INTERNALIZED EXPERIENCE: HOW TRAINING DATA SHAPES MEMORY FORMATION IN LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Parametric memory enables large language models (LLMs) to retain information from past experience and reuse it in subsequent tasks. Yet how training data supports the internalization of reusable task rules remains unclear. We introduce **FS-Mem** (Finite-State Machine Memory), a state-transition framework for constructing training data and studying how data content and organization shape memory formation. We evaluate rule reuse across synthetic worlds, classical planning, and code execution through valid-action identification and execution-outcome prediction, complemented by next-operation prediction in code. Our experiments reveal several systematic learning patterns. Under matched training conditions, even unchanged task rules become harder to internalize when learned alongside a larger variety of action types; by contrast, scaling the training data to provide broader state–action coverage generally improves rule learning. We further find that training at a single execution depth produces depth-specialized behavior, with models performing best when training and test executions have similar lengths. Motivated by this finding, we construct cumulative corpora that combine short and long trajectories, which can outperform long-trajectory-only training while using fewer training tokens. Together, these results demonstrate that the same task rules can be internalized differently depending on how execution experience is constructed, and that understanding these learning patterns can guide the design of more effective training data for parametric memory.

1 INTRODUCTION

Large language models (LLMs) are expected to build on past experience when facing new tasks and contexts. Memory supports this reuse by preserving information beyond the context in which it was acquired (Wang et al., 2023). Existing approaches realize this capacity mainly through text-based memory, which stores information externally for later retrieval (Lewis et al., 2020; Zhong et al., 2024; Lu et al., 2023), or parametric memory, which incorporates information into model parameters through updates (Roberts et al., 2020; Meng et al., 2022a;b; Back et al., 2026). In this work, we focus on parametric memory given its potential to turn past experience into lasting improvements in model capabilities.

Existing research on LLMs' parametric memory focuses on mechanisms for encoding useful information into model parameters (De Cao et al., 2021; Mitchell et al., 2021; Charakorn et al., 2026) or dedicated parametric modules for later use (Lample et al., 2019; Wang et al., 2024b; Behrouz et al., 2026b; Wei et al., 2026). These studies have advanced our understanding of memory capacity (Allen-Zhu & Li, 2024; Xu et al., 2026), the internalization of factual information (Petroni et al., 2019; Allen-Zhu & Li, 2023; Chang et al., 2024), and reasoning over learned information (Allen-Zhu & Li, 2025; Wang et al., 2024a; Back et al., 2026). What models internalize also depends on the training data itself (Tirumala et al., 2022; Ye et al., 2026). For example, planning trajectories (Kokel et al., 2025) and program execution traces (Nye et al., 2021) have been used to study how LLMs reason about task execution. These records contain information about particular entities and their states, as illustrated in Figure 1(a). However, existing work has overlooked the fact that execution trajectories involving different entities could follow the same task rules: when an action is valid and how it changes the state, and apply across different entities and states. They are reusable across instances of a given task (Fikes & Nilsson, 1971; Ghallab et al., 1998; Yang et al., 2007). How the

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.

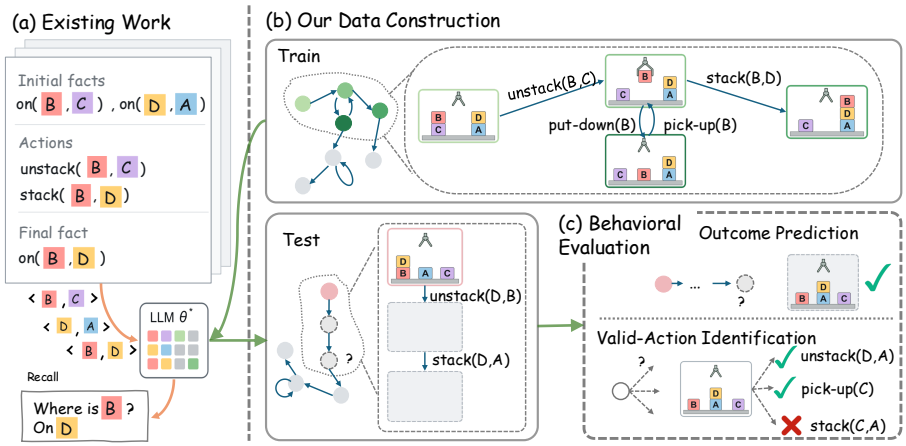


Figure 1: An illustration of FSMem in Blocksworld. (a) Recalling facts about specific entities. (b) Constructing training data from execution trajectories. (c) Identifying valid actions and predicting execution outcomes on held-out instances, without access to training records.

task content and organization of these records facilitate the internalization of such reusable rules remains less well understood.

Studying the internalization of task rules requires distinguishing which rules training data contains from how their executions are recorded. In this work, we introduce **FSMem** (**F**inite-**S**tate **M**achine **Mem**ory), a framework that makes this distinction explicitly by representing task execution as a finite-state machine (Section 2). For the Blocksworld example in Figure 1(b), states describe the configuration of blocks and the gripper's status, while concrete actions manipulate particular blocks. The transition function specifies whether an action is valid and, if so, the resulting state; trajectories record sequences of concrete actions and states. This representation allows us to vary the action types selected for supervision and, with the rules fixed, the state–action coverage and depth of execution records.

Consistent with our understanding of parametric memory, past experience should induce lasting changes in model parameters that influence subsequent behavior (De Cao et al., 2021; Charakorn et al., 2026). Compared with existing work centered on task-level results (Charakorn et al., 2026; Back et al., 2026), FSMem characterizes this influence through two complementary behaviors: **identifying valid actions** and **predicting execution outcomes**, as illustrated in Figure 1(c). These behaviors reflect the two aspects of task rules encoded by the transition function: when an action can be executed and how execution changes the state. They therefore connect the experience represented in training data to the subsequent behavior that rule internalization should support.

We instantiate FSMem in synthetic worlds, classical planning, and code execution (Section 3), and study how training-data content and organization shape parametric memory. Our study of **action complexity** shows that, under matched training conditions, task rules can become harder to internalize when learned alongside more action types (Section 4). Expanding **trajectory breadth** generally supports more effective rule reuse without increasing the number of training steps (Section 5). Our study of **depth** reveals depth-specialized behavior under single-depth training: models tend to perform better at execution depths represented in training. Based on this observation, training on a cumulative corpus of short and long trajectories can outperform long-only one with fewer tokens (Section 6).

2 Method

Our goal is to understand how training-data content and organization affect memory formation in LLMs. To this end, we introduce **FSMem**, a state-transition framework for constructing training data. We first formalize task rules and execution trajectories and define action complexity, trajectory breadth, and depth (Section 2.1). Building on this formulation, we describe how to construct supervised training examples from execution records (Section 2.2).

2.1 A FINITE-STATE MACHINE FRAMEWORK

To study how training data supports the learning of reusable execution rules, we need to specify both the rules underlying the data and the particular executions recorded. We use a finite-state machine to make this execution process explicit: its transition function specifies action validity and state changes, while its trajectories provide records for constructing training examples. This representation allows us to vary the action types selected for supervision or, with the rules fixed, change the state–action coverage and depth of execution records.

Finite-State Worlds To construct execution records under specified rules, we model each task instance as a deterministic finite-state machine, with configurations as states and concrete actions as inputs. The task world is specified by

$$\Omega = (\mathcal{E}, \mathcal{A}, \mathcal{S}, F), \tag{1}$$

where $\mathcal{E}$ is a finite set of typed entities, $\mathcal{A}$ contains the action types under study, $\mathcal{S}$ is the finite state space, and F is the transition function.

A state s assigns values to state variables describing entity attributes and relations. Let $\mathcal{V}$ denote a finite set of these variables, each with a finite value domain $\mathcal{D}_v$. The state space consists of admissible configurations:

$$\mathcal{S} \subseteq \prod_{v \in \mathcal{V}} \mathcal{D}_v, \qquad s = (s_v)_{v \in \mathcal{V}}. \tag{2}$$

Actions and Transitions The action set $\mathcal{A}$ specifies which operation types are represented in the training data. Instantiating these types with entity arguments yields concrete actions, whose validity and execution outcomes are determined by F. This separation of action validity and state changes follows classical planning representations (Fikes & Nilsson, 1971; Ghallab et al., 1998).

Definition 1 (Action Complexity)**.** Let $\mathcal{A} = \{a_1, a_2, \ldots, a_A\}$ be the set of distinct action identifiers represented in the corpus. The action complexity is $A = |\mathcal{A}|$.

In our framework, each action type provides a template for concrete actions that serve as inputs to the finite-state machine. Instantiating the same type with different entity arguments does not increase action complexity.

Definition 2 (Concrete Action)**.** A concrete action $a_i(\mathbf{x})$ is obtained by instantiating an action a_i with a type-compatible tuple of entities $\mathbf{x}$ from $\mathcal{E}$. Here $a_i \in \mathcal{A},\ 1 \le i \le A$.

Reusing task rules requires knowing when a concrete action can trigger a valid state transition. We express the requirements for such transitions as execution conditions.

Definition 3 (Execution Condition)**.** An execution condition $p(\mathbf{x})$ is a Boolean expression involving state variables and entity arguments $\mathbf{x}$. Its value in state $s = (s_v)_{v \in \mathcal{V}}$ is

$$p(s, \mathbf{x}) = p(\mathbf{x})|_{v = s_v,\ v \in \mathcal{V}} \in \{0, 1\}. \tag{3}$$

Definition 4 (Valid Action)**.** Let $\mathcal{P}_i$ be the execution conditions for action type a_i, and define

$$g_i(s, \mathbf{x}) = \bigwedge_{p \in \mathcal{P}_i} p(s, \mathbf{x}). \tag{4}$$

A concrete action $a_i(\mathbf{x})$ is valid in state s if and only if $g_i(s, \mathbf{x}) = 1$.

Identifying valid actions means determining whether a transition is permitted. Reusing task rules also requires predicting the state reached after executing a valid action.

Definition 5 (Execution Outcomes)**.** The execution outcome of a valid action $a_i(\mathbf{x})$ in state s is the resulting state s' after executing that action.

Consistent with our view of parametric memory, past experience should induce lasting changes in model parameters that influence subsequent behavior. Here, the rules to be internalized govern action validity and state changes, jointly represented by the transition function F.

Definition 6 (Transition Function)**.** The transition function F encodes task rules by mapping each concrete action and current state to execution outcomes:

$$F(s, a_i(\mathbf{x})) = \begin{cases} s', & g_i(s, \mathbf{x}) = 1, \\ \perp, & g_i(s, \mathbf{x}) = 0, \end{cases} \tag{5}$$

Here, $s' \in \mathcal{S}$ is the execution outcome and $\perp$ denotes an invalid transition.

Execution Trajectories In a finite-state machine, a sequence of valid actions produces a trajectory of states. We record how these actions change entity attributes and relations, and use the resulting execution trajectories to construct training data.

Definition 7 (Execution Trajectory)**.** An execution trajectory of length T_ξ is a sequence $\xi = (s_0, u_1, s_1, \ldots, u_{T_\xi}, s_{T_\xi})$, where $s_0 \in \mathcal{S}$ and, for $1 \le t \le T_\xi$, $u_t = a_{k_t}(\mathbf{x}_t)$ with $a_{k_t} \in \mathcal{A}$, and $s_t = F(s_{t-1}, u_t) \in \mathcal{S}$.

Training trajectories can follow the same task rules while differing in state–action coverage (**breadth**) and execution length (**depth**). These dimensions let us study how the organization of execution records shapes memory formation.

Definition 8 (Trajectory Breadth)**.** Let $\mathcal{T}$ be a collection of training trajectories. With the action set $\mathcal{A}$, transition function F, and trajectory depth fixed, breadth is the range of state-action pairs represented by these trajectories. For $t \ge 1$, their coverage at execution step t is

$$b_t(\mathcal{T}) = \left| \left\{ \left(s_{t-1}^{(\xi)}, u_t^{(\xi)}\right) \;\middle|\; \xi \in \mathcal{T},\ T_\xi \ge t \right\} \right|. \tag{6}$$

We expand breadth by collecting trajectories from additional distinct source examples.

Definition 9 (Trajectory Depth)**.** The depth of an execution trajectory ξ is the number of actions it contains: $d(\xi) = T_\xi$.

2.2 Constructing Training Data from Execution Records

Algorithm 1 constructs supervised pairs (q_j, y_j) from source examples z_i and selected trajectory depths $\mathcal{L}$. After initializing the corpus (Line 1), it iterates over each source and depth (Lines 2–3). COLLECT (Line 4) generates or selects T-step trajectories using action types from $\mathcal{A}$, with transitions satisfying $s_t = F(s_{t-1}, u_t) \neq \perp$.

For each trajectory, QUERIES (Line 5) constructs prediction inputs from the source example and observed execution information. Each query is indexed (Line 6) and paired with a target computed by LABEL from the execution record and F (Line 7). The pair is appended to the corpus (Line 8), which is returned in Line 9.

Algorithm 1 Training Corpus Construction

Input: Action set $\mathcal{A}$, transition function F, source examples $\{z_i\}_{i=1}^N$, trajectory depths $\mathcal{L}$.
Output: Training corpus $\mathcal{D} = \{(q_j, y_j)\}_{j=1}^M$.

1: $\mathcal{D} \leftarrow [\,]$; $M \leftarrow 0$
2: **for** $i = 1, \ldots, N$ **do**
3: **for** each $T \in \mathcal{L}$ **do**
4: **for** each $\xi \in$ COLLECT$(z_i; \mathcal{A}, F, T)$ **do**
5: **for** each $q \in$ QUERIES(z_i, ξ) **do**
6: $M \leftarrow M + 1$; $q_M \leftarrow q$
7: $y_M \leftarrow$ LABEL$(q_M, \xi; F)$
8: APPEND$(\mathcal{D}, (q_M, y_M))$
9: **return** $\mathcal{D}$

3 Experimental Setup

This section describes the task instantiations, training and evaluation procedures, and experimental design used to study memory formation with FSMem.

Task instantiations. We instantiate Algorithm 1 in three task families, constructing execution records and deriving training targets from their task rules. **(1) Synthetic worlds.** We execute valid actions from sampled entity configurations; execution conditions and resulting states provide targets for valid-action identification and state prediction. **(2) Classical planning.** We construct records

from problem instances in 13 ACPBench domains (Kokel et al., 2025). Domain rules determine valid-action targets for query states, specified directly or through an initial state and an action sequence. **(3) Code execution.** Running CodeI/O programs (Li et al., 2025) on concrete inputs yields operation sequences and outputs. Recorded operations supervise next-operation prediction; outputs supervise output prediction from the program and input (Native), optionally augmented with execution order without intermediate values (Explicit trace). Further details are provided in Appendix B.

Training and evaluation. We fine-tune Qwen2.5-0.5B-Instruct (Qwen et al., 2025) with LoRA (Hu et al., 2021), keeping the procedure fixed within each comparison. We assess memory formation through two complementary behaviors: identifying valid actions and predicting execution outcomes on held-out sources without access to training records. In **synthetic worlds**, we evaluate valid-action selection by order-robust accuracy and state prediction by exact match. In **classical planning**, we use candidate selection (order-robust accuracy) or complete valid-action-set prediction (exact-set accuracy and action-set F1). In **code execution**, we evaluate program-output prediction by numeric-equivalence accuracy and, as a complementary test of rule reuse, next-operation selection by order-robust accuracy, using repository-disjoint training and test splits. Order-robust accuracy requires correct answers under every tested candidate ordering. Appendix C reports additional valid-action identification experiments with Qwen2.5-3B-Instruct, Qwen2.5-7B-Instruct, and Qwen3-8B (Yang et al., 2025) in synthetic worlds and classical planning.

Experimental Design Our experiments aim to investigate three research questions about how training-data content and organization shape memory formation. First, **how does action complexity affect the effectiveness of task-rule internalization?** Controlled comparisons introduce additional action types while keeping existing actions' execution conditions and effects unchanged (Section 4). With these rules fixed, the remaining studies focus on trajectory organization. Second, **does broader state–action coverage support memory formation more effectively than repeatedly learning from fewer sources?** Breadth experiments vary the number of distinct source examples while holding trajectory depth and the number of training steps fixed (Section 5). Third, **how does training trajectory depth affect generalization across execution depths, and can short and long trajectories complement each other?** Depth experiments compare training at individual depths with training on cumulative corpora containing both shorter and longer trajectories (Section 6).

4 Action Complexity Affects Memory Formation Effectiveness

We examine how action complexity affects the effectiveness of memory formation. Increasing the number of action types represented in training data expands the range of task rules the model must learn. We assess how effectively models internalize and reuse these rules through two complementary behaviors: **identifying valid actions** in synthetic worlds and classical planning (Section 4.1), and **predicting execution outcomes** in synthetic worlds, with an additional test of next-operation prediction in code (Section 4.2).

4.1 Identifying Valid Actions

We begin with synthetic worlds instantiated from our state-transition formulation, which allows us to vary action complexity in a controlled setting. In synthetic worlds, we expand nested sets of action types from $A{=}5$ to $A{=}20$, holding the other world properties fixed within each comparison. Mean order-robust accuracy decreases from 88.1% to 65.1% (Figure 2 a) with the same number of training steps and examples processed. These results indicate that increasing action complexity reduces the effectiveness of rule internalization under the same training conditions. We next examine whether this finding could be extended to classical planning tasks.

In 13 planning domains, exact-set accuracy is negatively associated with the number of action types appearing in the ground-truth valid-action sets of the training data (Spearman $\rho = -0.819$; Figure 2 b). We then progressively expand the set of action types within seven planning domains to examine whether the same decline appears (Figure 2 c). Including more action types generally makes valid-action identification harder to learn within each domain.

4.2 Predicting Execution Outcomes

Identifying valid actions tests whether a model has learned when an operation can be executed, but does not establish whether it can predict how execution proceeds. We examine next-state prediction in synthetic worlds and, as a complementary test of rule reuse, next-operation prediction in code.

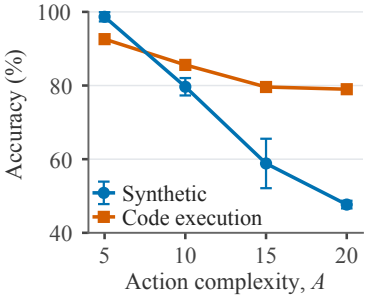


Figure 3: Next-state and next-operation prediction as action complexity increases.

In synthetic worlds, models predict complete next states on the same 200 state-action queries involving five shared action types. Exact accuracy decreases from 98.7% at $A = 5$ to 47.7% at $A = 20$ (Figure 3). A similar decline appears in code next-operation prediction. Given a program, its input, and an execution prefix without intermediate values, the model selects the next operation from two candidates. We expand the training action vocabulary from $A = 5$ to $A = 20$, keeping the same 800 training programs and evaluating the same 500 test events under both candidate orders. In this single-seed comparison, order-robust accuracy decreases from 92.6% to 79.0% (Figure 3). Even on identical questions about shared rules, models trained on a broader rule set make less reliable execution predictions.

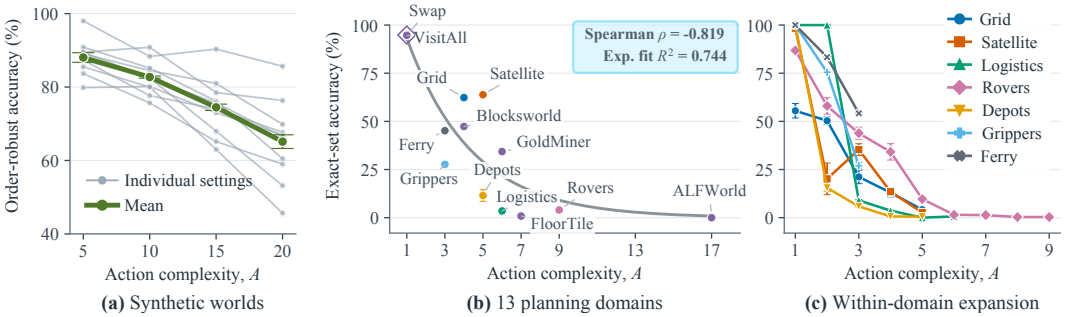
Takeaway 4.1 Increasing action complexity impairs valid-action identification and execution-outcome prediction. The same task rules can become harder to internalize when learned alongside a wider range of actions.

5 Scaling Breadth to Improve Memory Formation

Learning task rules from execution trajectories requires generalizing beyond the particular entities and states recorded during training. To test whether broader state–action coverage supports this generalization, we vary breadth while holding action complexity and trajectory depth fixed (Section 5.1), and assess valid-action identification (Section 5.2) and execution-outcome prediction (Section 5.3).

5.1 Varying Breadth

Following Algorithm 1, we collect trajectories $\mathcal{T}$ from nested sets of N distinct source examples, with the action set $\mathcal{A}$, transition function F, and trajectory depths fixed. Increasing N adds trajectories in which the same action types are applied to different entities and states, expanding the state–action coverage $b_t(\mathcal{T})$ at each execution step t (synthetic counts in Table 5). Within each task family, all conditions use the same number of training steps and effective batch size and share the



(a) Synthetic worlds **(b)** 13 planning domains **(c)** Within-domain expansion

Figure 2: Valid-action identification as action complexity A increases. **(a)** Synthetic order-robust accuracy: light curves show individual settings and the green curve their mean. **(b)** Exact-set accuracy in 13 planning domains; the gray curve is an exponential fit. **(c)** Exact-set accuracy as more action types are included in training within each of seven planning domains.

same test set, so increasing N reduces repeated use of each source without increasing the number of training examples processed.

5.2 Trajectory Breadth Supports Valid-Action Identification

In synthetic worlds, increasing N from 50 to 1,600 raises order-robust accuracy from 39.0% to 61.8%, with most of the gain reached by $N = 800$ (Figure 4, left): at fixed action complexity, broader state–action coverage helps the model learn the execution conditions of the same actions.

In classical planning, 12 of the 13 domains achieve higher exact-set accuracy at $N = 1{,}600$ than at $N = 50$, including gains of 20.1 percentage points on Grid and 18.0 points on Blocksworld (Figure 4, right); gains vary across domains, and ALFWorld remains near zero throughout the sweep. Together with Section 4.1, these results identify breadth as a way to address the difficulty of rule internalization without changing action complexity.

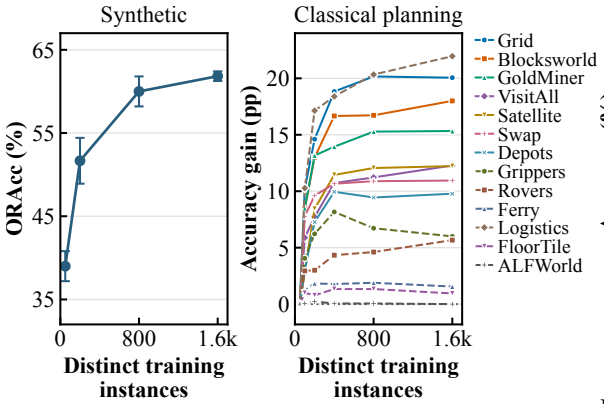


Figure 4: ORAcc: order-robust accuracy (mean ± SD). Planning: gains over N=50.

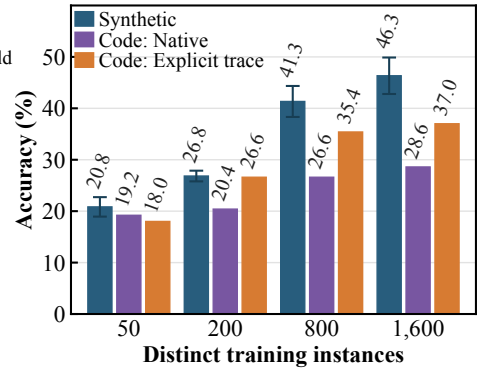


Figure 5: Next-state exact match and numeric-equivalence output accuracy.

5.3 Breadth Scaling Enhances Execution-Outcome Prediction

For execution-outcome prediction, synthetic models predict complete next states under the same A=20 rule set, and code models predict outputs on the same 500 repository-disjoint test programs, trained on nested program sets with a fixed mixture of short, medium, and long trajectories (S+M+L; depth ranges defined in Section 6.3). Synthetic training uses 667,200 prompt tokens and 48,000 supervised tokens per condition; code token counts are not exactly matched. Synthetic next-state exact-match accuracy rises from 20.83 ± 1.89% at N=50 to 46.33 ± 3.55% at N=1,600 (+25.5 points), and program-output accuracy rises from 19.2% to 28.6% in the Native view (+9.4 points) and from 18.0% to 37.0% in the Explicit-trace view (+19.0 points) (Figure 5).

Takeaway 5.1 Expanding trajectory breadth generally improves memory formation, providing a way to mitigate the difficulty of rule internalization under higher action complexity.

6 Benefits of Accumulating Trajectories across Depths

Entity attributes and relations change along a trajectory, but the transition function governing an action does not change with depth, so a model that has learned the rules should reuse them at different trajectory **depths**. We first test this expectation and find that single-depth training produces depth-specialized behavior (Section 6.1). This motivates accumulating trajectories across depths while retaining shorter ones; we compare the resulting cumulative corpora with long-only training on valid-action identification (Section 6.2) and execution-outcome prediction (Section 6.3).

6.1 Single-Depth Training Produces Specialized Behavior

Motivated by known limits of length generalization (Anil et al., 2022), we train separate models at individual depths and evaluate each across depths in the synthetic, planning, and code task families. Table 1 reports, for selected settings, the matching gain (the difference in mean accuracy between matched and mismatched training–test depths) and ρ (Spearman's correlation between the binary depth-matching indicator and accuracy); matched-depth training achieves higher accuracy on average in each. Figure 6 shows the synthetic $A{=}5$ setting, where the matching gain is 12.2 percentage points and is most pronounced at shallow evaluation depths. Longer training executions thus do not automatically support stronger performance at shorter depths.

6.2 Valid-Action Identification across Depths

We test whether retaining shorter trajectories improves valid-action identification as longer trajectories are added. The synthetic corpora cover depths 1, 1+4, 1+4+8, or 1+4+8+16, where "+" denotes depths included together in one corpus, and 16-only training uses the same maximum depth without shorter trajectories. The synthetic and code conditions in Sections 6.2 and 6.3 each use 600 optimizer updates and 9,600 training examples. Order-robust accuracy averaged over evaluation depths rises from 62.1% to 85.7% as depths are added, against 71.2% for 16-only training (Figure 7): a 14.5-point advantage with 29.1% fewer non-padding tokens.

Table 1: Depth matching in selected settings.

Task	Setting	Gain (pp)	ρ
Synthetic	$A = 5$	+12.2	0.751
	$A = 20$	+5.5	0.408
Planning	Ferry	+20.1	0.836
	GoldMiner	+14.2	0.825
	Satellite	+38.2	0.825
Code	Native	+14.8	0.504
	Explicit trace	+12.1	0.275

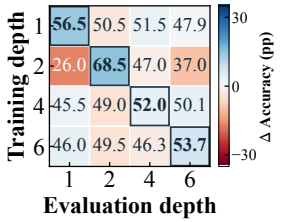


Figure 6: Synthetic depth specialization.

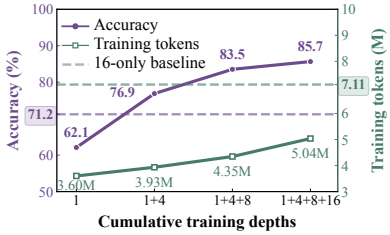


Figure 7: Valid-action identification.

In classical planning, training on depths 2+4+6 outperforms 6-only training in all ten domains displayed in Table 2, using 2.6–5.6% fewer non-padding tokens; Logistics is evaluated by candidate selection (order-robust accuracy) and the other displayed domains by exact-set accuracy. Full exact-set results for all 13 domains are in Appendix A.

Table 2: Planning accuracy (%) and token savings.

Depth	BW	Log.	Dep.	Ferry	Grip.	Sat.	Grid	GM	Swap	VA	Avg.
6-only	35.2	61.6	6.4	50.2	22.5	55.3	36.2	69.1	35.9	75.3	44.8
2	31.5	60.4	6.7	33.7	9.8	32.7	31.1	37.6	35.4	46.8	32.6
2+4	62.3	67.3	11.4	55.8	24.0	66.0	45.2	77.3	91.8	79.3	58.0
2+4+6	**77.0**	**73.1**	**11.8**	**67.8**	**31.1**	**95.9**	**47.1**	**99.4**	**99.7**	**80.0**	**68.3**
ΔTok	−2.9	−3.5	−3.5	−3.4	−4.3	−4.9	−3.2	−4.3	−2.6	−5.6	−3.7

BW: Blocksworld; Log.: Logistics; Dep.: Depots; Grip.: Grippers; Sat.: Satellite; GM: GoldMiner; VA: VisitAll.

6.3 Execution-Outcome Prediction across Depths

In synthetic worlds with varying action complexity A, models receive a held-out initial state and an executed action prefix and predict the complete terminal state. As the corpus expands from depth 1 to 1+4+8+16, terminal-state exact-match accuracy averaged over evaluation depths rises from 32.3% to 99.0%, with corresponding non-padding training tokens reported in Table 3.

To determine whether this gain comes only from the depth-16 trajectories, we compare the full cumulative corpus with 16-only training at each evaluation depth (Figure 8). The cumulative corpus exceeds 16-only training throughout the tested synthetic settings, 99.0% against 64.4% on average (+34.6 points) with 30.3% fewer non-padding tokens; the gap is concentrated at shorter depths (59.4, 62.2, and 16.5 points at depths 1, 4, and 8) and narrows to +0.3 points at depth 16.

For program-output prediction, we construct corpora from the same 800 training programs using short (S: $T \leq 6$), medium (M: $8 \leq T \leq 11$), and long (L: $T \geq 14$) trajectories. We evaluate cumulative corpora S, S+M, and S+M+L on long executions against long-only (L-only) training,

shown as dashed baselines in Figure 9. The full cumulative corpus exceeds L-only training by 1.73 percentage points in the Native view and 7.87 points in the Explicit-trace view, using 1.0% and 15.1% fewer non-padding tokens, respectively. The benefit thus extends to organizing execution records across depths within a fixed set of training programs.

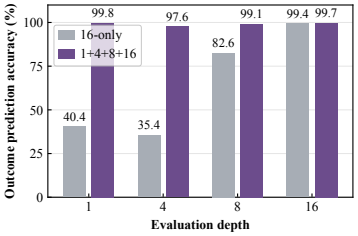


Figure 8: Synthetic accuracy by evaluation depth.

Table 3: Synthetic Results.

Training depths	Avg. acc. (%)	Tokens (M)
16-only	64.4	6.86
1	32.3	3.34
1+4	91.6	3.67
1+4+8	98.3	4.09
1+4+8+16	**99.0**	**4.78**

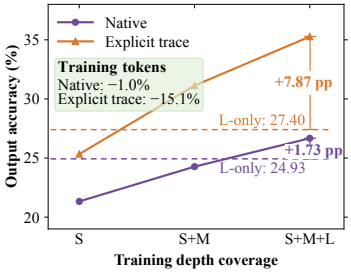


Figure 9: Code Execution on long executions.

Takeaway 6.1 Single-depth training produces depth-specialized behavior, while a cumulative corpus of short and long trajectories can outperform long-only training with fewer tokens.

7 Related Work

Parametric memory in LLMs. Analyses of pretrained Transformers characterize feed-forward layers as key-value memories (Geva et al., 2021) and identify neurons associated with particular facts (Dai et al., 2022). Existing approaches update stored information through targeted parameter changes (Meng et al., 2022a; Fang et al., 2025; Wang et al., 2024b), additional trainable parameters (Dong et al., 2022; Huang et al., 2023), or dedicated memory modules (Lample et al., 2019; Behrouz et al., 2026b; Wei et al., 2026). Engram adds conditional memory through sparse lookup (Cheng et al., 2026), while consolidation methods transfer short-term memories into long-term parameters through distillation and replay (Behrouz et al., 2026a). Recent LoRA-based studies examine capacity, retention, and reasoning over internalized information (Back et al., 2026; Xu et al., 2026).

Training data and memory formation. Varied presentations of facts (Allen-Zhu & Li, 2023), supervision formats (Back et al., 2026), and the number and frequency of facts (Ye et al., 2026) influence acquisition and memorization. For document-specific LoRA memory, training on question-answer pairs outperforms next-token prediction on raw documents when answering questions about the encoded content (Liu et al., 2026). Data repetition also affects verbatim memorization (Carlini et al., 2022; Lee et al., 2022). However, recalling stored information does not ensure its flexible use in downstream reasoning (Allen-Zhu & Li, 2025). Our study concerns execution records of rules shared across entities and states. With fine-tuning fixed, we vary action complexity, trajectory breadth, and depth to study how data construction supports rule internalization, as reflected in identifying valid actions and predicting execution outcomes.

8 Conclusion

We introduced FSMem, a state-transition framework for studying how training-data content and organization shape parametric memory in LLMs. Our findings show that increasing action complexity can make even shared task rules harder to internalize, while broader state–action coverage supports more effective rule reuse. Single-depth training produces depth-specialized behavior, motivating cumulative corpora of short and long trajectories that can outperform long-only training with fewer tokens. Together, these findings show how training-data construction shapes the internalization of experience into reusable task rules, providing guidance for more effective memory formation under a fixed fine-tuning procedure.

AI USE DISCLOSURE

In this work, we used generative AI tools for generating synthetic datasets through programmatic pipelines, helping develop the conceptual framework, proposing or refining hypotheses, providing feedback on research methodology and experiments, implementing methods, assisting with translation, cleaning and reformatting datasets, and interpreting results. We have not used generative AI tools for formulating mathematical claims, and providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, and supporting qualitative and thematic data analysis are not applicable to this work. Additionally, we used generative AI tools for creating or modifying scientific figures, creating or editing software code, drafting parts of the paper, summarizing and analyzing existing literature, brainstorming, identifying relevant literature, improving manuscript readability, formatting references, suggesting a paper structure, and proposing titles and keywords. We have reviewed all AI-assisted work. Specifically, we checked cited sources, verified reported results against experimental outputs, and reviewed AI-assisted code, text, and figures. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.1, knowledge storage and extraction. *arXiv preprint arXiv:2309.14316*, 2023.

Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.3, knowledge capacity scaling laws. *arXiv preprint arXiv:2404.05405*, 2024.

Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.2, knowledge manipulation. In *International Conference on Learning Representations*, volume 2025, pp. 85824–85832, 2025.

Cem Anil, Yuhuai Wu, Anders Andreassen, Aitor Lewkowycz, Vedant Misra, Vinay Ramasesh, Ambrose Slone, Guy Gur-Ari, Ethan Dyer, and Behnam Neyshabur. Exploring length generalization in large language models. *Advances in Neural Information Processing Systems*, 35:38546–38556, 2022.

Seungju Back, Dongwoo Lee, Naun Kang, Taehee Lee, SK Hong, Youngjune Gwon, and Sungjin Ahn. Understanding lora as knowledge memory: An empirical analysis. *arXiv preprint arXiv:2603.01097*, 2026.

Ali Behrouz, Farnoosh Hashemi, Adel Javanmard, and Vahab Mirrokni. Language models need sleep: Learning to self-modify and consolidate memories. *arXiv preprint arXiv:2606.03979*, 2026a.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. *Advances in Neural Information Processing Systems*, 38:113506–113543, 2026b.

Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. Quantifying memorization across neural language models. *arXiv preprint arXiv:2202.07646*, 2022.

Hoyeon Chang, Jinho Park, Seonghyeon Ye, Sohee Yang, Youngkyung Seo, Du-Seong Chang, and Minjoon Seo. How do large language models acquire factual knowledge during pretraining? *Advances in neural information processing systems*, 37:60626–60668, 2024.

Rujikorn Charakorn, Edoardo Cetin, Shinnosuke Uesaka, and Robert Tjarko Lange. Doc-to-lora: Learning to instantly internalize contexts. *arXiv preprint arXiv:2602.15902*, 2026.

Xin Cheng, Wangding Zeng, Damai Dai, Qinyu Chen, Bingxuan Wang, Zhenda Xie, Kezhao Huang, Xingkai Yu, Zhewen Hao, Han Zhang, et al. Conditional memory via scalable lookup: A new axis of sparsity for large language models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4968–4990, 2026.

Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. Knowledge neurons in pretrained transformers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8493–8502, 2022.

Nicola De Cao, Wilker Aziz, and Ivan Titov. Editing factual knowledge in language models. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pp. 6491–6506, 2021.

Qingxiu Dong, Damai Dai, Yifan Song, Jingjing Xu, Zhifang Sui, and Lei Li. Calibrating factual knowledge in pretrained language models. In *Findings of the association for computational linguistics: EMNLP 2022*, pp. 5937–5947, 2022.

Junfeng Fang, Houcheng Jiang, Kun Wang, Yunshan Ma, Jie Shi, Xiang Wang, Xiangnan He, and Tat-Seng Chua. AlphaEdit: Null-space constrained knowledge editing for language models. In *International Conference on Learning Representations*, volume 2025, pp. 16366–16396, 2025.

Richard E Fikes and Nils J Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pp. 5484–5495, 2021.

Malik Ghallab, Adele Howe, Craig Knoblock, Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL—the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998. URL `https://ipc08.icaps-conference.org/deterministic/data/mcdermott-et-al-tr-1998.pdf`.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Zeyu Huang, Yikang Shen, Xiaofeng Zhang, Jie Zhou, Wenge Rong, and Zhang Xiong. Transformer-Patcher: One mistake worth one neuron. *arXiv preprint arXiv:2301.09785*, 2023.

Harsha Kokel, Michael Katz, Kavitha Srinivas, and Shirin Sohrabi. Acpbench: Reasoning about action, change, and planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 26559–26568, 2025.

Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large memory layers with product keys. *Advances in Neural Information Processing Systems*, 32, 2019.

Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. Deduplicating training data makes language models better. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8424–8445, 2022.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33: 9459–9474, 2020.

Junlong Li, Daya Guo, Dejian Yang, Runxin Xu, Yu Wu, and Junxian He. Codei/o: Condensing reasoning patterns via code input-output prediction. *arXiv preprint arXiv:2502.07316*, 2025.

Yewei Liu, Xiyuan Wang, Yansheng Mao, Yoav Gelbery, Haggai Maron, and Muhan Zhang. Shine: A scalable in-context hypernetwork for mapping context to lora in a single pass. *arXiv preprint arXiv:2602.06358*, 2026.

Junru Lu, Siyu An, Mingbao Lin, Gabriele Pergola, Yulan He, Di Yin, Xing Sun, and Yunsheng Wu. MemoChat: Tuning LLMs to use memos for consistent long-range open-domain conversation. *arXiv preprint arXiv:2308.08239*, 2023.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. *Advances in neural information processing systems*, 35:17359–17372, 2022a.

Kevin Meng, Arnab Sen Sharma, Alex Andonian, Yonatan Belinkov, and David Bau. Mass-editing memory in a transformer. *arXiv preprint arXiv:2210.07229*, 2022b.

Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D Manning. Fast model editing at scale. *arXiv preprint arXiv:2110.11309*, 2021.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. Language models as knowledge bases? In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 2463–2473, 2019.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL `https://arxiv.org/abs/2412.15115`.

Adam Roberts, Colin Raffel, and Noam Shazeer. How much knowledge can you pack into the parameters of a language model? In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pp. 5418–5426, 2020.

Kushal Tirumala, Aram Markosyan, Luke Zettlemoyer, and Armen Aghajanyan. Memorization without overfitting: Analyzing the training dynamics of large language models. *Advances in Neural Information Processing Systems*, 35:38274–38290, 2022.

Boshi Wang, Xiang Yue, Yu Su, and Huan Sun. Grokked transformers are implicit reasoners: A mechanistic journey to the edge of generalization. *arXiv preprint arXiv:2405.15071*, 2024a.

Peng Wang, Zexi Li, Ningyu Zhang, Ziwen Xu, Yunzhi Yao, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Wise: Rethinking the knowledge memory for lifelong model editing of large language models. *Advances in Neural Information Processing Systems*, 37:53764–53797, 2024b.

Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. Augmenting language models with long-term memory. *Advances in neural information processing systems*, 36:74530–74543, 2023.

Rubin Wei, Jiaqi Cao, Jiarui Wang, Jushi Kai, Qipeng Guo, Bowen Zhou, and Zhouhan Lin. Mlp memory: A retriever-pretrained memory for large language models. In *International Conference on Learning Representations*, volume 2026, pp. 132772–132795, 2026.

Ziwen Xu, Haiwen Hong, Linsong Yu, Benglei Cui, Longtao Huang, Hui Xue, and Ningyu Zhang. How lora remembers? a parametric memory law for llm finetuning. *arXiv preprint arXiv:2605.30260*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Qiang Yang, Kangheng Wu, and Yunfei Jiang. Learning action models from plan examples using weighted MAX-SAT. *Artificial Intelligence*, 171(2-3):107–143, 2007.

Jiayuan Ye, Vitaly Feldman, and Kunal Talwar. Cram less to fit more: Training data pruning improves memorization of facts. *arXiv preprint arXiv:2604.08519*, 2026.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 19724–19731, 2024.

A Full Planning Depth Results

Table 4 reports exact-set accuracy for all 13 planning domains. Each score averages evaluation depths 2, 4, and 6 and then seeds 13, 37, and 73. Training on depths 2+4+6 outperforms 6-only training in ten domains, ties in Logistics, and is lower in Rovers and FloorTile, while using fewer non-padding training tokens in every domain. Logistics, FloorTile, and ALFWorld remain near the exact-set floor. Unlike the Logistics column in Table 2, which reports valid-action selection, this table consistently evaluates prediction of the complete valid-action set.

Table 4: Full planning depth results. Exact-set accuracy (%) averages evaluation depths 2, 4, and 6. C denotes training on depths 2+4+6; L denotes 6-only training. ΔAcc and ΔTok compare C with L. Tokens are mean non-padding training tokens in millions.

	Exact-set accuracy (%)					Training tokens		
Domain	2	2+4	C	L	ΔAcc (pp)	C (M)	L (M)	ΔTok (%)
Blocksworld	31.50	62.28	77.00	35.17	+41.83	1.424	1.466	−2.9
Depots	6.72	11.44	11.83	6.39	+5.44	2.972	3.080	−3.5
Ferry	33.72	55.78	67.78	50.17	+17.61	2.342	2.424	−3.4
FloorTile	0.11	0.06	0.00	0.11	−0.11	5.142	5.212	−1.3
GoldMiner	37.56	77.28	99.44	69.06	+30.39	3.953	4.132	−4.3
Grid	31.06	45.22	47.11	36.22	+10.89	3.911	4.039	−3.2
Grippers	9.78	24.00	31.11	22.50	+8.61	2.239	2.341	−4.3
Logistics	1.39	0.56	0.67	0.67	0.00	4.250	4.412	−3.7
Rovers	5.06	5.61	6.28	6.61	−0.33	4.626	4.765	−2.9
Satellite	32.67	66.00	95.89	55.33	+40.56	2.527	2.656	−4.9
Swap	35.44	91.83	99.67	35.89	+63.78	3.917	4.022	−2.6
VisitAll	46.78	79.28	80.00	75.33	+4.67	1.988	2.106	−5.6
ALFWorld	0.22	0.17	0.06	0.00	+0.06	28.429	28.772	−1.2

B Data Construction Details

Synthetic worlds. Valid-action queries provide an initial state, an execution record, and candidate actions; labels are obtained by checking execution conditions in the resulting state. The action-complexity study of next-state prediction instead supplies a complete current state and one valid concrete action. Actions use fixed identifiers without descriptions of their effects, and the target contains the resulting state of every entity, including entities unchanged by the action. The depth study supplies an initial state and a valid action sequence and uses the complete terminal state as the target.

Classical planning. Problem instances specify objects and initial states. Domain rules generate valid execution records and determine which concrete actions are valid at each reached state. Current-state queries render the queried state directly; trajectory queries provide the initial state and executed actions. Targets either select a valid candidate or list the complete valid-action set. Records derived from the same source problem remain in the same data split. In the breadth experiments, Logistics and FloorTile use regenerated controlled instances; the other planning domains use native-domain instance construction.

Code execution. Training and test programs are separated by repository. Next-operation queries include the program description, input/output requirements, program, concrete input, and an execution prefix with intermediate values removed. The target operation is obtained from the recorded execution, and each query is presented under both candidate orders. Program-output queries use the same task information and concrete input; Explicit-trace inputs also include executed line and statement order, without intermediate values or the terminal output. Both Native and Explicit-trace inputs share the output produced by execution as their target.

Distinct sources are sampled trajectories, planning problems, and programs for synthetic worlds, classical planning, and code execution, respectively. Breadth comparisons collect records from nested source pools with the action types and trajectory depths fixed. Cumulative depth comparisons retain records at shorter depths when records at longer depths are included.

Result aggregation. Synthetic and planning breadth results average three training seeds; code breadth results use seed 13. Synthetic cumulative-depth outcome results also use seed 13, while the cumulative-depth code results average seeds 13, 37, and 73. Error bars report sample standard deviations where shown.

B.1 Measured Trajectory Breadth

Table 5 reports $b_t(\mathcal{T})$ from the frozen synthetic training corpora, following Definition 8. Counts use symbolic states and executed concrete actions before surface rendering; candidate-order variants and repeated training do not add pairs. Increasing N expands coverage at every step, although distinct trajectories can share state–action pairs, so b_t need not equal N.

Training corpus	Step t	$N = 50$	$N = 200$	$N = 800$	$N = 1600$
Valid-action identification (depth 4)	1	50	193	683	1,191
	2	49	180	602	982
	3	47	149	399	604
	4	35	97	192	291
Next-state prediction (depth 1)	1	50	200	800	1,600

Table 5: Measured breadth b_t in synthetic training corpora ($A = 20$); N counts distinct sources.

C Results on Additional Models

We extend valid-action identification to Qwen2.5-3B-Instruct, Qwen2.5-7B-Instruct, and Qwen3-8B. Candidate selection uses order-robust accuracy, requiring correct answers under every tested ordering. Completed comparisons retain all intermediate conditions.

C.1 Action Complexity and Trajectory Breadth

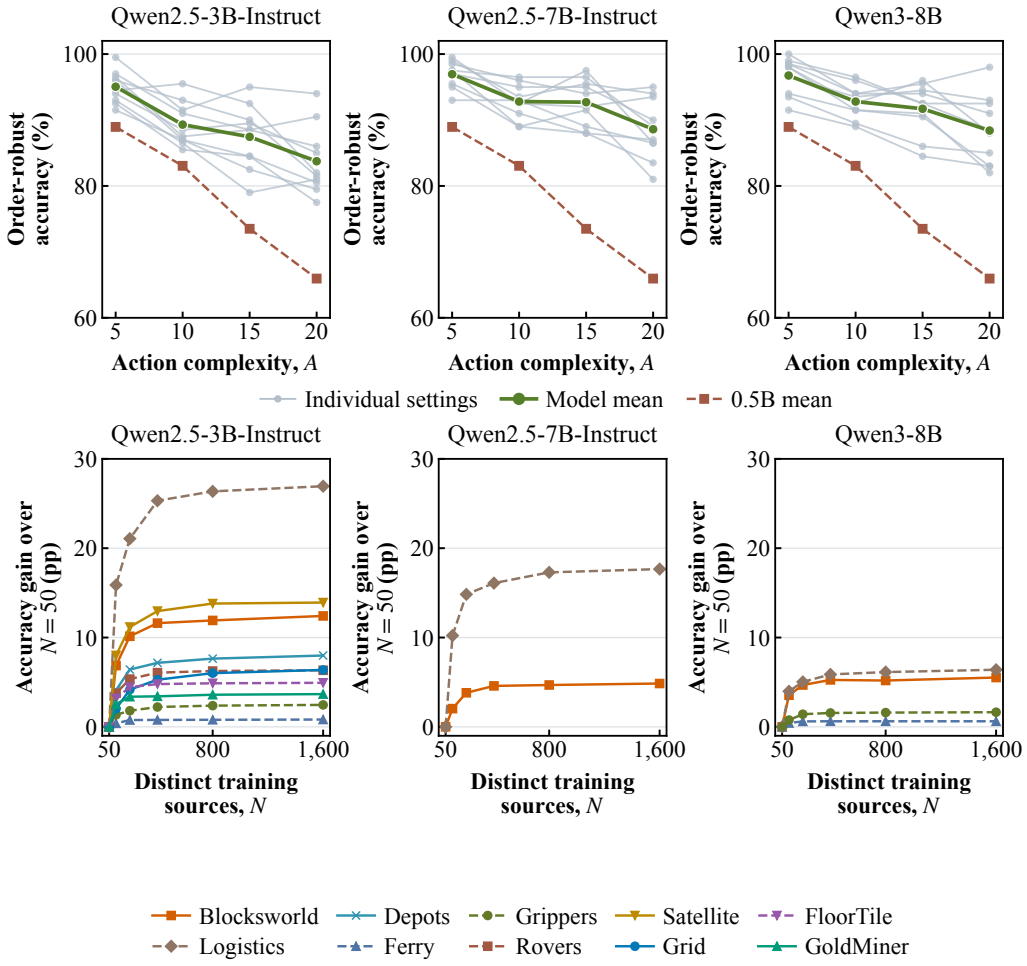


Figure 10: Action complexity and breadth across models. Top: synthetic accuracy; gray lines show individual settings, green their mean, and dashed brown the 0.5B reference. Bottom: planning gains relative to $N = 50$, where N counts distinct training sources; colors identify domains.

Action complexity. Across the same 40 synthetic configurations and 300 training steps (seed 13), mean accuracy decreases from $A = 5$ to $A = 20$ in all three models. The decline is smaller than for 0.5B: larger models mitigate, but do not remove, the effect of action complexity in this comparison.

Trajectory breadth. Increasing distinct sources expands state–action coverage while action complexity and trajectory depths remain fixed. Each condition uses 300 steps and 4,800 training examples. The $N = 1600$ mean exceeds $N = 50$ in all 16 complete model–domain comparisons, despite intermediate plateaus and decreases. Broader coverage remains beneficial across these models without additional training steps.

C.2 Depth Specialization and Cumulative Training

Figure 11 connects single-depth specialization with cumulative training. Planning candidate-selection scores in this section average four candidate counts (2, 3, 4, 8), three training seeds (13, 37, 73), and three evaluation depths (2, 4, 6), except when these factors are shown separately. Every required evaluation must be present.

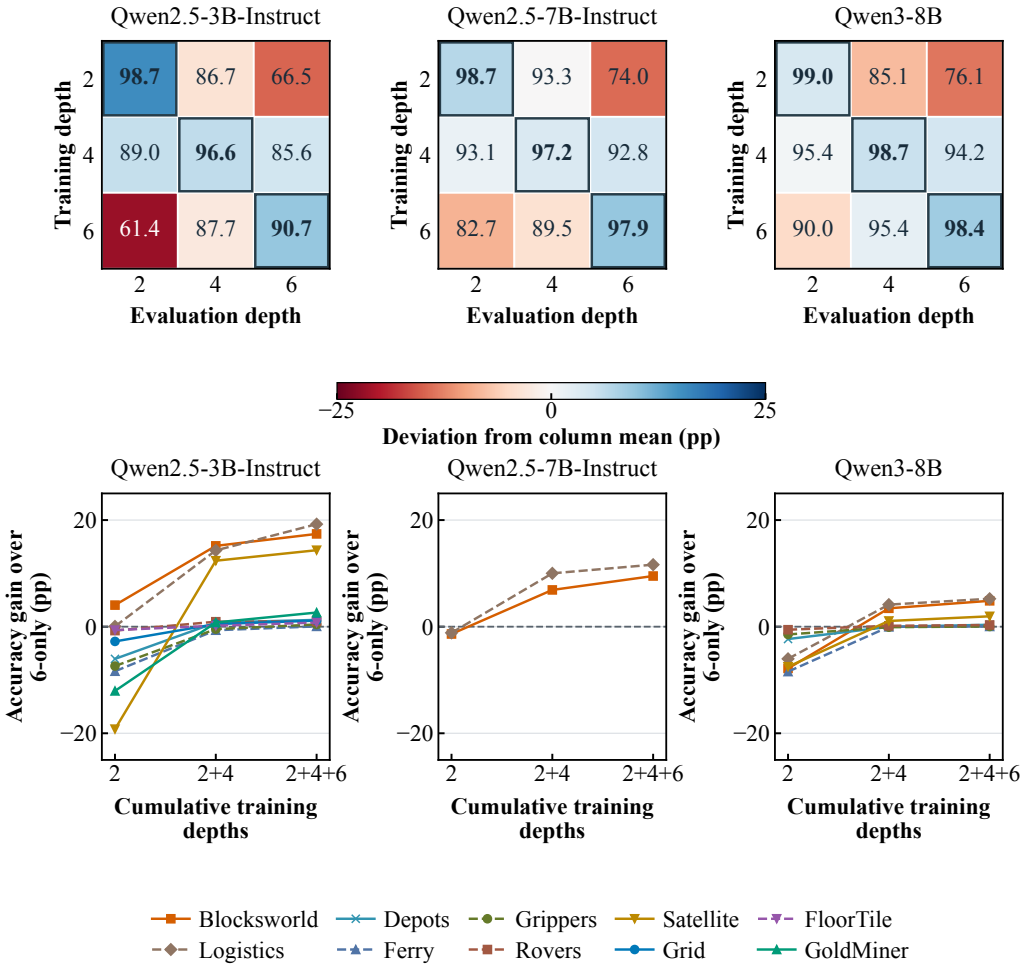


Figure 11: Depth specialization and cumulative training. Top: Blocksworld training–test matrices; numbers show accuracy (%), colors show deviations from column means, and outlines mark matched depths. Bottom: cumulative-training gains over 6-only training (dashed zero line). Each condition uses 300 steps; colors identify domains.

Cumulative training. The 2+4+6 corpus outperforms 6-only training in all 19 complete model–domain comparisons. The plus sign denotes trajectory depths included together in one corpus. Accuracy also increases from 2 to 2+4 and then to 2+4+6 in every comparison. Retaining shorter trajectories therefore supports valid-action identification across models without increasing training steps.

Training–test depth matching. Table 6 provides the other 22 complete matrices; the three Blocksworld matrices appear in Figure 11. Each cell averages four candidate counts and three training seeds. The matching gain, Δ, is the mean of the three matched-depth cells minus the mean of the six mismatched-depth cells. It is positive in all 25 matrices: ten for Qwen2.5-3B, eight for Qwen2.5-7B, and seven for Qwen3-8B. This supports depth specialization as an average tendency; the matched model need not be best at every test depth.

	Train depth 2			Train depth 4			Train depth 6			
Test depth	2	4	6	2	4	6	2	4	6	Δ
Qwen2.5-3B-Instruct										
Logistics	**86.6**	72.5	70.0	81.9	**77.4**	79.7	76.7	77.0	**75.2**	+3.4
Depots	**99.1**	92.8	84.0	97.8	**98.7**	97.5	96.9	98.1	**99.0**	+4.5
Ferry	**100.0**	95.0	79.6	99.9	**99.8**	97.9	99.8	100.0	**100.0**	+4.6
Grippers	**99.9**	93.4	82.8	96.4	**99.5**	95.5	99.1	99.8	**99.3**	+5.1
Rovers	**99.9**	99.2	94.6	98.1	**99.0**	97.9	98.1	98.8	**98.9**	+1.5
Satellite	**99.2**	54.5	43.9	62.0	**98.5**	89.1	61.8	94.6	**98.7**	+31.1
Grid	**99.1**	95.8	91.3	98.6	**98.8**	96.1	98.6	98.3	**97.5**	+2.0
FloorTile	**100.0**	98.1	96.9	99.7	**99.0**	98.7	98.5	98.9	**99.4**	+1.0
GoldMiner	**100.0**	91.0	64.7	99.6	**99.8**	93.9	94.4	97.8	**99.8**	+9.6
Qwen2.5-7B-Instruct										
Logistics	**92.5**	80.7	77.7	90.9	**89.0**	87.4	83.6	84.6	**86.0**	+5.0
Depots	**99.9**	98.2	96.6	99.7	**99.5**	99.3	99.2	98.9	**99.5**	+1.0
Ferry	**100.0**	98.0	93.2	100.0	**100.0**	99.7	100.0	100.0	**100.0**	+1.5
Grippers	**100.0**	98.0	94.7	98.3	**99.6**	95.5	100.0	100.0	**99.7**	+2.0
Satellite	**99.8**	77.2	65.5	85.4	**100.0**	96.1	83.4	99.2	**99.5**	+15.3
Grid	**99.7**	98.4	98.3	99.6	**99.2**	99.1	99.5	98.8	**99.2**	+0.4
FloorTile	**99.8**	98.1	95.1	99.8	**99.9**	99.1	99.8	99.9	**99.4**	+1.0
Qwen3-8B										
Logistics	**98.0**	83.3	77.8	96.7	**91.8**	89.0	92.5	93.2	**91.5**	+5.0
Depots	**99.5**	97.0	94.9	99.5	**99.6**	99.1	99.5	99.5	**99.5**	+1.3
Ferry	**100.0**	91.1	83.4	100.0	**100.0**	100.0	100.0	99.9	**100.0**	+4.3
Grippers	**99.9**	99.0	96.4	99.7	**99.9**	99.3	99.9	100.0	**99.8**	+0.8
Rovers	**99.8**	99.1	97.7	99.5	**99.5**	99.0	99.7	99.4	**99.2**	+0.5
Satellite	**99.8**	85.8	85.6	90.4	**99.6**	95.7	94.5	99.4	**99.8**	+7.9

Table 6: Training–test depth matrices for the remaining planning domains. Values are order-robust accuracy (%); bold marks matched depths. Δ is the matching gain in percentage points. Blocksworld appears in Figure 11.

Scope. These experiments extend the valid-action findings to additional Qwen models. They do not evaluate execution-outcome prediction or establish token savings for these models. Breadth is varied through source-pool size; N is not a count of distinct state–action pairs. The comparisons support the effects of data construction within each model; differences between Qwen2.5 and Qwen3 also involve model generation, not only size.

C.3 Results by Candidate Count

Figure 12 separates the planning results by candidate count to check whether the benefits depend on tasks with few candidates. All 16 complete breadth groups and all 19 complete cumulative-depth groups are retained. Each point averages three training seeds and three evaluation depths. These are disaggregated results from the same experiments, rather than additional training runs.

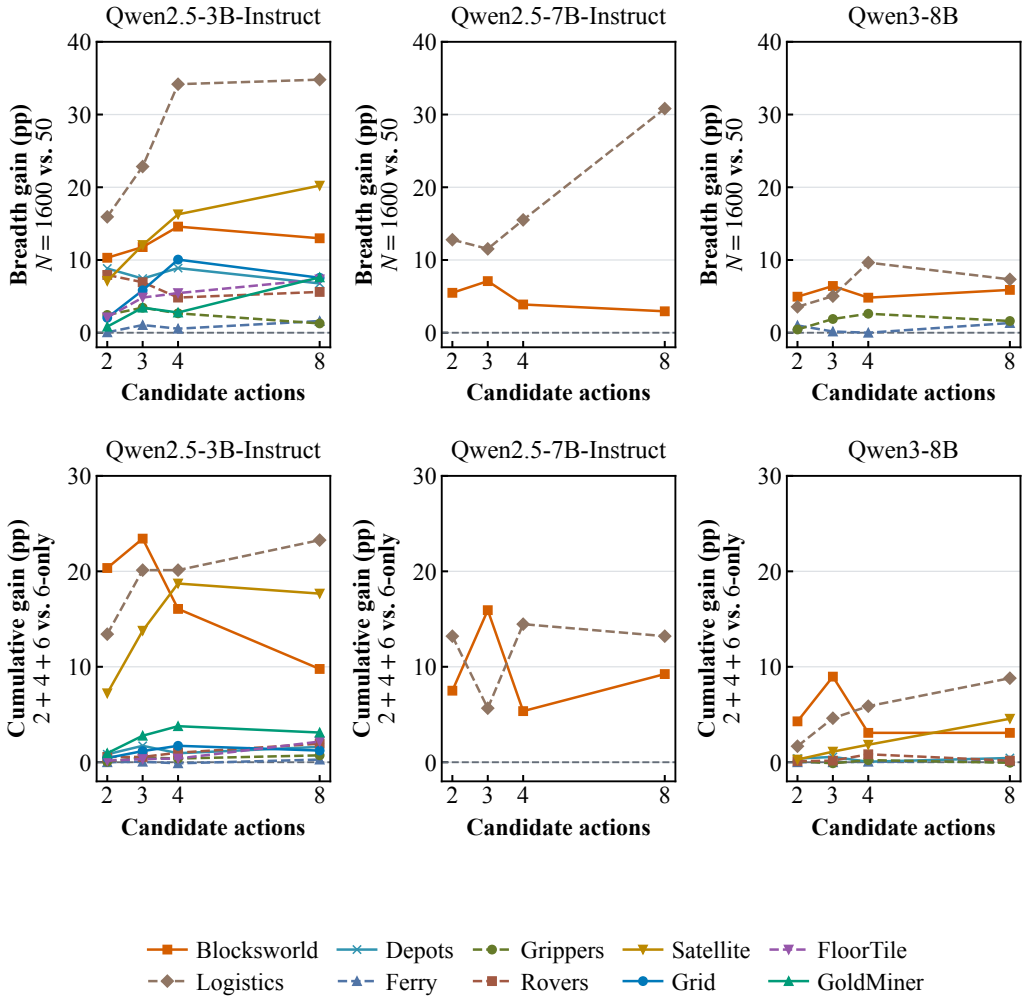


Figure 12: Planning gains by candidate count. Top: $N = 1600$ minus $N = 50$. Bottom: cumulative 2+4+6 minus 6-only training. All comparisons use 300 training steps. The dashed line marks zero gain; colors identify domains.

Breadth yields positive gains in 63 of 64 comparisons and ties in one. Cumulative training improves 70 of 76 comparisons, ties in three, and decreases slightly in three (at most 0.12 percentage points). Both gains remain positive at every candidate count in Blocksworld and Logistics for all three models, including the eight-candidate setting. The average benefits therefore do not arise only from tasks with few candidates.

C.4 Cumulative Training at Different Evaluation Depths

The preceding cumulative-training curves average over test depths. Figure 13 separates these depths to locate the gains from retaining shorter trajectories. It compares 2+4+6 with 6-only training in all 19 complete model–domain groups, using 300 training steps per condition. Each point averages four candidate counts and three training seeds.

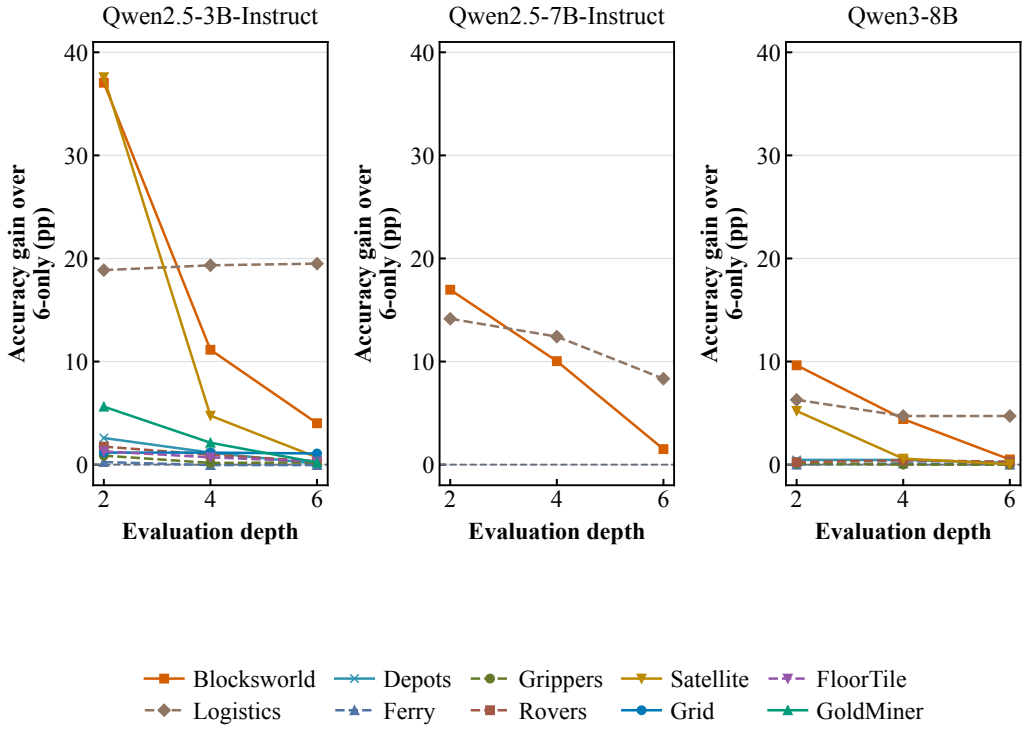


Figure 13: Cumulative-training gains at each evaluation depth. Curves show 2+4+6 minus 6-only order-robust accuracy in all complete planning comparisons. The dashed line marks zero gain; colors identify domains.

Cumulative training improves 18 of 19 comparisons at depth 2, 17 at depth 4, and 16 at depth 6. The remaining comparisons comprise four ties and two decreases smaller than 0.05 percentage points. Within each model's completed task set, the mean gain is greatest at depth 2 and remains positive at depth 6. Retaining shorter trajectories therefore helps most at shallower depths on average, while often also benefiting the deepest tested executions.

These are the same trained models and evaluation records underlying Figure 11, separated by test depth. Model columns contain different sets of completed domains and should not be compared as a controlled model-size sweep.

C.5 Additional Planning Comparisons

Complete valid-action sets. Table 7 complements candidate selection by requiring the entire valid-action set to be correct. All three models have results for 11 domains. Training uses 800 source instances for three epochs; scores average seeds 1, 2, and 3 on the original combined test set.

Domain	Qwen2.5-3B	Qwen2.5-7B	Qwen3-8B
Blocksworld	68.00	88.17	86.00
Depots	56.00	65.00	68.33
Ferry	67.83	92.00	92.33
FloorTile	0.50	0.00	4.17
GoldMiner	47.83	71.83	64.33
Grid	67.00	71.17	69.83
VisitAll	93.50	97.17	97.17
Grippers	51.50	85.67	91.00
Logistics	5.83	22.67	18.00
Rovers	7.83	12.33	18.00
Satellite	80.00	88.67	89.67

Table 7: Complete valid-action-set accuracy (%) in 11 planning domains.

Training-step sensitivity. Table 8 compares 300 total steps with 300 steps per included depth: 600 for 2+4 and 900 for 2+4+6. Additional steps improve 32 of 38 comparisons, tie in three, and reduce accuracy in three. Cumulative training can therefore retain room for further optimization. Its advantage over long-only training in Figure 11 is established separately at matched step counts.

	2+4		2+4+6	
Domain	300 steps	600 steps	300 steps	900 steps
Qwen2.5-3B-Instruct				
Blocksworld	95.08	96.15	97.32	98.03
Logistics	90.62	92.03	95.55	95.96
Depots	98.76	99.33	99.28	99.64
Ferry	99.24	99.39	99.96	99.96
Grippers	98.94	99.19	99.82	99.82
Rovers	99.56	99.69	99.64	99.82
Satellite	97.43	97.46	99.40	99.71
Grid	98.62	98.97	99.28	99.60
FloorTile	99.14	99.42	99.72	99.76
GoldMiner	98.08	98.39	99.94	99.97
Qwen2.5-7B-Instruct				
Blocksworld	96.92	97.56	99.53	99.33
Logistics	94.76	94.97	96.38	96.70
Qwen3-8B				
Blocksworld	98.03	98.29	99.43	99.63
Logistics	96.54	96.65	97.64	98.01
Depots	99.46	99.67	99.85	99.90
Ferry	99.92	99.92	99.99	100.00
Grippers	99.79	99.75	99.93	100.00
Rovers	99.60	99.51	99.75	99.79
Satellite	98.99	99.15	99.86	99.93

Table 8: Order-robust accuracy (%) with 300 total steps or 300 steps per included depth.