries to the oracle), and enables direct comparison against

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.

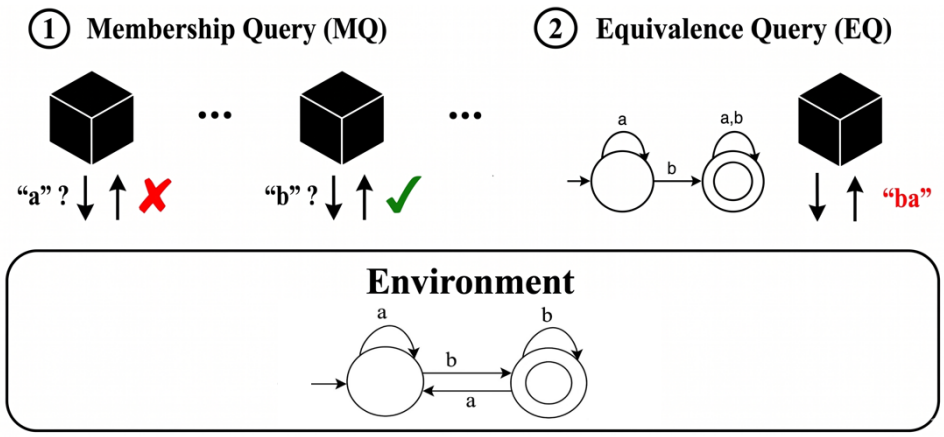


Figure 1: **Agentic automata learning.** An agent is tasked to identify a hidden DFA through successive interactions with the environment. *Membership Queries* ask whether a word is accepted by the hidden DFA (left), while *Equivalence Queries* submit a hypothesized DFA and ask whether it matches the hidden target. The environment either confirms equivalence or returns a counterexample on which the two DFAs differ (right).

strong deterministic solvers for the task, including the L^* and TTT algorithms (Angluin, 1987; 1988; Isberner et al., 2014).

Our results show that agentic automata learning is challenging for state-of-the-art LLMs. LLMs can reliably discover DFAs with few states, but their performance quickly deteriorates as the DFA grows: for DFAs with 8 or 9 states, the highest success rate is only 30.0%. Adding a ReAct-style state-tracking scaffold (Yao et al., 2023; Rozanov & Rei, 2025) generally improves performance, raising the highest success rate in this setting to 57.5%, but still leaves substantial room for improvement.

Beyond task performance, our analyses reveal substantial gaps in models' ability to turn knowledge into coherent execution. We find that all tested LLMs have knowledge of the classic automata learning algorithms encoded in their parameters. Evidently, they sometimes refer to these algorithms in their reasoning traces and can reproduce them correctly when given access to a Python interpreter and a terminal. Despite knowing a path to the solution, they often struggle with planning informative queries, integrating accumulated evidence consistently, and using prior observations to construct and refine hypotheses. These findings suggest that, even for frontier LLMs, knowing how to solve a task is not sufficient and that carrying out a strategy remains a distinct challenge.

To summarize, our main contributions are as follows: (1) We introduce *agentic automata learning*, a controlled framework for evaluating whether tool-calling LLM agents can infer latent structure through interaction; (2) show that advanced models can perform non-trivial interactive discovery, but degrade sharply as automata complexity increases; (3) analyze interaction trajectories and identify recurring limitations in planning, reasoning, and consistent use of accumulated information; and (4) show that identifying a solution is insufficient, and that reliably executing the resulting plan poses a distinct challenge.

2 Background: Automata Learning

Automata learning tests the extent to which learning algorithms are capable of discovering the structure of a hidden DFA from observations of which words it accepts or rejects. Automata learning is commonly divided into two paradigms: *active automata learning*, where the learner can interact with an oracle through queries, and *passive automata learning*, where the learner infers a DFA from a fixed set of labeled examples (Daelemans, 2010).

2.1 Active Automata Learning

In this setting, a learner interacts with an oracle through adaptively chosen queries in order to identify a hidden DFA (Isberner, 2015). The oracle has access to the hidden DFA and answers queries about the language it recognizes. Traditionally, there are two types of queries (Angluin, 1988; 1990). A Membership Query allows the learner to ask whether a given word is accepted by the hidden DFA, while an Equivalence Query allows the learner to propose a hypothesis DFA and test whether it is equivalent to the hidden DFA, which the oracle can either accept or refute with a counterexample – a word on which the hypothesis DFA and the hidden DFA disagree.

Prior work has established formal complexity and evaluation measures, as well as efficient deterministic active automata learning algorithms (Angluin, 1987; Rivest & Schapire, 1993). Task complexity is typically characterized by the number of states in the minimal DFA accepting the language of the hidden DFA, while learning efficiency is measured by the number of queries required to identify the hidden DFA (Isberner, 2015).

As can be seen on the right-hand side of Table 1, classic learners such as L* and TTT solve this task through an iterative hypothesis-refinement process: they consider a hypothesis which matches all given observations using EQ, and when a counterexample is returned, use a polynomial number of MQs to construct a new hypothesis that again accounts for all given information while adding at least one previously unidentified state in the hidden DFA. These algorithms were proven to converge on the hidden DFA in polynomial time (Angluin, 1987; Isberner et al., 2014). This exemplifies several desired agentic real-world traits, including maintaining a memory of observed behavior, reasoning over it, choosing actions based on previous interactions, and adapting to new information.

2.2 Passive Automata Learning

In passive learning, the learner does not query an oracle. Instead, it receives a fixed labeled sample of accepted and rejected words and must infer a DFA consistent with that sample (Murphy, 1996). Classic passive algorithms such as RPNI, EDSM, and Blue-Fringe learn automata in this way (Oncina & Garcia, 1992; Lang et al., 1998).

While we focus on active automata learning, we use passive learning algorithms to assess the information gathered by LLM agents during active interaction: given the labeled words collected by an agent via membership queries, we test whether a passive learner can already infer the hidden DFA.

# step	**LLM-based learners**				**Classic algorithms**			
	Gemini 3.1 Pro		DeepSeek-v4-Pro		$L^\star$		TTT	
	Tool Call	Resp.	Tool Call	Resp.	Tool Call	Resp.	Tool Call	Resp.
1	EQ($\emptyset$)	bba	MQ(ϵ)	0	MQ(b)	1	MQ(ϵ)	0
2	EQ(Σ^*)	aaa	MQ(a)	0	MQ(a)	0	EQ($\emptyset$)	bba
3	EQ($b \in w$)	$\top$	MQ(b)	1	MQ(ϵ)	0	MQ(bba)	1
4	—		MQ(aa)	0	MQ(bb)	1	MQ(ba)	1
5	—		MQ(ab)	1	MQ(ba)	1	MQ(a)	0
6	—		MQ(ba)	1	EQ($b \in w$)	$\top$	MQ(b)	1
7	—		MQ(bb)	1	—		MQ(aa)	0
8	—		MQ(aba)	1	—		MQ(baa)	1
9	—		MQ(aaa)	0	—		EQ($b \in w$)	$\top$
10	—		EQ($b \in w$)	$\top$	—		—	

Table 1: **Example interactions with an oracle** (*Resp.* column), for the hidden DFA accepting all words containing at least one b. — indicates that the learner has already solved the problem. Gemini 3.1 Pro solves the task using only equivalence queries, whereas DeepSeek-v4-Pro, $L^\star$, and TTT use both membership and equivalence queries. In EQ notation, $\emptyset$ denotes the empty language, Σ^* denotes all words, and $b \in w$ denotes the language of words containing b.

3 AGENTIC AUTOMATA LEARNING

In this section we formally define the task of *agentic automata learning*, which recasts the classic interactive automata learning for agentic tool-calling LLMs (§3). A representative example of such an interaction is shown on the left-hand side of Table 1. We then describe how task instances can be generated automatically at arbitrary scale by sampling DFAs while controlling for their complexity, and discuss how the task requires important agentic capabilities, such as adaptive planning, memory organization, and efficient tool use.

Task definition. Let Σ be a finite alphabet, $\mathcal{D}_\Sigma$ be the set of all DFAs over Σ, and $A^\star \in \mathcal{D}_\Sigma$ be a minimal DFA inducing the target language $L^\star \subseteq \Sigma^*$. An agent interacts with an oracle $\mathcal{O}_{A^\star}$ through a sequence of tool calls, each termed a query.

At each time step $t = 1, \ldots, T$, the agent chooses one of two tool calls: *(1) Membership Query*, $\mathrm{MQ} : \Sigma^* \mapsto \{0, 1\}$ for some word $w \in \Sigma^*$, to which the oracle returns $\mathbb{1}_{L^\star}\{w\}$. *(2) Equivalence Query*, $\mathrm{EQ} : \mathcal{D}_\Sigma \mapsto \{\top\} \cup \Sigma^*$ for some hypothesis DFA A, to which the oracle returns either $\top$ if $L(A) = L^\star$, or a counterexample $w \in L(A) \triangle L^\star$ if they differ.

The agent's prompt at time t contains the entire interaction history:

$$h_{t-1} = ((a_1, o_1), \ldots, (a_{t-1}, o_{t-1})),$$

where a_i is the query and o_i the oracle response at step i. See prompt examples in Appendix A.2.

The goal of the agent is to identify the target language by issuing an equivalence query $\mathrm{EQ}(\hat{A})$ such that:

$$L(\hat{A}) = L^\star.$$

An interaction is successful if the oracle returns $\top$ for an equivalence query within a query budget T, while its cost is the number of queries issued up to and including the accepted equivalence query.

The complexity of an interaction is defined by the number of states in $A^\star$. This is motivated by the classic active automata learning problem, where the number of states is a central factor affecting the efficiency of algorithms such as L* and TTT.

Automatic generation of instances. A key advantage of agentic automata learning is that task instances can be generated procedurally and at arbitrary scale, while controlling for their complexity through properties of the target DFA, such as the number of states. In particular, we use the Boltzmann sampling method, which samples uniformly from the space of accessible DFAs over a binary alphabet,[1] and then applies rejection sampling to filter out non-minimal DFAs (Bassino & Nicaud, 2007). While in our experiments we keep the alphabet fixed, this sampling procedure itself does not inherently restrict the alphabet size. This procedural construction allows us to create arbitrarily many environments over a chosen alphabet, rather than relying on a fixed pool of predefined tasks or examples requiring expensive human annotation.

Agentic automata learning provides a controlled, simple, yet challenging environment for evaluating core agentic capabilities. We intentionally simplify the environment, scaffold, and harness to isolate the model's inherent limitations while retaining the core demands of interactive problem solving. To discover the latent structure defined by a hidden DFA, an agent must combine adaptive planning, reasoning from observations, memory organization, and tool use. Because each query produces new feedback, the agent must select subsequent queries that reduce uncertainty about the target DFA, infer structural properties from the accumulated history of accepted and rejected words, and maintain consistency with prior observations by avoiding contradictory hypotheses and redundant queries. Although DFAs are less complex than many real-world environments, this controlled setting makes failures in these capabilities easier to identify and analyze. Moreover, the minimal harness and toolset limit the extent to which agents can compensate for weaknesses in parametric reasoning through external support.

[1] A DFA is accessible if every state can be reached from the initial state by some input word.

4 EXPERIMENTAL SETUP

In this section, we describe the different design choices and evaluation criteria used to assess LLMs on the agentic automata learning task. Additional details on the experimental setup are provided in Appendix A.

Stateless vs. stateful agent scaffolds. We evaluate a multi-turn agent under two scaffolds, which we refer to as *stateless* and *stateful* scaffolds. In the *stateless scaffold*, at each step, the model receives the full history of previous actions and environment feedback and is equipped with membership and equivalence query tools for obtaining new information from the environment. The *stateful scaffold* adds a ReAct-style state-tracking mechanism (Yao et al., 2023; Rozanov & Rei, 2025). As illustrated in Figure 2, its output format requires the model at every step to explicitly maintain its current goal, a compact working-memory representation of the accumulated state, and the reasoning used to select the next tool call. These fields are carried forward and updated across interaction steps, allowing intermediate structure to be preserved rather than repeatedly recovered from the interaction history.

Models and metrics. We evaluate three advanced models, with their reasoning configuration set to high: DeepSeek-V4-Pro (DeepSeek-AI, 2026), a 1.6T-parameter model, Gemini 3.1 Pro Preview (Google DeepMind, 2026a), and Gemini 3.8 Flash (Google DeepMind, 2026b). We selected these models to cover frontier-level reasoning capabilities while keeping the full evaluation computationally feasible across our experimental setup. We did not include OpenAI or Claude models because evaluating their high-reasoning configurations at the same scale would have substantially increased the total API cost.

We evaluate two aspects of model performance. First, *Success Rate* measures the rate at which models successfully identify a hidden DFA:

$$SuccessRate = \frac{1}{N}\sum_{i=1}^{N} Succ(M, x_i), \tag{1}$$

where $Succ(M, x_i)$ is a binary indicator of whether model M successfully infers the hidden DFA x_i, and N is the number of sampled automata.

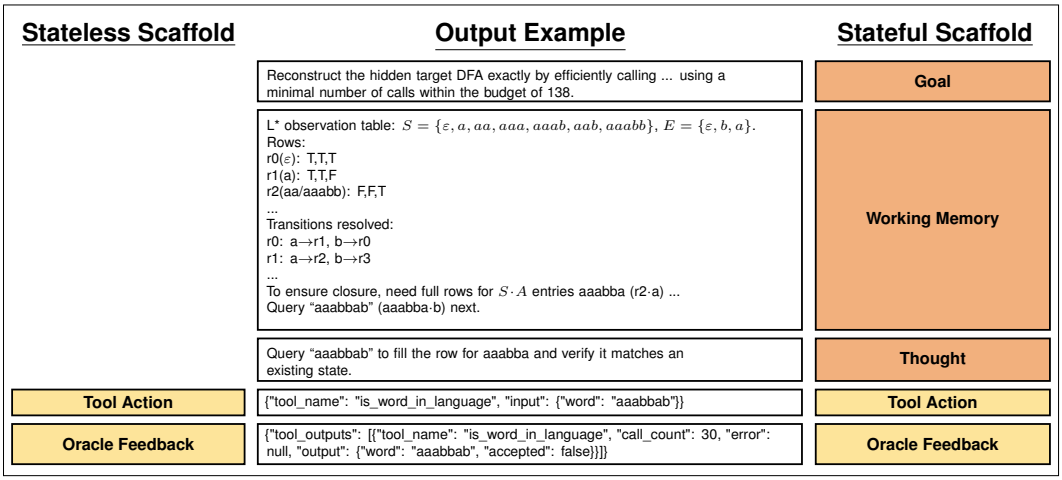


Figure 2: **Comparison of the stateless (left) and stateful (right) scaffolds along with an output example (middle).** At each step, the stateless scaffold outputs the next tool action directly from the interaction history, whereas the stateful scaffold additionally exposes an explicit goal, working memory, and thought state. Orange components are unique to the stateful scaffold, while yellow components are shared by both scaffold types. The center panel presents an excerpt from a stateful DeepSeek run, in which the model maintains an L^* observation table in working memory and uses it to select the next query.

Second, we measure *interaction efficiency* by the difference between the number of tool calls made by a model M when it finds the hidden DFA and the minimum number of tool calls made by the deterministic automata-learning algorithms L^* and TTT Angluin (1987); Isberner et al. (2014):

$$\Delta\textit{Tool Calls from Deterministic Algorithms} = \frac{1}{|S_{\text{succ}}|} \sum_{i \in S_{\text{succ}}} \left(T_M(x_i) - \min\left\{T_{L^*}(x_i), T_{\text{TTT}}(x_i)\right\}\right), \quad (2)$$

where S_{succ} denotes successful runs, and $T_A(x_i)$ denotes the number of tool calls used by algorithm A to identify the hidden DFA x_i.

Query budget. The number of queries allowed in each instance is defined to be at most twice the number of queries required by the better of L^* and TTT for the corresponding DFA, both of which have provable upper bounds on the number of queries. In particular, L* uses $O(|\Sigma|n^2m)$ queries (Angluin, 1987), while TTT uses $O(|\Sigma|n^2 + n \log m)$ queries (Isberner et al., 2014), where n is the number of states in the minimal hidden DFA, $|\Sigma|$ is the alphabet size, and m is the maximum counterexample length. Defining the budget as twice the number of queries guarantees that a correct solution exists within the allocated query budget. Throughout the evaluation, we discard any malformed tool call and do not count it against the query budget.

Dataset construction. We sample an evaluation dataset consisting of 160 task instances evenly distributed into 4 complexity levels, defined by the number of states in the minimal DFA: 2–3, 4–5, 6–7, and 8–9. Although additional instances can be generated automatically, evaluation remains expensive in runtime and monetary cost. We decide to evaluate our models on 160 instances to base our conclusions on a larger set of tasks and reduce dependence on the outcomes of individual instances, while keeping the evaluation feasible. This scale is also consistent with other benchmarks for long-horizon agent tasks, such as AgencyBench, which contains 138 instances (Li et al., 2026). To quantify uncertainty around the performance estimates and assess the stability of the observed trends with respect to variation in the sampled instances, we report 95% confidence intervals computed using bootstrap with 100,000 resamples (Hesterberg, 2015).

Oracle counterexample selection strategy. For each equivalence query, if the proposed DFA is not equivalent to the hidden DFA, the oracle must return a counterexample (a word on which the two DFAs disagree), yet the classic task formulation does not define *which* counterexample it should return. We return a counterexample designed to balance two considerations. On one hand, the length of the minimal counterexample directly affects the runtime of the classic learning algorithms and, consequently, the query budget allocated to the model and the overall evaluation cost. On the other hand, a minimal counterexample provides additional implicit information. Intuitively, if the oracle consistently returns a minimal counterexample of length n, then the two DFAs agree on all words of length $< n$, allowing the agent to infer their membership without explicitly querying them. We therefore sample a short counterexample at random.

5 RESULTS

Figure 3 reports *Success Rate* which measures whether models identify the hidden DFA, and Δ *Tool Calls from Deterministic Algorithms* which measures interaction efficiency. We complement this analysis by differentiating between planning and reasoning errors and examining non-informative query rates in Figure 4. Together, these results characterize task difficulty and reveal systematic differences both across models and across scaffolds. We first highlight the main findings, then turn to a detailed error and cost analysis.

***Agentic automata learning* is challenging for state-of-the-art models in both stateless and stateful scaffolds** . Figure 3 shows that, under both stateless and stateful scaffolds, the performance of all models generally declines as the number of states in the minimal DFA increases. For automata with 8 or 9 states, the highest success rate is 57.5% under the stateful scaffold and 30.0% under the stateless scaffold. The gap relative to classic algorithms is also reflected in interaction efficiency: in the same 8–9-state range, when considering only successful runs, even the most efficient model-scaffold

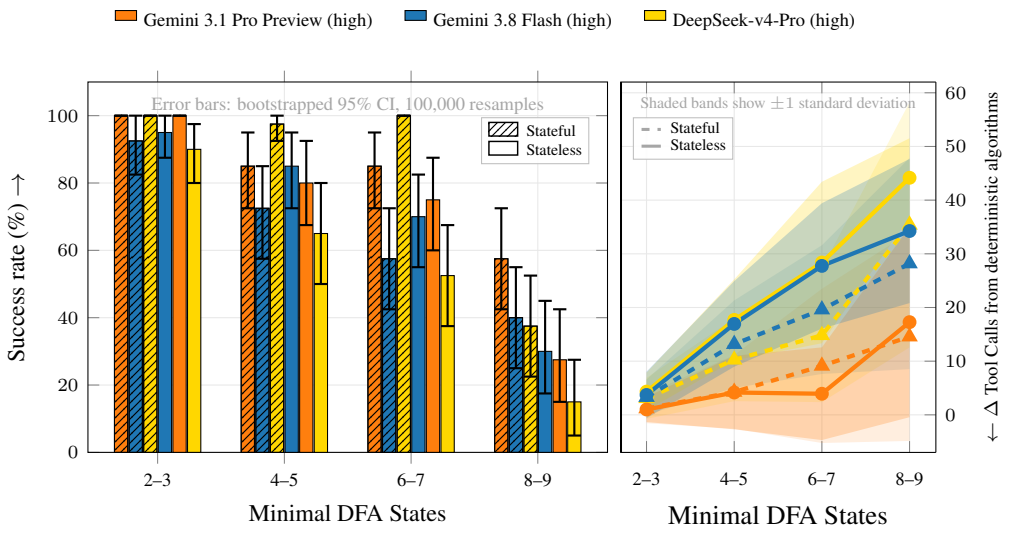


Figure 3: **Model performance and query efficiency as the number of states in the minimal DFA increases.** Left: success rate for DFA-state buckets 2–3, 4–5, 6–7, and 8–9, with $n = 40$ runs for each bucket. Right: mean Δ Tool Calls relative to deterministic algorithms (L^*, TTT). Positive values indicate more tool calls than the more query-efficient of classic strategies.

combination, Gemini 3.1 Pro with a stateful scaffold, uses approximately 23.5% more tool calls than TTT.

Stateful scaffolding generally improves performance over stateless scaffolding on the agentic automata learning task and changes the relative ranking of models. Figure 3 shows that the stronger models, DeepSeek-v4-Pro and Gemini 3.1 Pro, benefit substantially from the stateful scaffold, both in success rate and interaction efficiency. One possible explanation is that the agent does not need to repeatedly re-integrate all accumulated information at every step. As illustrated in Figure 2, the model maintains an observation table similar to that used by the classic L^* algorithm in its working memory. Based on this table, it chooses to query whether the word *"aaabbab"* belongs to the target language in order to fill in a missing table entry. In contrast, the smaller Gemini 3.8 Flash model is harmed by the addition of state. This may be since the more complex structured output format makes it harder for the model to consistently satisfy the scaffold requirements. This interpretation is supported by the increase in its invalid tool-call rate from 0.2% to 8.7% when state tracking is added.

LLM agents identify L^* as the appropriate solution, but struggle to execute it reliably over long sequences. In a complementary evaluation, we give the models access to coding tools through the OpenHands harness (Wang et al., 2025) without Internet access, and evaluate whether they can identify prior work on automata learning and reproduce a classic algorithm for the task. *All examined models identify and write the code for the classic L^* algorithm*. Thus, the lack in performance does not stem from a lack of knowledge of the classic algorithms or from failing to recognize them as an appropriate solution. However, this knowledge does not translate into reliable execution during interaction: since L^* is deterministic, we can compare the sequence of queries generated by a model with the sequence generated by each algorithm on the same task instance. We find no task instance in which a model exactly reproduces the same query sequence. In addition, when solving the task using these algorithms, the number of equivalence queries is bounded by the number of states in the hidden DFA, and the size of hypotheses submitted through EQs increases monotonically throughout the learning process. Every model–scaffold combination violates at least one of these properties in a substantial fraction of runs. Even the combination closest to classic behavior, DeepSeek-v4-Pro with a stateful scaffold, satisfies both properties in only 31% of runs. This conclusion is consistent with prior findings which showed that LLMs struggle to reliably execute algorithms over long sequences of interactive steps, even when the algorithm is provided to them (Yang et al., 2025).

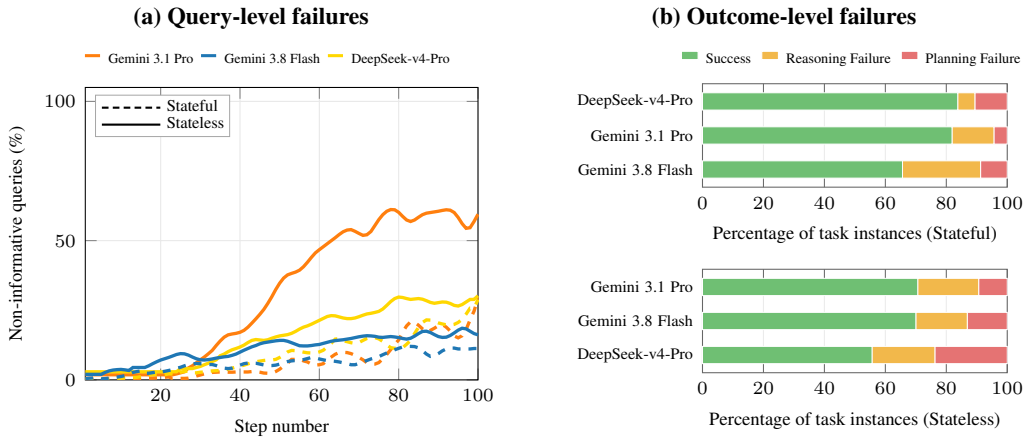


Figure 4: **Error analysis at two levels: query level on the left and task-instance level on the right.** The left panel shows the percentage of queries at each interaction step that are non-informative across all task instances, for each model and scaffold. Curves are smoothed using a 1D Gaussian filter ($\sigma = 2$) for visualization. We report values up to step 100, as only a small number of task instances continue beyond this point. The right panel shows the percentage of task instances classified as successful runs, reasoning failures, or planning failures.

5.1 Error Analysis

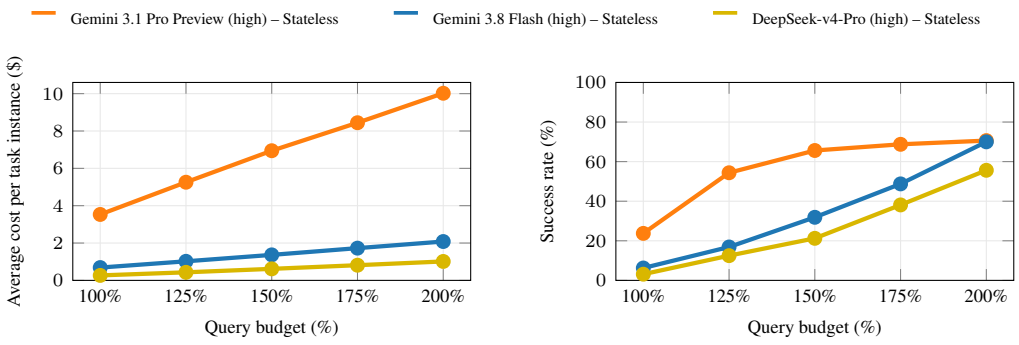
We conduct several analyses to better understand the performance of the models, shed light on which aspects of the task are challenging, and outline avenues for future research.

Models fail both in planning which queries to make and in inferring structure from available evidence. We classify model failures into two mutually exclusive categories: planning failures and reasoning failures. A *planning failure* occurs when the model fails and none of the classic passive learning algorithms, RPNI, EDSM, or Blue-Fringe, introduced in Section 2.2, can infer the hidden DFA from the observations accumulated by the model (Oncina & Garcia, 1992; Lang et al., 1998). In this case, we assume that the model did not collect sufficient information during interaction. In contrast, a *reasoning failure* occurs when the model fails, but at least one passive learning algorithm can infer the hidden DFA from the accumulated observations. Thus, the necessary information was available, but the model failed to infer the correct DFA from it. Figure 4 shows that all evaluated models suffer from both reasoning and planning failures. In general, failure rates are lower under the stateful scaffold, suggesting that explicit state tracking generally improves the models' ability to maintain progress through the interaction process.

All tested LLMs increasingly make non-informative queries as interactions progress, while stateful scaffolding reduces but does not eliminate this behavior. We measure how consistently LLMs use the information accumulated during interaction through the notion of *non-informative queries*: tool calls whose answers can be implied by previous oracle responses and therefore cannot provide new evidence. For membership queries, a query is non-informative if the same word has already been queried. For equivalence queries, a query is non-informative if it duplicates a previous hypothesis, or if the proposed DFA is already contradicted by known evidence, i.e., there exists a previously queried word that the hypothesis classifies differently from the oracle. Figure 4 shows that the rate of non-informative queries increases as interactions grow longer, suggesting that as evidence accumulates in context, LLM agents increasingly struggle to organize and use it consistently, while explicit state tracking helps mitigate this difficulty.

5.2 Cost Analysis

A central question in current LLM evaluation is the computational and monetary cost. This issue is especially important for agents, where each instance may involve a long sequence of interactions. Figure 9 shows runtime and cost increase with DFA size, reaching up to about 16 million tokens, 20



(a) Average cost per instance.

(b) Success rate.

Figure 5: **Effect of the query budget on evaluation cost and success rate.** The budget ranges from 100% to 200% of the number of queries required by the more efficient classic learner between $L^\star$ and TTT on the corresponding task instance.

hours, and $50 per run. Overall, the total monetary cost of the full experimental suite was $3000. The Stateful scaffold reduces cost across models because it reduces the number of reasoning tokens required at each step: the model can preserve useful intermediate information instead of repeatedly reconstructing it from the full interaction history. The largest reduction is observed for Gemini 3.1 Pro Preview, whose average cost per run decreases from $10 to $3.25.

Figure 5 shows that evaluation cost increases with the query budget, while success rates saturate at different budgets across models. We analyze the need for such a budget and find that additional budget is critical for solving DFAs that models cannot recover under smaller budgets. Specifically, DeepSeek-v4-Pro improves from 3.1% at 100% to 55.6% at 200%, with each increase in budget yielding a substantial improvement of similar magnitude in success rate. In contrast, Gemini 3.1 Pro Preview reaches 54.4% success already at 125% of the baseline budget, after which the gains taper, reaching 70.6% at 200%.

6 Related Work

Recent benchmarks test whether LLM agents can reconstruct hidden mechanisms from documentation and interactive black-box access (Yang et al., 2026; Rahmani et al., 2026; Lehmann et al., 2026), but fixed query budgets need not make every instance identifiable, and real-world complexity obscures sources of failure. AQA-Bench tests whether LLMs can maintain state and execute a predefined algorithm across successive interactions (Yang et al., 2025), while work on in-context learning of hidden Markov models and probabilistic automata evaluates non-interactive behavioral approximation rather than exact structural recovery (Akyürek et al., 2024; Dai et al., 2025). Other work uses classic algorithms such as L* to extract DFAs from trained neural models serving as oracles (Weiss et al., 2018; Chen et al., 2024; Vazquez-Chanlatte et al., 2025). Our framework instead makes the LLM the learner: it must recognize and retrieve an appropriate algorithm, adaptively query an external oracle, and execute the algorithm to reconstruct the hidden DFA exactly. Deriving query budgets from classic learning algorithms ensures every instance is solvable within its budget and supports controlled, fine-grained analysis of information gathering, planning, and reasoning.

7 Conclusion

We introduce *agentic automata learning*, a controlled framework that models environments as hidden DFAs and uses membership and equivalence queries to evaluate both whether language models recover hidden structure and how they do so. Even strong models struggle as complexity increases, with failures in accuracy, planning, reasoning, tool use, and managing accumulated information. These results motivate evaluations of how agents search, update beliefs, and use feedback over time. Future work could extend the framework to nondeterministic or stochastic environments and use the framework for training agents that better interact with their environment.

Reproducibility Statement

We provide detailed information to facilitate reproduction of our experiments. The task formulation, procedural DFA generation, evaluation metrics, and query-budget construction are described in the main text. Additional experimental details are provided in Appendix A, including the complete prompts for both the Stateless and Stateful scaffolds, an example interaction, exact model versions and inference settings, evaluated task instances, and the deterministic counterexample-sampling procedure. Our evaluation uses seeds 1–20 for each minimal DFA size from 2 to 9, and the query budget for each instance is set to twice the number of queries required by the better of L^* and TTT. We provide the full anonymized codebase, together with the experimental configuration, as part of the supplementary materials. Upon publication, we will release the code publicly as open source.

AI Use Statement

Generative AI tools were used during the preparation of this work to assist with translation, language editing, software implementation and code editing, LaTeX formatting, and the presentation of experimental results. Generative AI was not used to generate datasets, formulate mathematical claims or proofs, propose research hypotheses, design the research methodology or experiments, or interpret the experimental results. All AI-assisted text and code were reviewed by the authors, and AI-assisted code was tested for correctness. All experimental results and scientific claims were independently verified by the authors. The authors take full responsibility for the final content of this work, including any text, code, or other artifacts produced with the assistance of generative AI.

References

Ekin Akyürek, Bailin Wang, Yoon Kim, and Jacob Andreas. In-context language learning: Architectures and algorithms. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 787–812. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/akyurek24a.html.

Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987. doi: 10.1016/0890-5401(87)90052-6. URL https://doi.org/10.1016/0890-5401(87)90052-6.

Dana Angluin. Queries and concept learning. *Mach. Learn.*, 2(4):319–342, 1988. doi: 10.1007/BF00116828. URL https://doi.org/10.1007/BF00116828.

Dana Angluin. Negative results for equivalence queries. *Mach. Learn.*, 5:121–150, 1990. doi: 10.1007/BF00116034. URL https://doi.org/10.1007/BF00116034.

Frédérique Bassino and Cyril Nicaud. Enumeration and random generation of accessible automata. *Theor. Comput. Sci.*, 381(1-3):86–104, 2007. doi: 10.1016/J.TCS.2007.04.001. URL https://doi.org/10.1016/j.tcs.2007.04.001.

Lekai Chen, Ashutosh Trivedi, and Alvaro Velasquez. Llms as probabilistic minimally adequate teachers for DFA learning. *CoRR*, abs/2408.02999, 2024. doi: 10.48550/ARXIV.2408.02999. URL https://doi.org/10.48550/arXiv.2408.02999.

Walter Daelemans. Colin de la higuera: Grammatical inference: learning automata and grammars - cambridge university press, 2010, iv + 417 pages. *Mach. Transl.*, 24(3-4):291–293, 2010. doi: 10.1007/S10590-011-9086-9. URL https://doi.org/10.1007/s10590-011-9086-9.

Yijia Dai, Zhaolin Gao, Yahya Sattar, Sarah Dean, and Jennifer J. Sun. Pre-trained large language models learn to predict hidden markov models in-context. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30*

- *December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/29a0ea49a103a233b17c0705cdeccb66-Abstract-Conference.html.

DeepSeek-AI. DeepSeek V4 Preview Release. https://api-docs.deepseek.com/news/news260424/, April 2026. Accessed: 2026-08-04.

Google DeepMind. Gemini 3.1 Pro Model Card. Technical report, Google DeepMind, February 2026a. URL https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-1-Pro-Model-Card.pdf.

Google DeepMind. Gemini 3.8 Flash Model Card. Technical report, Google DeepMind, September 2026b. URL https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-8-Flash-Model-Card.pdf.

Tim C Hesterberg. What teachers should know about the bootstrap: Resampling in the undergraduate statistics curriculum. *The american statistician*, 69(4):371–386, 2015. URL https://arxiv.org/abs/1411.5279.

Malte Isberner. *Foundations of active automata learning: an algorithmic perspective*. PhD thesis, Technical University Dortmund, Germany, 2015. URL https://hdl.handle.net/2003/34282.

Malte Isberner, Falk Howar, and Bernhard Steffen. The TTT algorithm: A redundancy-free approach to active automata learning. In Borzoo Bonakdarpour and Scott A. Smolka (eds.), *Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings*, Lecture Notes in Computer Science, pp. 307–322. Springer, 2014. doi: 10.1007/978-3-319-11164-3_26. URL https://doi.org/10.1007/978-3-319-11164-3_26.

Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In Vasant G. Honavar and Giora Slutzki (eds.), *Grammatical Inference, 4th International Colloquium, ICGI-98, Ames, Iowa, USA, July 12-14, 1998, Proceedings*, Lecture Notes in Computer Science, pp. 1–12. Springer, 1998. doi: 10.1007/BFB0054059. URL https://doi.org/10.1007/BFb0054059.

Jens Lehmann, Andrei Aioanei, and Sahar Vahdati. Tycho: Active abstraction with programmatic world models for ARC-AGI-3. *CoRR*, abs/2607.28287, 2026. doi: 10.48550/ARXIV.2607.28287. URL https://doi.org/10.48550/arXiv.2607.28287.

Keyu Li, Junhao Shi, Yang Xiao, Mohan Jiang, Jie Sun, Yunze Wu, Dayuan Fu, Shijie Xia, Xiaojie Cai, Tianze Xu, Weiye Si, Wenjie Li, Dequan Wang, and Pengfei Liu. Agencybench: Benchmarking the frontiers of autonomous agents in 1m-token real-world contexts. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 7422–7440. Association for Computational Linguistics, 2026. doi: 10.18653/V1/2026.ACL-LONG.337. URL https://doi.org/10.18653/v1/2026.acl-long.337.

Kevin P. Murphy. Passively learning finite automata. Technical Report UCB/CSD-96-917, University of California, Berkeley, 1996. URL https://sfi-edu.s3.amazonaws.com/sfi-edu/production/uploads/sfi-com/dev/uploads/filer/7f/cf/7fcf81e6-a6de-4928-9288-9bd22e50f805/96-04-017.pdf.

Jose Oncina and Pedro Garcia. Inferring regular languages in polynomial update time. In *Pattern Recognition and Image Analysis*, pp. 49–61, 1992. URL https://www.researchgate.net/publication/239643375_Inferring_regular_languages_in_polynomial_update_time/link/54044cdb0cf2c48563b0705c/download?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6InB1YmxpY2F0aW9uIiwicGFnZSI6InB1YmxpY2F0aW9uIn19.

Dongmin Park, Minkyu Kim, Beongjun Choi, Junhyuck Kim, Keon Lee, Jonghyun Lee, Inkyu Park, Byeong-Uk Lee, Jaeyoung Hwang, Jaewoo Ahn, Ameya Sunil Mahabaleshwarkar, Bilal Kartal, Pritam Biswas, Yoshi Suhara, Kangwook Lee, and Jaewoong Cho. Orak: A foundational benchmark for training and evaluating LLM agents on diverse video games. *CoRR*, abs/2506.03610, 2025. doi: 10.48550/ARXIV.2506.03610. URL https://doi.org/10.48550/arXiv.2506.03610.

Babak Rahmani, Sebastian Dziadzio, Joschka Strüber, Sergio Hernández-Gutiérrez, and Matthias Bethge. Revengebench: Reverse engineering code-space policies from behavioral experiments. *CoRR*, abs/2606.26094, 2026. doi: 10.48550/ARXIV.2606.26094. URL https://doi.org/10.48550/arXiv.2606.26094.

Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Inf. Comput.*, 103(2):299–347, 1993. doi: 10.1006/INCO.1993.1021. URL https://doi.org/10.1006/inco.1993.1021.

Nikolai Rozanov and Marek Rei. StateAct: Enhancing LLM base agents via self-prompting and state-tracking. In Ehsan Kamalloo, Nicolas Gontier, Xing Han Lu, Nouha Dziri, Shikhar Murty, and Alexandre Lacoste (eds.), *Proceedings of the 1st Workshop for Research on Agent Language Models (REALM 2025)*, pp. 367–385, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-264-0. doi: 10.18653/v1/2025.realm-1.27. URL https://aclanthology.org/2025.realm-1.27/.

Keyon Vafa, Justin Y. Chen, Ashesh Rambachan, Jon M. Kleinberg, and Sendhil Mullainathan. Evaluating the world model implicit in a generative model. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/2f6a6317bada76b26a4f61bb70a7db59-Abstract-Conference.html.

Marcell Vazquez-Chanlatte, Karim Elmaaroufi, Stefan J. Witwicki, Matei Zaharia, and Sanjit A. Seshia. L*lm: Learning automata from demonstrations, examples, and natural language. In George J. Pappas, Pradeep Ravikumar, and Sanjit A. Seshia (eds.), *International Conference on Neuro-symbolic Systems, 28-30 May 2025, University of Pennsylvania, Philadelphia, Pennsylvania, USA*, Proceedings of Machine Learning Research, pp. 543–569. PMLR, 2025. URL https://proceedings.mlr.press/v288/vazquez-chanlatte25a.html.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, and et al. Openhands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=OJd3ayDDoF.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, Proceedings of Machine Learning Research, pp. 5244–5253. PMLR, 2018. URL http://proceedings.mlr.press/v80/weiss18a.html.

John Yang, Kilian Lieret, Jeffrey Ma, Parth Thakkar, Dmitrii Pedchenko, Sten Sootla, Emily McMilin, Pengcheng Yin, Rui Hou, Gabriel Synnaeve, Diyi Yang, and Ofir Press. Programbench: Can language models rebuild programs from scratch? *CoRR*, abs/2605.03546, 2026. doi: 10.48550/ARXIV.2605.03546. URL https://doi.org/10.48550/arXiv.2605.03546.

Siwei Yang, Bingchen Zhao, and Cihang Xie. Aqa-bench: An interactive benchmark for evaluating llms' sequential reasoning ability in algorithmic environments. *Trans. Mach. Learn. Res.*, 2025, 2025. URL https://openreview.net/forum?id=W22g6Ksmbi.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Junhao Zheng, Xidi Cai, Qiuke Li, Duzhen Zhang, Zhong-Zhi Li, Yingying Zhang, Le Song, and Qianli Ma. Lifelongagentbench: Evaluating LLM agents as lifelong learners. *CoRR*, abs/2505.11942, 2025. doi: 10.48550/ARXIV.2505.11942. URL https://doi.org/10.48550/arXiv.2505.11942.

A Experimental Setup

This appendix provides additional details on the experimental setup used throughout our evaluation. We first present the initial prompt templates used in the stateless and stateful settings, followed by an example interaction illustrating the expected interaction format. We then introduce a more general evaluation metric that goes beyond a binary notion of success or failure, and describe the procedure used to sample counterexamples. Finally, we analyze invalid tool calls, provide the model configurations used in our experiments, and examine the context-window size throughout the interaction.

A.1 Initial Prompt Templates

As described in the main text, we evaluate the agents under two scaffolds: *stateless*, where the model receives the full interaction history at each step, and *stateful*, which additionally introduces an explicit state-tracking mechanism. Below, we provide the complete prompt templates used for both scaffolds.

A.1.1 Stateless Scaffold

```
You are a language model tasked with reconstructing an unknown Deterministic Finite Automaton (DFA).
A fixed target DFA exists, but its structure is hidden from you.

You can learn about the target DFA only through:
- External tools available to you.
- Your own internal reasoning.

Game rules:
- You have a total budget of {MAX_CALLS} budgeted DFA-learning tool calls.
- If you exceed the allowed number of budgeted DFA-learning calls, your final score is 0.
- On your final remaining budgeted DFA-learning call, you must invoke the evaluation tool.

Scoring policy:
- Using fewer tool calls while still reconstructing the correct DFA yields a higher score. Therefore, tools
↪  should be invoked efficiently and purposefully to obtain novel and informative results.
- If you believe you have already identified the correct DFA structure, you should invoke immediately the
↪  evaluation tool,
  even if you still have remaining calls available.

STRICT OUTPUT POLICY (MANDATORY):
- You MUST NOT output any explanations, reasoning, comments, or natural language text.
- You MUST output ONLY TOOL_ACTION blocks (Do NOT output any text) and exactly ONE TOOL_ACTION block.
- Each TOOL_ACTION block must contain a single JSON object with exactly:
  {"tool_name": "<tool_name>", "input": { ... }}
- Never include call_count (it is added externally).

# IMPORTANT: The <TOOL_ACTION> block must include exactly one tool call. Each block represents a single tool
↪  invocation, so multiple tool calls--either to the same tool or to different tools--are not allowed within
↪  the same block.

Tools:

[is_word_in_language]
Input JSON: { tool_name: 'is_word_in_language', call_count: int, input: { word: str } }
Output JSON: { tool_name: 'is_word_in_language', call_count: int, error: str|null, output: { word: str,
↪  accepted: bool }|null }
Checks whether the given word is accepted by the game's DFA. If the input word is the empty string "", the
↪  returned output.word will be "ϵ" (epsilon).

[evaluate_dfa_candidate]
Input JSON: { tool_name: 'evaluate_dfa_candidate', call_count: int, input: { candidate_dfa: { states: [str],
↪  alphabet: [str], start_state: str, accept_states: [str], transitions: [[str, str, str]] } } }
Output JSON: { tool_name: 'evaluate_dfa_candidate', call_count: int, error: str|null, output: { score: float,
↪  optimal: bool, witness_word: str }|null }
Evaluates whether the provided candidate DFA is equivalent to the game's target DFA. If equivalent,
↪  output.optimal=true, output.score=1.0, and output.witness_word='' . If not equivalent,
↪  output.optimal=false, output.score=0.0, and output.witness_word is a counterexample word accepted by
↪  exactly one of the DFAs. If the witness word is the empty string, output.witness_word will be 'ϵ'. The
↪  counterexample also has a learning role: it shows where the candidate DFA fails and helps refine the next
↪  hypothesis.

Hints:
- DFA vocabulary: {a, b}
```

A.1.2 STATEFUL SCAFFOLD

```
Goal: You are a language model tasked with reconstructing an unknown Deterministic Finite Automaton (DFA).
A fixed target DFA exists, but its structure is hidden from you.

You can learn about the target DFA only through:
- External tools available to you.
- Your own internal reasoning.

Game rules:
- You have a total budget of {MAX_CALLS} budgeted DFA-learning tool calls.
- If you exceed the allowed number of budgeted DFA-learning calls, your final score is 0.
- On your final remaining budgeted DFA-learning call, you must invoke the evaluation tool.

Scoring policy:
- Using fewer tool calls while still reconstructing the correct DFA yields a higher score. Therefore, tools
↪  should be invoked efficiently and purposefully to obtain novel and informative results.
- If you believe you have already identified the correct DFA structure, you should invoke immediately the
↪  evaluation tool,
  even if you still have remaining calls available.

STRUCTURED OUTPUT POLICY (MANDATORY):
- You MUST output exactly four blocks and nothing else, in this exact order:
  1. <GOAL>...</GOAL>
  2. <WORKING_MEMORY>...</WORKING_MEMORY>
  3. <THOUGHT>...</THOUGHT>
  4. <TOOL_ACTION>...</TOOL_ACTION>
- The <GOAL> block must restate the overall task objective that must remain fixed throughout the interaction:
↪  reconstruct the hidden target DFA exactly while using the available DFA-learning tool-call budget
↪  efficiently. Keep it concise and do not place reasoning or a tool call in this block.
- The <WORKING_MEMORY> block must contain a compact, up-to-date summary of the information and intermediate
↪  state that should be preserved for future steps. Update it on every turn as a complete replacement for the
↪  previous working-memory summary.
- The <THOUGHT> block must contain the reasoning used to choose the single next action.
- The <TOOL_ACTION> block must contain a single JSON object with exactly:
  {"tool_name": "<tool_name>", "input": { ... }}
- Never include call_count; it is added externally.
- Exactly one block of each type is allowed. Do not output markdown fences, comments, headings, or text outside
↪  the four blocks.
- Tool documentation later in the prompt may show a standalone <TOOL_ACTION> snippet only to illustrate that
↪  tool's JSON arguments. Those snippets are not complete response examples. In every actual response, you
↪  must still output all four required blocks.
# IMPORTANT: The <TOOL_ACTION> block must include exactly one tool call. Each block represents a single tool
↪  invocation, so multiple tool calls--either to the same tool or to different tools--are not allowed within
↪  the same block.

Tools:

[is_word_in_language]
Input JSON: { tool_name: 'is_word_in_language', call_count: int, input: { word: str } }
Output JSON: { tool_name: 'is_word_in_language', call_count: int, error: str|null, output: { word: str,
↪  accepted: bool }|null }
Checks whether the given word is accepted by the game's DFA. If the input word is the empty string "", the
↪  returned output.word will be "ϵ" (epsilon).

[evaluate_dfa_candidate]
Input JSON: { tool_name: 'evaluate_dfa_candidate', call_count: int, input: { candidate_dfa: { states: [str],
↪  alphabet: [str], start_state: str, accept_states: [str], transitions: [[str, str, str]] } } }
Output JSON: { tool_name: 'evaluate_dfa_candidate', call_count: int, error: str|null, output: { score: float,
↪  optimal: bool, witness_word: str }|null }
Evaluates whether the provided candidate DFA is equivalent to the game's target DFA. If equivalent,
↪  output.optimal=true, output.score=1.0, and output.witness_word='' . If not equivalent,
↪  output.optimal=false, output.score=0.0, and output.witness_word is a counterexample word accepted by
↪  exactly one of the DFAs. If the witness word is the empty string, output.witness_word will be 'ϵ'. The
↪  counterexample also has a learning role: it shows where the candidate DFA fails and helps refine the next
↪  hypothesis.

Hints:
- DFA vocabulary: {a, b}
```

A.2 Example Interaction Format

We provide an example interaction illustrating the protocol under which the model operates during a task instance:

Initial Prompt: The interaction begins with the task prompt described in Appendix A.1

Assistant:

```
<TOOL_ACTION>
{ "tool_name": "is_word_in_language",
"input": { "word": "" } }
</TOOL_ACTION>
```

User:

```
<TOOL_RESULT>
{"tool_outputs": [{
"tool_name": "is_word_in_language",
"call_count": 1,
"error": null,
"output": {"word": "ϵ", "accepted": true} }]}
</TOOL_RESULT>
```

Skipped tool calls: 2–10.

Assistant:

```
<TOOL_ACTION>
{ "tool_name": "evaluate_dfa_candidate",
"input": {
"candidate_dfa": {
"states": ["q0","q1"],
"alphabet": ["a","b"],
"start_state": "q0",
"accept_states": ["q0"],
"transitions": [
["q0","a","q0"],
["q0","b","q1"],
["q1","a","q0"],
["q1","b","q1"]]}}}
</TOOL_ACTION>
```

User:

```
<TOOL_RESULT>
{"tool_outputs": [{
"tool_name": "evaluate_dfa_candidate",
"call_count": 11,
"error": null,
"output": {"score": 1.0,"optimal": true,"witness_word": null}}]}
</TOOL_RESULT>
```

Termination: The interaction ends once the evaluation tool indicates that the proposed DFA is equivalent to the hidden DFA, as reflected by "score": 1.0 and "optimal": true.

A.3 Best Hypothesis Similarity Metric

To complement the binary success metric, we define a continuous similarity measure that captures how close the best hypothesis proposed during interaction is to the hidden DFA.

For each task instance, we compute the similarity between every intermediate hypothesis proposed by the model and the hidden DFA, and report the highest similarity value achieved throughout the interaction, referred to as *Best Hypothesis Similarity*. This extends the binary success metric by providing a finer-grained measure of progress toward the hidden DFA.

Formally, for a hypothesis DFA h and a hidden DFA x, we compute a symmetric-difference-based similarity approximation over all words up to length $k = 200$:

$$\mathrm{Sim}_{\triangle,200}(h, x) = 1 - \frac{|L(h)_{\leq 200} \triangle L(x)_{\leq 200}|}{|\Sigma_{\leq 200}|}$$

Here, $L(h)_{\leq 200}$ and $L(x)_{\leq 200}$ denote the sets of words of length at most 200 accepted by the hypothesis and hidden DFAs, respectively, $\triangle$ denotes the symmetric difference between the two languages, and $\Sigma_{\leq 200}$ denotes the set of all words over the alphabet whose length is at most 200.

This formulation measures the fraction of words on which the two DFAs agree, producing a similarity score in $[0, 1]$, where higher values indicate greater behavioral agreement between the hypothesis and hidden languages. We use symmetric-difference similarity rather than Jaccard similarity because it is better aligned with the structure of our sampled DFA distribution. Since our DFA generation process produces languages in which, in expectation, a word has approximately equal probability of being accepted or rejected, the resulting language space is relatively balanced. In this setting, normalizing by the full input space $|\Sigma_{\leq 200}|$ provides a stable and directly interpretable measure of behavioral agreement across the entire observable space. In contrast, Jaccard similarity normalizes by the union of accepted words, making it more sensitive to variation in language density and less suitable for our objective of measuring agreement over the full input space. Although the number of words of length at most 200 is exponential, approximately 2^{200} for a binary alphabet, this metric can be computed efficiently using dynamic programming over the product DFA of the two DFAs. This enables counting the number of words in the symmetric difference without explicitly enumerating all possible strings.

Beyond aggregate evaluation, this metric enables tracking intermediate hypothesis similarity throughout interaction, as shown in Figure 7. This reveals whether a model gradually approaches the target DFA or instead oscillates between hypotheses without consistent and meaningful improvement over time.

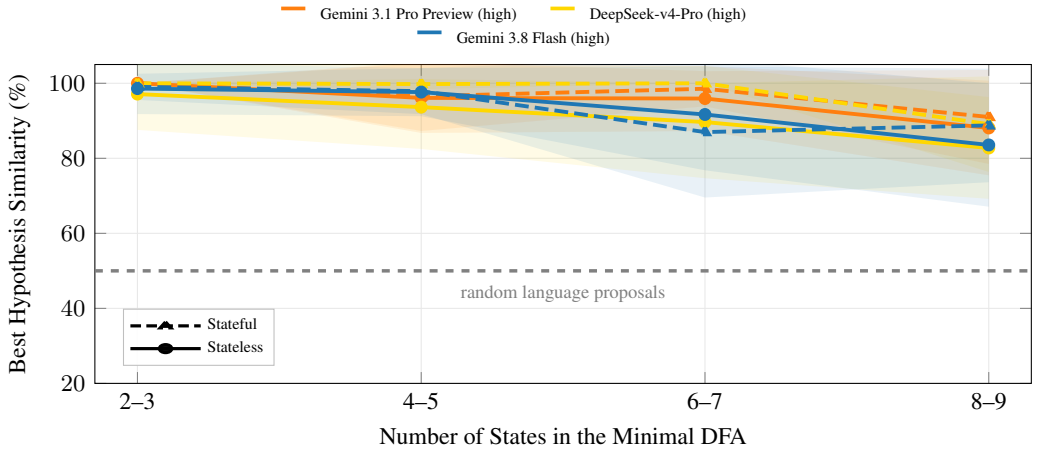


Figure 6: Best Hypothesis Similarity as the number of states in the minimal DFA increases. The shaded regions represent one standard deviation, and the dashed gray line represents a random hypothesis baseline. Stateful runs are shown with dashed curves and Stateless runs with solid curves.

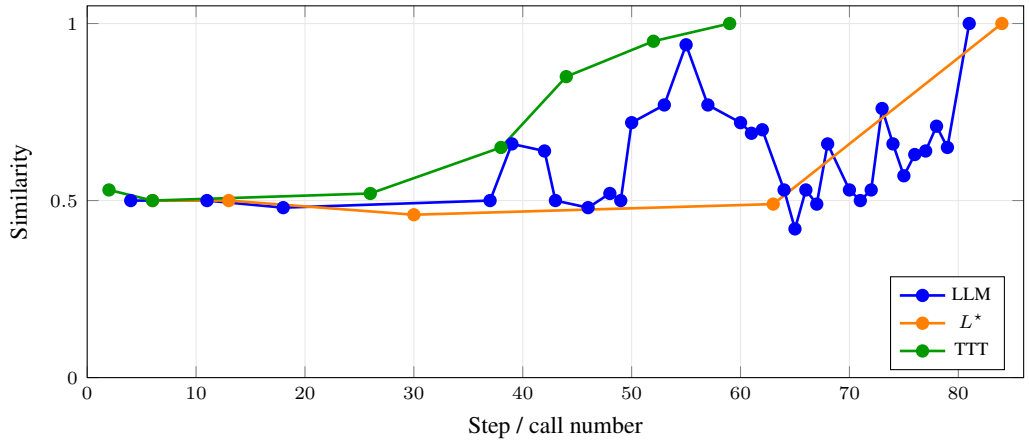


Figure 7: Example interaction trajectory for Gemini 3.1 Pro on a single task instance. The graph tracks the symmetric-difference similarity between the hidden DFA language and the hypotheses proposed throughout interaction.

A.4 How Counterexamples Are Sampled

In each equivalence query between two DFAs, we construct the symmetric difference automaton, which accepts exactly the words on which the two automata disagree. A counterexample is a word accepted by this automaton.

We first compute the minimal length $k_{\min}$ of a word accepted by the symmetric difference automaton. Instead of selecting a single word of this minimal length, we define a set of candidate words consisting of short words whose lengths lie in the range $k_{\min}$ to $k_{\min} + 3$.

The selection from this candidate set is performed in a pseudo-random yet deterministic manner. Specifically, for each given pair of automata (the target DFA and the hypothesis), we compute a fixed seed using a stable hash function over their representations. This seed is then used to select a single word from the candidate set.

This approach ensures stability by maintaining identical conditions across runs, while at the same time avoiding providing the model with additional information that could simplify the problem.

A.5 Invalid Tool-Call Analysis

LLM agents express their queries and candidate DFAs through a JSON-based tool-calling protocol, unlike L^* and TTT, which operate directly on formal objects. We identify three broad types of invalid calls: *formatting/JSON errors*, *malformed DFA specifications*, and *invalid tool or input errors*. Table 2 reports the overall percentage of invalid tool calls for each model and run type.

Model	Run	Invalid Rate
Gemini 3.8 Flash (high)	Stateful	8.7%
DeepSeek-v4-Pro (high)	Stateless	7.3%
DeepSeek-v4-Pro (high)	Stateful	4.6%
Gemini 3.1 Pro Preview (high)	Stateful	4.6%
Gemini 3.1 Pro Preview (high)	Stateless	1.3%
Gemini 3.8 Flash (high)	Stateless	0.2%

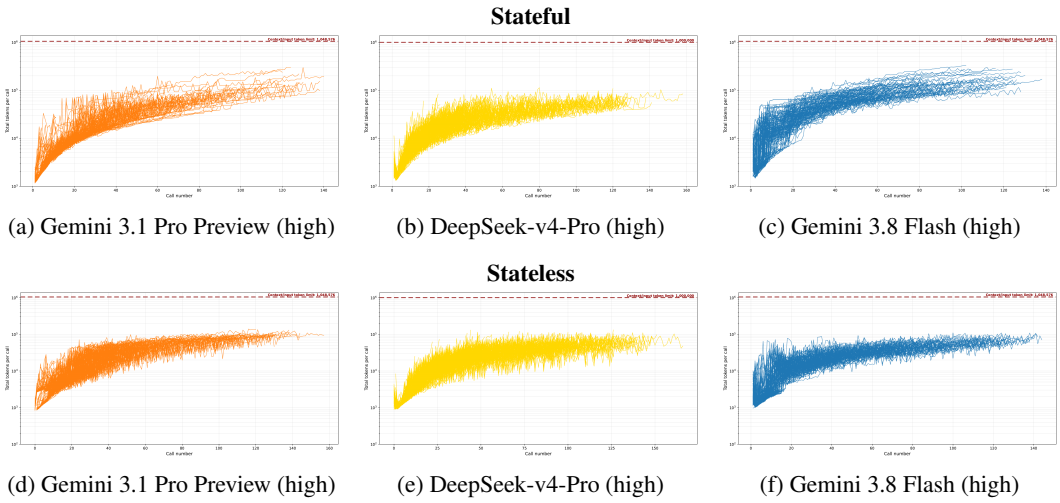
Table 2: Aggregate invalid tool-call rates by model and run type. Invalid calls include formatting or JSON errors, malformed DFA specifications, and invalid tools or input arguments.

A.6 Model Configurations

Provider	Exact model ID (click for documentation)	Run dates	Reasoning parameters	Other parameters (e.g., temperature)	Context token limit	Output token limit	Evaluated task instances
DeepSeek Platform	`deepseek-v4-pro`	May 4–7, 2026 July 11–12, 2026 September 09–10, 2026	`thinking.type=enabled` `reasoning_effort=high`	vendor-default hyperparameters	1,000,000	384,000	Seeds 1–20 for each minimal DFA size 2–9 2 distinct scaffolds (320 task instances total)
Google Cloud	`gemini-3.8-flash`	September 09–10, 2026	`thinking_level=high`	vendor-default hyperparameters	1,048,576	65,536	Seeds 1–20 for each minimal DFA size 2–9 2 distinct scaffolds (320 task instances total)
Google Cloud	`gemini-3.1-pro-preview`	May 4–7, 2026 July 11–12, 2026 September 09–10, 2026	`thinking_level=high`	vendor-default hyperparameters	1,048,576	65,536	Seeds 1–20 for each minimal DFA size 2–9 2 distinct scaffolds (320 task instances total)

Table 3: **Model configurations used in the evaluation.** Model IDs are clickable and link to the corresponding vendor documentation.

A.7 Context Window Analysis



(a) Gemini 3.1 Pro Preview (high) (b) DeepSeek-v4-Pro (high) (c) Gemini 3.8 Flash (high)

(d) Gemini 3.1 Pro Preview (high) (e) DeepSeek-v4-Pro (high) (f) Gemini 3.8 Flash (high)

Figure 8: Each line represents a single run of one task instance. The x-axis shows the tool-call number, while the y-axis shows the total number of tokens in the context window at each call on a base-10 logarithmic scale. Context-window usage throughout the interaction is shown separately for the Stateful and Stateless runs of each of the three models. The red dashed line marks the model-specific context-window limit. Since no trajectory crosses this line, the observed failures cannot be attributed to exceeding the available context window.

B Computational Cost, Runtime, and Token Usage

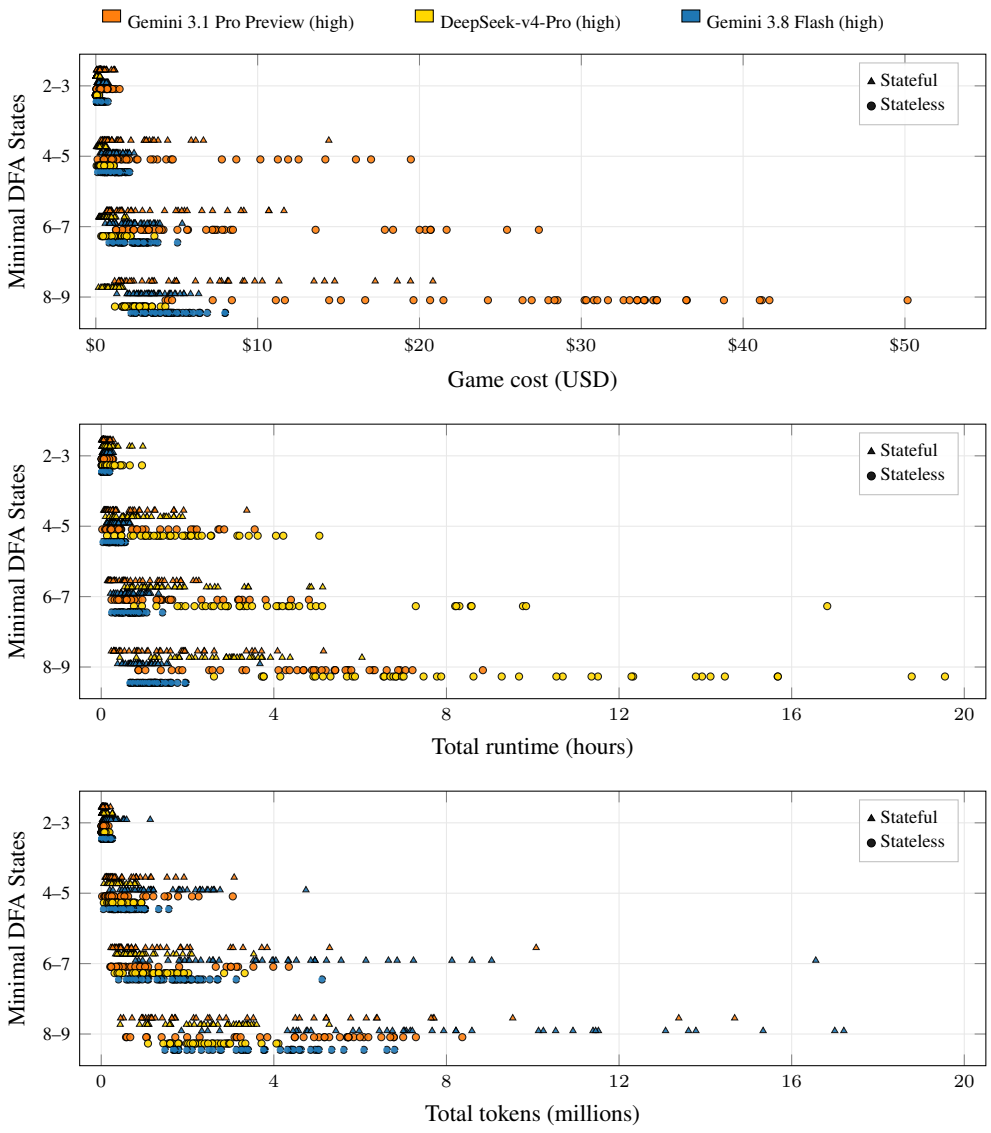


Figure 9: **Computational cost, runtime, and token usage by paired number of minimal DFA states.** Each point is a run. The y-axis groups runs into state pairs (2–3, 4–5, 6–7, and 8–9); color indicates the model, while triangles denote Stateful and circles denote Stateless interaction.