IMAGINE BEFORE YOU CALL: FACTORED WORLD MODELS FOR TOOL-USING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-using agents extend the capabilities of large language models by interacting with external APIs and environments. However, reliable tool use remains challenging because interaction histories contain substantial task-irrelevant information, while the value of a tool call often depends on consequences that are unavailable before execution. Existing reward models score tool calls directly from the current context without explicitly modeling how each call changes the environment, leaving the agent's understanding of environment dynamics underconstrained. To tackle these issues, we introduce **FOLD**, a factored world-modeling framework for tool-using agents that predicts the consequences of candidate calls through one-step latent dynamics. FOLD partitions the latent state according to two functional properties: (1) whether state information changes in response to the agent's action; and (2) whether it affects future task reward. This yields four latent blocks with distinct transition and decision roles. Based on this representation, FOLD jointly learns latent transitions, process rewards, and state values. The process reward provides step-level supervision for standard policy optimization, while the predicted successor states support test-time look-ahead before a tool is invoked. We further provide a block-wise identifiability analysis for the learned factorization. Experiments on ALFWorld, τ^2-Bench, and DataSearch show that FOLD achieves 97.6% average accuracy on pairwise tool-call preference decisions, raises downstream task success by 3.7 points, and improves test-time tool-selection accuracy to 95.7% on average.

1 INTRODUCTION AND RELATED WORK

Large language models (LLMs) are increasingly equipped with external tools to retrieve up-to-date information and interact with the environments (Schick et al., 2023; Jin et al., 2025; Li et al., 2026). This enables agents to gather evidence and solve complex multi-step tasks beyond text generation (Qian et al., 2026; Xu et al., 2026). However, reliable tool use remains challenging.

First, interaction histories often contain verbose tool outputs, mixing decision-relevant state information with task-irrelevant content (Hu et al., 2026). This problem becomes more pronounced over multiple steps as tool responses accumulate in the context. For example, searches for data tables or source files may return entries unrelated to the user's request (Cui et al., 2025; Lei et al., 2025), and structured API responses may contain redundant metadata and other task-irrelevant fields (Wang et al., 2026). Existing approaches reduce this redundancy by pruning observations (Yang et al., 2025b; Kerboua et al., 2025) or summarizing changes between consecutive states (Chae et al., 2025). However, reducing context length alone does not ensure that the learned state representation separates task-relevant information from irrelevant variation. This makes it difficult for the agent to select the next tool call.

Second, many tool-use agents are trained with rewards based only on final task success (Jin et al., 2025; Qian et al., 2026). Such feedback provides limited guidance on individual calls: a search may retrieve the correct evidence even if the agent later gives an incorrect answer. One way to address this limitation is to provide a step-level reward signal for each tool call (Ma et al., 2026; Xi et al., 2026). However, existing process reward models (PRMs) typically score a call directly from the current context, without explicitly modeling the latent transition induced by that call. In fact, recent

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.

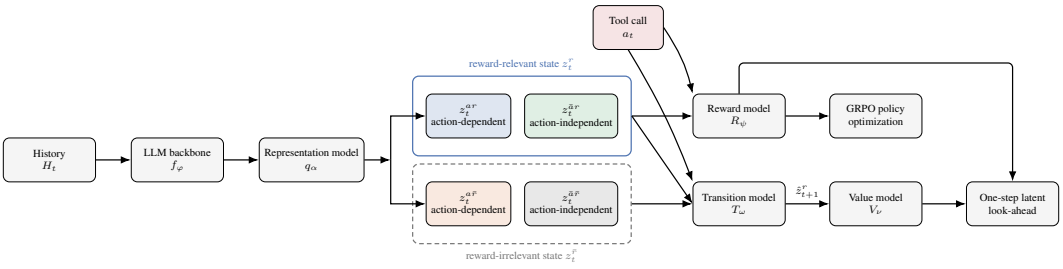


Figure 1: Overview of FOLD. The LLM representation is partitioned according to action dependence and reward relevance. All four latent blocks are used for structured transition modeling, while only the reward-relevant state is used for process reward prediction. During policy optimization, the reward model provides process supervision for GRPO. At inference time, the transition and value models evaluate the predicted consequences of candidate tool calls.

studies have shown that learning environment dynamics can improve decision making by tool-using agents (Richens et al., 2025; Zhang et al., 2025; Guo et al., 2025).

To address these limitations, we introduce **FOLD**, a factored world-modeling framework that predicts the consequences of candidate tool calls through one-step latent dynamics. Specifically, FOLD partitions the latent state according to two properties: (1) whether the state information changes in response to the agent's action, and (2) whether it is relevant to future task reward. Combining these properties produces four latent blocks with different roles in state transition and decision making. The reward-relevant blocks preserve both task progress induced by tool calls and external conditions that influence subsequent decisions, while the reward-irrelevant blocks capture incidental variations in actions and observations. Based on this structured representation, FOLD jointly learns latent transitions, process rewards, and state values. The learned reward model provides step-level supervision for downstream policy optimization, and the transition and value models enable latent look-ahead at inference time: FOLD predicts the reward-relevant successor state of each candidate call and evaluates both its immediate reward and future utility. The agent can therefore assess the likely consequence of a tool call before interacting with the actual environment.

Our main contributions are summarized as follows:

- **Factored world modeling for tool use.** We propose FOLD, which partitions the latent interaction state by action dependence and reward relevance, and jointly learns structured latent transitions, process rewards, and state values on a shared LLM backbone.
- **Consequence-aware policy learning and tool selection.** The learned model supports both training-time and inference-time tool use: its reward module provides step-level supervision for policy optimization, and its transition and value modules enable one-step latent look-ahead over candidate tool calls before execution.
- **Theoretical characterization.** We establish sufficient conditions under which the four latent blocks are identifiable up to block-wise transformations.
- **Empirical evaluation across tool-use settings.** Experiments on ALFWorld (Shridhar et al., 2020), τ^2-Bench (Barres et al., 2025), and DataSearch demonstrate that FOLD learns accurate latent dynamics and process rewards, provides effective process supervision for downstream policy optimization, and improves consequence-aware tool selection.

2 Problem Setting and Background

2.1 Tool Use as a Partially Observable Decision Process

We formulate tool use as a partially observable Markov decision process $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, P, \Omega, \rho, \gamma \rangle$, where $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{O}$ denote the state, action, and observation spaces, respectively; $P(s_{t+1} \mid s_t, a_t)$ is the state-transition distribution; $\Omega(o_t \mid s_t)$ is the observation distribution; $\rho(s_t, a_t)$ is the reward function; and $\gamma \in [0, 1)$ is the discount factor.

At step t, the agent receives an observation $o_t \in \mathcal{O}$, such as a tool response or environment feedback, and appends it to the interaction history

$$H_t = (o_0, a_0, o_1, \ldots, a_{t-1}, o_t). \tag{1}$$

The observation reflects an underlying environment state $s_t \in \mathcal{S}$ that is not directly accessible to the agent, formally $o_t \sim \Omega(\cdot \mid s_t)$. Based on H_t, the agent samples a tool call $a_t \in \mathcal{A}$ according to $\pi_\theta(a_t \mid H_t)$. Executing a_t moves the environment to $s_{t+1} \sim P(\cdot \mid s_t, a_t)$, produces the next observation $o_{t+1} \sim \Omega(\cdot \mid s_{t+1})$, and yields the reward $r_t = \rho(s_t, a_t)$. The discounted return from step t is

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}. \tag{2}$$

2.2 Process Reward Models for Tool Calls

Outcome reward evaluates the final task result but does not directly indicate which intermediate tool calls contribute to success or failure. Instead, a process reward model provides finer-grained supervision by assigning a score to each tool call,

$$\hat{r}_t = R_\psi(H_t, a_t). \tag{3}$$

During policy optimization, these step-level scores can be assigned to their corresponding tool-call turns and used to construct turn-level advantages. At inference time, the same reward model can rerank a candidate set $\mathcal{C}_t = \{a_t^{(1)}, \ldots, a_t^{(N)}\}$ generated by the policy:

$$\hat{a}_t = \arg\max_{a \in \mathcal{C}_t} R_\psi(H_t, a). \tag{4}$$

2.3 Latent World Models for Tool-Using Agents

For an LLM-based agent, the interaction history H_t is first processed by an LLM backbone f_φ, producing a contextual representation $h_t = f_\varphi(H_t)$. A representation model then maps h_t to a compact state z_t that retains the information required to model future interaction dynamics and rewards. Using z_t as the latent state, the world model (Ha & Schmidhuber, 2018; Yu et al., 2026) characterizes the agent-environment interaction through three components:

$$\begin{cases} \text{representation model:} & q_\alpha(z_t \mid h_t), \\ \text{transition model:} & p_\omega(z_{t+1} \mid z_t, a_t), \\ \text{reward model:} & p_\psi(r_t \mid z_t, a_t). \end{cases} \tag{5}$$

Here, the transition model predicts how the latent state changes after executing a tool call, while the reward model estimates the immediate utility of that call.

3 Factored World Modeling for Tool Use

In this section, we first introduce the four-block latent representation used by FOLD and construct the corresponding factored world model. We then present the learning objectives for estimating the latent dynamics. Finally, we describe how FOLD is used as a process reward model during policy optimization and as a one-step look-ahead model during inference. Figure 1 provides an overview of the proposed framework.

3.1 Factored Latent World Model

3.1.1 Four-Block State Factorization

To characterize the different roles of information in tool interaction, we decompose the latent state z_t according to two properties: (1) whether the state information is affected by the preceding action; and (2) whether it is relevant to the future return. Combining the two properties yields four latent blocks:

$$z_t = \left(z_t^{ar}, z_t^{\bar{a}r}, z_t^{a\bar{r}}, z_t^{\bar{a}\bar{r}}\right), \tag{6}$$

where the first superscript indicates action dependence (a versus $\bar{a}$) and the second indicates reward relevance (r versus $\bar{r}$). Figure 2 illustrates the four blocks and the dependencies that define their roles.

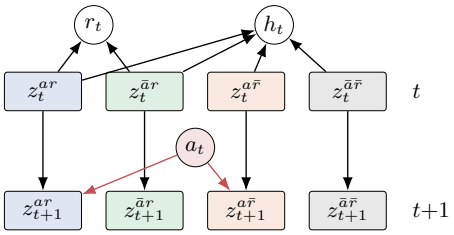


Figure 2: Four-block latent state in FOLD.

Accordingly, the four blocks capture distinct functional roles in a tool interaction. The block z_t^{ar} represents task progress induced by previous tool calls, such as retrieved evidence or slots filled by a successful API call. The block $z_t^{\bar{a}r}$ represents external task conditions that affect future decisions but are not changed by the preceding call, such as tool availability or the backend state. The block $z_t^{a\bar{r}}$ captures call-induced variation that does not affect task completion, such as session tokens or request identifiers regenerated at each call, while $z_t^{\bar{a}\bar{r}}$ captures task-irrelevant content that persists across calls, such as background page elements or unrelated objects. We group these blocks by reward relevance and action dependence:

$$z_t^r = \left[z_t^{ar}; z_t^{\bar{a}r}\right], \quad z_t^{\bar{r}} = \left[z_t^{a\bar{r}}; z_t^{\bar{a}\bar{r}}\right], \\ z_t^a = \left[z_t^{ar}; z_t^{a\bar{r}}\right], \quad z_t^{\bar{a}} = \left[z_t^{\bar{a}r}; z_t^{\bar{a}\bar{r}}\right]. \tag{7}$$

The process reward and state-value models use z_t^r, since both action-induced progress and external task conditions can influence the value of a candidate call. The complete latent state z_t is retained for transition modeling.

3.1.2 Structured Transition, Reward, and Value Models

Let $\mathcal{B}_a = \{ar, a\bar{r}\}$ and $\mathcal{B}_{\bar{a}} = \{\bar{a}r, \bar{a}\bar{r}\}$ denote the action-dependent and action-independent blocks, respectively. Building on the four-block state in Eq. (6), FOLD instantiates the transition and reward models of Eq. (5) and additionally learns a state-value model V_ν to support one-step look-ahead. The transition model follows the action structure of the latent state: each block evolves from its own previous state, and only the action-dependent blocks are conditioned on the current call,

$$\hat{z}_{t+1} = T_\omega(z_t, a_t) = \big(T_\omega^{ar}(z_t^{ar}, a_t),\, T_\omega^{\bar{a}r}(z_t^{\bar{a}r}),\, T_\omega^{a\bar{r}}(z_t^{a\bar{r}}, a_t),\, T_\omega^{\bar{a}\bar{r}}(z_t^{\bar{a}\bar{r}})\big), \tag{8}$$

so that the transitions of z^{ar} and $z^{a\bar{r}}$ reflect action-induced changes, whereas $z^{\bar{a}r}$ and $z^{\bar{a}\bar{r}}$ evolve independently of the current call. The process reward model R_ψ and the state-value model V_ν both operate on the reward-relevant state, estimating the immediate utility of a call and its expected return:

$$\hat{r}_t = R_\psi(z_t^r, a_t), \qquad \hat{G}_t = V_\nu(z_t^r). \tag{9}$$

This restriction excludes reward-irrelevant variation from decision making.

3.2 Learning Objectives

In this section, we describe the estimation procedure for FOLD using tool-interaction trajectories (H_t, a_t, H_{t+1}, G_t), where H_{t+1} is the observed successor history and G_t is the return-to-go. Specifically, FOLD jointly minimizes

$$\mathcal{L}_{\text{FOLD}} = \mathcal{L}_{\text{rew}} + \lambda_{\text{dyn}}\mathcal{L}_{\text{dyn}} + \lambda_{\text{val}}\mathcal{L}_{\text{val}}, \tag{10}$$

where $\mathcal{L}_{\text{dyn}}$ learns the action-conditioned successor state, $\mathcal{L}_{\text{rew}}$ learns to rank the immediate utility of candidate calls, and $\mathcal{L}_{\text{val}}$ estimates their downstream utility. The coefficients λ_{dyn} and λ_{val} control the contributions of the transition and value objectives relative to the reward-ranking objective.

Latent transition objective. For an executed call, the transition model predicts the encoded successor state. Let $h_{t+1} = f_\varphi(H_{t+1})$ and $z_{t+1} = \text{sg}(q_\alpha(h_{t+1}))$ denote the stop-gradient successor encoding. We minimize the blockwise prediction error

$$\mathcal{L}_{\text{dyn}} = \frac{1}{|\mathcal{B}|}\sum_{b\in\mathcal{B}} \left\| T_\omega^b(z_t^b, a_t) - z_{t+1}^b \right\|_2^2, \tag{11}$$

where $\mathcal{B} = \mathcal{B}_a \cup \mathcal{B}_{\bar{a}}$, and T_ω^b receives a_t only when $b \in \mathcal{B}_a$. This objective grounds the action-dependent and action-independent split in the observed consequences of tool calls.

Process-reward objective. For a shared history H_t and candidate tool calls $\mathcal{C}_t = \{a_t^{(1)}, \ldots, a_t^{(N)}\}$, the reward model assigns each candidate the score $u_t^{(i)} = R_\psi(z_t^r, a_t^{(i)})$, which is supervised by a scalar process label $y_t^{(i)}$ that indicates the candidate's relative utility for task progress. For example, a call that completes a required subgoal receives a higher label than an invalid or task-irrelevant call. We convert the labels and scores into distributions within the candidate group using a temperature τ,

$$p_i = \frac{\exp(y_t^{(i)}/\tau)}{\sum_j \exp(y_t^{(j)}/\tau)}, \qquad \hat{p}_i = \frac{\exp(u_t^{(i)}/\tau)}{\sum_j \exp(u_t^{(j)}/\tau)}, \tag{12}$$

and optimize the listwise cross-entropy

$$\mathcal{L}_{\text{rew}} = -\sum_{i=1}^{N} p_i \log \hat{p}_i. \tag{13}$$

This objective trains R_ψ to distinguish useful calls from ineffective alternatives under the same history, providing the immediate-utility term used for policy optimization and candidate reranking. Details of the process-label construction are provided in Appendix C.2.

State-value objective. To provide the future-utility term used by latent look-ahead, the value model regresses the return G_t from the reward-relevant state:

$$\mathcal{L}_{\text{val}} = \left(V_\nu(z_t^r) - \text{sg}(G_t)\right)^2. \tag{14}$$

Consequently, the reward model evaluates the immediate quality of a candidate, while the value model estimates the utility remaining after its predicted successor state is reached.

3.3 Policy Optimization and Test-Time Look-Ahead

3.3.1 Process-Reward-Guided Policy Optimization

During policy optimization, FOLD uses $R_\psi(z_t^r, a_t)$ as the process reward for each tool call. For a rollout consisting of T tool-call turns followed by a final-answer turn, the reward is defined as

$$\tilde{r}_t = \begin{cases} R_\psi(z_t^r, a_t), & 1 \le t \le T, \\ r^{\text{out}}, & t = T+1, \end{cases} \tag{15}$$

where r^{out} denotes the outcome reward determined by the correctness of the final answer. Following Ma et al. (2026), for $1 \le t \le T$, $\tilde{r}_t$ is assigned to the tool-call turn containing a_t, whereas r^{out} is assigned to the final-answer turn. Turn-level advantages are then estimated by normalizing the rewards independently for each turn across rollouts in the same group, and the policy is optimized using GRPO (Shao et al., 2024). Details are provided in Appendix C.4.

3.3.2 Latent Look-Ahead for Tool Selection

At inference time, the policy samples a candidate set $\mathcal{C}_t$, and FOLD performs a one-step look-ahead over these candidates. For each candidate $a_t^{(i)}$, FOLD predicts the reward-relevant successor state

$$\hat{z}_{t+1}^{r,(i)} = \left[T_\omega^{ar}(z_t^{ar}, a_t^{(i)}); T_\omega^{\bar{a}r}(z_t^{\bar{a}r})\right], \tag{16}$$

where $[\cdot;\cdot]$ denotes concatenation. Since $z^{\bar{a}r}$ evolves independently of the current call, its predicted transition is shared across candidates, so the successor states differ only in the action-dependent component. Each candidate is then scored by

$$S(a_t^{(i)}) = R_\psi(z_t^r, a_t^{(i)}) + \gamma V_\nu\left(\hat{z}_{t+1}^{r,(i)}\right), \tag{17}$$

where the first term estimates the immediate process reward and the second estimates the future utility of the predicted successor state. Accordingly, FOLD selects the candidate with the highest look-ahead score:

$$a_t^* = \arg\max_{a_t^{(i)} \in \mathcal{C}_t} S(a_t^{(i)}). \tag{18}$$

This allows the agent to compare the expected consequences of candidate calls before interacting with the environment.

3.4 BLOCKWISE IDENTIFIABILITY

We now present an identifiability result for the four-block state decomposition. Specifically, Theorem 1 shows that each learned block is identifiable up to a C^1 invertible transformation within the corresponding ground-truth block, provided that the action assignment and the return satisfy suitable separation conditions. Below we first introduce the two auxiliary variables and the definitions required to state the result, and then present the theorem. Formal definitions of the auxiliary variables and the separation conditions, together with the complete proofs, are provided in Appendix A.

Let *action assignment* E_t^a denote the observed value of a randomized intervention on the preceding tool call A_{t-1}, and let *return* G_t denote the discounted return of the trajectory. The former separates action-dependent from action-independent information, whereas the latter separates reward-relevant from reward-irrelevant information. We further denote Z_t and $\widehat{Z}_t$ as the latent states of the ground-truth model M and a population-equivalent candidate model $\widehat{M}$, respectively. Let $Z_t^a = (Z_t^{ar}, Z_t^{a\bar{r}})$ and $Z_t^{\bar{a}} = (Z_t^{\bar{a}r}, Z_t^{\bar{a}\bar{r}})$ for the action-based grouping, and $Z_t^r = (Z_t^{ar}, Z_t^{\bar{a}r})$ and $Z_t^{\bar{r}} = (Z_t^{a\bar{r}}, Z_t^{\bar{a}\bar{r}})$ for the reward-based grouping.

Definition 1 (Population equivalence)**.** *A candidate $\widehat{M}$ is* population equivalent *to M, denoted $M \equiv_{\text{pop}} \widehat{M}$, if the two models induce the same law of (H_t, X_t^e, C_t^e) for both auxiliary variables $e \in \{a, r\}$. If the observation maps are bijective with measurable inverses, the common observable law induces the coupling $Z_t = g^{-1}(H_t)$ and $\widehat{Z}_t = \widehat{g}^{-1}(H_t)$.*

Definition 2 (Blockwise C^1 identifiability)**.** *Let $\mathcal{B} = \{ar, \bar{a}r, a\bar{r}, \bar{a}\bar{r}\}$. The representation is* blockwise C^1 identifiable *within a model class if every population-equivalent candidate in that class satisfies $\widehat{Z}_t^b = h_b(Z_t^b)$ for all $b \in \mathcal{B}$, where each h_b is a C^1 diffeomorphism.*

Theorem 1 (Four-block identifiability of FOLD)**.** *Let M be a ground-truth four-block model. If every population-equivalent candidate $\widehat{M}$ in the model class under consideration satisfies the following conditions:*

- ***(A1) Regular product geometry.*** *For every $b \in \mathcal{B}$, $\mathcal{Z}_b$ and $\widehat{\mathcal{Z}}_b$ are nonempty connected open subsets of Euclidean spaces of the same dimension, and the ground-truth and candidate domains are the corresponding four-fold products. Let $P_{Z_t}^e$ denote the latent-state law under the auxiliary variable e; for $e \in \{a, r\}$, $P_{Z_t}^e$ has full support relative to the ground-truth domain $\mathcal{Z}$, i.e., $P_{Z_t}^e(O) > 0$ for every nonempty open $O \subseteq \mathcal{Z}$.*
- ***(A2) Invertible observation representation.*** *$H_t = g(Z_t) = \widehat{g}(\widehat{Z}_t)$, where g and $\widehat{g}$ are C^1 diffeomorphisms onto the same observable C^1 manifold.*
- ***(A3) Separation by action and return.*** *Fix the transformation classes $\mathcal{H}_a$ and $\mathcal{H}_r$ once for the model class, before any candidate is selected. The action and return regroupings of $h := \widehat{g}^{-1} \circ g$ belong to $\mathcal{H}_a$ and $\mathcal{H}_r$, respectively. The action assignment smoothly separates $(Z_t^a, Z_t^{\bar{a}})$ relative to $(h, \mathcal{H}_a)$, and the return G_t smoothly separates $(Z_t^r, Z_t^{\bar{r}})$ relative to $(h, \mathcal{H}_r)$, in the sense of Definition 5, with $P_Z = P_{Z_t}^e$ for the corresponding auxiliary variable.*

Then the representation is blockwise C^1 identifiable.

4 EXPERIMENTS

In this section, we evaluate the benefits of FOLD for agentic tool use. Specifically, our evaluation is designed to answer the following three research questions:

- **RQ1 (Tool-call assessment):** Can FOLD accurately assess the quality and anticipated consequences of tool calls?
- **RQ2 (End-to-end policy improvement):** Do FOLD process rewards translate into improved end-to-end tool-use task performance?
- **RQ3 (Tool selection with latent look-ahead):** Can FOLD enable agents to make better tool-selection decisions through latent look-ahead?

Additional experiments and analyses, including ablation studies, are reported in Appendix D.

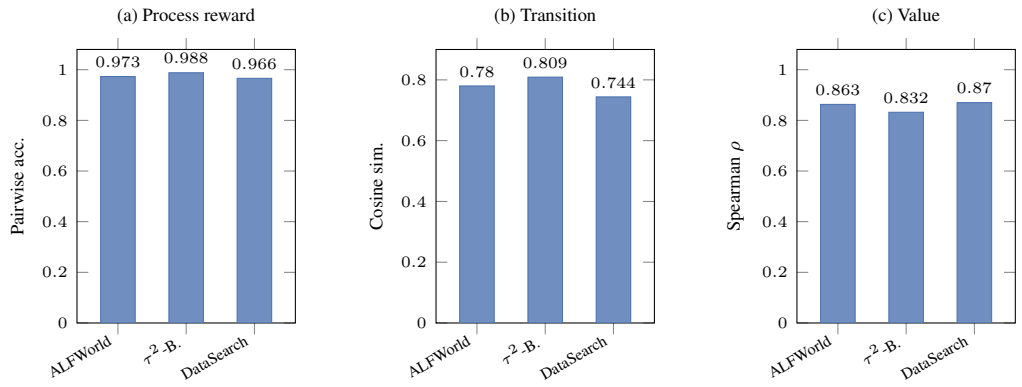


Figure 3: FOLD's assessment of tool-call quality and consequences across three environments. (a) Process-reward pairwise accuracy. (b) Cosine similarity between predicted and target successor representations. (c) Spearman correlation between the predicted value and the observed return-to-go.

4.1 FOLD Provides Reliable Tool-Call Assessment (RQ1)

Evaluation environments. We evaluate FOLD on two representative tool-use environments, namely ALFWorld and τ^2-Bench, as well as on a real-world search-agent dataset. ALFWorld contains temporally extended household tasks with explicit state-changing actions. τ^2-Bench involves multi-turn interactions with structured APIs and state-dependent tool outcomes. We further use DataSearch, a real-world search-agent dataset in which agents locate relevant tables and data sources through search, browsing, and retrieval tools. Detailed descriptions are provided in Appendix C.1.

Model configuration. We use Qwen3-4B-Instruct-2507 (Yang et al., 2025a) as the common backbone across all environments. Full training and optimization settings are provided in Appendix C.5, and a model-scale comparison with an 8B backbone is provided in Appendix D.5.

Evaluation metrics. We assess whether FOLD can evaluate both the immediate quality of a candidate tool call and its anticipated consequences. Specifically, macro-averaged pairwise accuracy within candidate groups measures whether the process reward ranks useful calls above ineffective alternatives. Cosine similarity between predicted and target successor representations evaluates latent dynamics, while semantic-event accuracy and macro-F1 from a frozen linear probe test whether the predicted successor preserves task-relevant state changes rather than only latent similarity. Finally, the Spearman correlation between the predicted value and the observed return-to-go evaluates whether the reward-relevant state captures downstream utility. Formal definitions of all metrics are provided in Appendix C.3.

Results. Figure 3 summarizes FOLD's assessment of tool-call quality and anticipated consequences. Process-reward prediction remains accurate across all three environments, with pairwise accuracy of 97.3% on ALFWorld, 98.8% on τ^2-Bench, and 96.6% on DataSearch, indicating that the reward-relevant state preserves the information required to distinguish useful calls even under heterogeneous real-world tool responses. For latent dynamics, a frozen probe recovers the next semantic event with 86.1% accuracy from the predicted successor state, versus 50.6% from the current state without applying the transition (see Table 1). This gap shows that the predicted successor encodes action-dependent semantic changes beyond those available in the current state. Value predictions correlate strongly with observed returns on all three environments, suggesting that the value model captures

Table 1: Semantic-event probing on ALFWorld. Accuracy and macro-F1 of a linear probe predicting the next semantic event, from the transition prediction $\hat{z}_{t+1}$ or from the current state z_t.

Probed state	Accuracy	Macro-F1
z_t (no transition)	0.506	0.169
$\hat{z}_{t+1}$	**0.861**	**0.348**



(a) Encoder state z_t (b) Transition prediction $\hat{z}_{t+1}$

Figure 4: Block recovery in the controlled environment: bidirectional R^2 between learned and ground-truth blocks, for (a) the encoder state z_t and (b) the transition prediction $\hat{z}_{t+1}$.

information about downstream task progress. Together, these results show that FOLD provides the reward, transition, and value signals needed for consequence-aware tool selection.

Controlled block recovery. We further examine whether FOLD recovers the intended four-block structure in a controlled environment with known ground-truth factors. Figure 4 shows that the encoder state alone yields only partial recovery, whereas the transition prediction produces a near-diagonal recovery matrix. This pattern suggests that the structured transition is central to separating the functional blocks, providing controlled empirical evidence for the factorization used by FOLD. Details of the controlled environment and evaluation protocol are provided in Appendix D.1.

4.2 FOLD Process Rewards Improve End-to-End Tool Use (RQ2)

Baselines. We compare FOLD against the following representative baselines for end-to-end tool-use policy learning:

- **SFT**: supervised fine-tuning on expert trajectories in each environment, which provides a reference without reinforcement learning.
- **ReAct** (Yao et al., 2022): a prompting-based agent that interleaves reasoning traces and tool calls, serving as a training-free reference without parameter updates or learned rewards.
- **GRPO** (Shao et al., 2024): GRPO trained solely with the environment outcome reward, isolating the contribution of process-level supervision.
- **RWML** (Yu et al., 2026): a two-stage world-model approach that learns a world model before downstream policy RL, without FOLD's factored process-reward and value signals.
- **TRM** (Ma et al., 2026): GRPO guided by a tool-call reward model trained on distilled turn-level utility labels, without an explicit factored transition model for consequence prediction.

All methods use the same base model within each environment: Qwen2.5-7B-Instruct (Yang et al., 2024) on ALFWorld, and Qwen3-8B (Yang et al., 2025a) on τ^2-Bench and DataSearch. The FOLD and TRM reward models remain frozen during policy optimization, and all policy-learning methods share the same rollout budget and optimization configuration. Further implementation details are provided in Appendix C.5.

Evaluation metrics. We report task success using the environment-specific primary metric: win rate for ALFWorld, database-match success for τ^2-Bench, and task accuracy for DataSearch. To examine whether performance gains arise from more reliable rather than less frequent tool use, we additionally report the invalid-call rate, the redundant-call rate, and the average number of tool calls.

Results. As shown in Table 2, FOLD achieves the highest average task success across the three environments. It improves average task success by 9.5 points over outcome-only GRPO and by 3.7 points over RWML. These gains are accompanied by lower invalid-call and redundant-call rates with a comparable number of tool calls, suggesting that FOLD improves the quality of tool use rather than merely reducing its frequency.

Table 2: Downstream policy performance across three tool-use environments. Task success is reported using the primary metric of each environment (%). "Avg." is the macro-average across environments; the final three columns report tool-use behavior averaged across environments.

	Task success ↑				Tool-use behavior		
Method	ALFWorld	τ^2-Bench	DataSearch	Avg.	Invalid ↓	Redundant ↓	Calls
SFT	34.4	33.3	63.4	43.7	16.4	12.8	5.3
ReAct	13.0	31.9	52.6	32.5	18.5	14.2	6.1
GRPO	77.6	38.0	76.8	64.1	13.2	9.8	5.0
RWML	87.9	43.7	78.2	69.9	11.6	8.4	4.9
TRM	83.6	44.6	79.5	69.2	10.8	7.6	4.8
FOLD	**90.2**	**47.9**	**82.6**	**73.6**	**9.4**	**6.5**	4.9

Table 3: Best-of-N tool-selection hit rate (%). "R" denotes reward-head scoring; "T+V" denotes transition and value-head scoring.

Selection method	ALFWorld	τ^2-Bench	DataSearch	Avg.	Scoring components
Random	18.6	56.9	50.3	41.9	–
Reward-only	90.5	94.3	90.7	91.8	R
FOLD look-ahead	**95.8**	**98.4**	**93.0**	**95.7**	R + T + V

4.3 FOLD Latent Look-Ahead Improves Tool Selection (RQ3)

Selection protocol. Tool-using agents must select a call from a set of available candidates. Given a candidate set $\mathcal{C}_t = \{a_t^{(1)}, \ldots, a_t^{(N)}\}$, we compare three selection strategies: *random* selection from $\mathcal{C}_t$; *reward-only* reranking, which selects $\arg\max_i R_\psi(z_t^r, a_t^{(i)})$; and FOLD *look-ahead*, which selects the candidate with the highest score in Eq. (17).

Evaluation metrics. We evaluate selection with a Best-of-N hit rate. For each held-out candidate set $\mathcal{C}_t$ with process labels $\{y_t^{(i)}\}_{i=1}^N$, a selection $a_t^{(i^*)}$ is correct if it is among the highest-labeled candidates:

$$\mathrm{Hit@}N = \frac{1}{|\mathcal{D}_{\mathrm{dev}}|} \sum_{\mathcal{C}_t \in \mathcal{D}_{\mathrm{dev}}} \mathbb{1}\left[y_t^{(i^*)} = \max_i y_t^{(i)}\right], \tag{19}$$

where $\mathcal{D}_{\mathrm{dev}}$ denotes the held-out collection of candidate sets. This metric measures whether a selector identifies a best available tool call, while treating tied best calls as equally correct.

Results. Table 3 shows that reward-only selection substantially outperforms random selection, indicating that the learned process reward can improve tool selection directly at test time. FOLD look-ahead further improves the hit rate in all three environments, from 90.5% to 95.8% on ALFWorld, 94.3% to 98.4% on τ^2-Bench, and 90.7% to 93.0% on DataSearch. The average gain of 3.9 points shows that incorporating predicted consequences provides information beyond immediate reward scores. Notably, these gains require neither additional RL training nor LLM generation at test time: look-ahead only adds transition and value-head scoring to the reward-model pass.

5 Conclusion

In this paper, we presented FOLD, a factored world-modeling framework for tool-using agents. FOLD partitions the latent interaction state by action dependence and reward relevance, and jointly learns structured transitions, process rewards, and state values. The learned reward model supports policy optimization, while the transition and value models enable test-time latent look-ahead. We further establish conditions for blockwise identifiability. Experiments on ALFWorld, τ^2-Bench, and DataSearch show that FOLD improves tool-call assessment, end-to-end tool use, and tool selection. Future work will extend FOLD to uncertainty-aware multi-step imagination and evolving tool environments.

AI USE STATEMENT

Generative AI tools were used to assist with manuscript language editing and implementation support, including code drafting and debugging. We have reviewed all AI-assisted work and take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeshwar, Sherjil Ozair, Yoshua Bengio, Aaron Courville, and Devon Hjelm. Mutual information neural estimation. In *International conference on machine learning*, pp. 531–540. PMLR, 2018.

Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

Hyungjoo Chae, Namyoung Kim, Kai Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. Web agents with world models: Learning and leveraging environment dynamics in web navigation. In *International Conference on Learning Representations*, volume 2025, pp. 63707–63738, 2025.

Lingxi Cui, Guanyu Jiang, Huan Li, Ke Chen, Lidan Shou, and Gang Chen. Tablecopilot: A table assistant empowered by natural language conditional table discovery. *arXiv preprint arXiv:2507.08283*, 2025.

Shangmin Guo, Omar Darwiche Domingues, Raphaël Avalos, Aaron Courville, and Florian Strub. World modelling improves language model agents. *arXiv preprint arXiv:2506.02918*, 2025.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2(3):440, 2018.

Jinchao Hu, Meizhi Zhong, Kehai Chen, Xuefeng Bai, and Min Zhang. Agentic tool use in large language models. *arXiv preprint arXiv:2604.00835*, 2026.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Imene Kerboua, Sahar Omidi Shayegan, Megh Thakkar, Xing Han Lù, Léo Boisvert, Massimo Caccia, Jérémy Espinas, Alexandre Aussem, Véronique Eglin, and Alexandre Lacoste. Focusagent: Simple yet effective ways of trimming the large context of web agents. *arXiv preprint arXiv:2510.03204*, 2025.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, et al. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. In *International Conference on Learning Representations*, volume 2025, pp. 28691–28735, 2025.

Xiaoxi Li, Jiajie Jin, Guanting Dong, Hongjin Qian, Yongkang Wu, Ji-Rong Wen, Yutao Zhu, and Zhicheng Dou. Webthinker: Empowering large reasoning models with deep research capability. *Advances in Neural Information Processing Systems*, 38:120091–120131, 2026.

Da Ma, Ziyue Yang, Hongshen Xu, Haotian Fang, Kai Yu, and Lu Chen. Empowering llm tool invocation with tool-call reward model. In *The Fourteenth International Conference on Learning Representations*, 2026.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tur, Gokhan Tur, and Heng Ji. Toolrl: Reward is all tool learning needs. *Advances in Neural Information Processing Systems*, 38:105523–105553, 2026.

Jonathan Richens, David Abel, Alexis Bellot, and Tom Everitt. General agents contain world models. *arXiv preprint arXiv:2506.01622*, 2025.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

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

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 1279–1297, 2025.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

Ruipeng Wang, Yuxin Chen, Yukai Wang, Chang Wu, Junfeng Fang, Xiaodong Cai, Qi Gu, Hui Su, An Zhang, Xiang Wang, et al. Agentnoisebench: Benchmarking robustness of tool-using llm agents under noisy condition. *arXiv preprint arXiv:2602.11348*, 2026.

Zhiheng Xi, Chenyang Liao, Guanyu Li, Zhihao Zhang, Wenxiang Chen, Binghai Wang, Senjie Jin, Yuhao Zhou, Jian Guan, Wei Wu, et al. Agentprm: Process reward models for llm agents via step-wise promise and progress. In *Proceedings of the ACM Web Conference 2026*, pp. 4184–4195, 2026.

Haoyuan Xu, Chang Li, Xinyan Ma, Xianhao Ou, Zihan Zhang, Tao He, Xiangyu Liu, Zixiang Wang, Jiafeng Liang, Zheng Chu, et al. The evolution of tool use in llm agents: From single-tool call to multi-tool orchestration. *arXiv preprint arXiv:2603.22862*, 2026.

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, 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. *arXiv preprint arXiv:2412.15115*, 2024.

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

Ke Yang, Yao Liu, Sapana Chaudhary, Rasool Fakoor, Pratik A Chaudhari, George Karypis, and Huzefa Rangwala. Agentoccam: A simple yet strong baseline for llm-based web agents. In *International Conference on Learning Representations*, volume 2025, pp. 97533–97565, 2025b.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Xiao Yu, Baolin Peng, Ruize Xu, Yelong Shen, Pengcheng He, Suman Nath, Nikhil Singh, Jiangfeng Gao, and Zhou Yu. Reinforcement world model learning for llm-based agents. *arXiv preprint arXiv:2602.05842*, 2026.

Kai Zhang, Xiangchao Chen, Bo Liu, Tianci Xue, Zeyi Liao, Zhihan Liu, Xiyao Wang, Yuting Ning, Zhaorun Chen, Xiaohan Fu, et al. Agent learning via early experience. *arXiv preprint arXiv:2510.08558*, 2025.

A PROOFS OF BLOCKWISE IDENTIFIABILITY

This section presents the complete proofs of the identifiability result stated in Section 3.4. We first introduce the formal definitions required by the analysis, including the four-block partition of the latent state and the separation conditions that the two auxiliary variables satisfy (Section A.1). Building on these definitions, we then establish an auxiliary lemma showing that a smoothly separating variable identifies a single relevant–ancillary split up to C^1 diffeomorphisms (Section A.2). Finally, we prove Theorem 1: applying the lemma to the action and return splits yields two complementary Jacobian patterns whose intersection is block diagonal, leaving only within-block reparameterizations (Section A.3).

A.1 DEFINITIONS FOR IDENTIFIABILITY ANALYSIS

All statements are at the population level and at a fixed time t; the time subscript is omitted when unambiguous. Equalities of random variables and conditional-independence statements hold almost surely. The conditional density kernels below are fixed structural components of the models on their stated domains; they are not chosen only up to null-set modifications after the models are compared.

Definition 3 (Four functional blocks)**.** *The latent state is partitioned as*

$$Z_t = \big(Z_t^{ar}, Z_t^{\bar{a}r}, Z_t^{a\bar{r}}, Z_t^{\bar{a}\bar{r}}\big). \tag{20}$$

Superscripts a and $\bar{a}$ denote sensitivity and insensitivity to the preceding action intervention; r and $\bar{r}$ denote relevance and irrelevance to the return variable. Define

$$Z_t^{a} := \big(Z_t^{ar}, Z_t^{a\bar{r}}\big), \qquad Z_t^{\bar{a}} := \big(Z_t^{\bar{a}r}, Z_t^{\bar{a}\bar{r}}\big), \tag{21}$$

$$Z_t^{r} := \big(Z_t^{ar}, Z_t^{\bar{a}r}\big), \qquad Z_t^{\bar{r}} := \big(Z_t^{a\bar{r}}, Z_t^{\bar{a}\bar{r}}\big). \tag{22}$$

The candidate $\widehat{Z}_t$ uses the same block indexing.

Definition 4 (Conditional score span)**.** *Consider an auxiliary variable X with context C and a split $Z = (U, V) \in \mathcal{U} \times \mathcal{V}$, with $d_U := \dim U$ and $d_V := \dim V$. Fix a reference measure for X (counting measure in the discrete case) and a set $\mathcal{I} \subseteq \mathcal{X} \times \mathcal{X} \times \mathcal{C}$. For every $\alpha = (x, x', c) \in \mathcal{I}$, assume that the ground-truth model specifies a conditional-density kernel on the full latent domain such that*

$$0 < p(x \mid z, c) < \infty, \qquad 0 < p(x' \mid z, c) < \infty \qquad \text{for every } z \in \mathcal{U} \times \mathcal{V}, \tag{23}$$

and define

$$\ell_\alpha(z) := \log \frac{p(x \mid z, c)}{p(x' \mid z, c)}. \tag{24}$$

If these ratios are C^1 in z, define

$$\mathscr{S}_{X|C}(z) := \operatorname{span}_{\alpha \in \mathcal{I}} \{\nabla_z \ell_\alpha(z)\}, \qquad E_U^* := \left\{ \begin{pmatrix} \xi_U \\ 0_{d_V} \end{pmatrix} : \xi_U \in \mathbb{R}^{d_U} \right\}. \tag{25}$$

Thus E_U^* is the d_U-dimensional coordinate subspace with zero V component.

Definition 5 (Smooth separation)**.** *Under Definition 4, let $\mathcal{H}$ be a family of admissible C^1 diffeomorphisms from $\mathcal{U} \times \mathcal{V}$ to product domains with block dimensions (d_U, d_V), fixed for the model class before any candidate is selected. The family contains every transformation induced by a candidate model in the class under consideration. Let*

$$h = (\widehat{U}, \widehat{V}) : \mathcal{U} \times \mathcal{V} \longrightarrow \widehat{\mathcal{U}} \times \widehat{\mathcal{V}} \tag{26}$$

belong to $\mathcal{H}$. The variable X with context C smoothly separates *(U, V) relative to $(h, \mathcal{H})$ if:*

(S1) ***Complete relevant score.*** *The true likelihood ratios are C^1 and*

$$\mathscr{S}_{X|C}(z) = E_U^* \qquad \text{for } P_Z\text{-almost every } z. \tag{27}$$

(S2) ***Likelihood-ratio compatibility.*** *For $\alpha = (x, x', c) \in \mathcal{I}$, the candidate model $\widehat{M}$ specifies a conditional-density kernel $\widehat{p}$ with respect to the same reference measure and the ratio*

$$\widetilde{\ell}_\alpha(\widehat{u}) := \log \frac{\widehat{p}(x \mid \widehat{U} = \widehat{u}, C = c)}{\widehat{p}(x' \mid \widehat{U} = \widehat{u}, C = c)}. \tag{28}$$

The two candidate densities are finite and strictly positive on $\widehat{\mathcal{U}}$, $\widetilde{\ell}_\alpha$ is C^1, and

$$\ell_\alpha(z) = \widetilde{\ell}_\alpha\big(\widehat{U}(z)\big) \tag{29}$$

for every $\alpha \in \mathcal{I}$ and every $z \in \mathcal{U} \times \mathcal{V}$.

(S3) ***Strong candidate ancillarity.***

$$X \perp\!\!\!\perp \widehat{V} \mid C. \tag{30}$$

(S4) ***Ancillary rigidity of the candidate class.*** *For every $k = (k_U, k_V) \in \mathcal{H}$,*

$$X \perp\!\!\!\perp k_V(Z) \mid C \quad \Longrightarrow \quad D_U k_V(Z) = 0_{d_V \times d_U} \quad P_Z\text{-almost surely.} \tag{31}$$

Definition 6 (Auxiliary variables)**.** *Let E_t^a be the observed assignment in a randomized intervention on A_{t-1}, and let C_t^a contain only pre-intervention variables. Under conditional randomization, positivity, consistency, and no interference,*

$$\mathcal{L}(H_t \mid E_t^a = a, C_t^a = c) = \mathcal{L}(H_t \mid \mathrm{do}(A_{t-1} = a), C_t^a = c) \tag{32}$$

for every admissible (a, c). The two auxiliary variables are defined by

$$(X_t^a, C_t^a, U_t^a, V_t^a) := (E_t^a, C_t^a, Z_t^a, Z_t^{\bar{a}}), \tag{33}$$

$$(X_t^r, C_t^r, U_t^r, V_t^r) := (G_t, C_t^r, Z_t^r, Z_t^{\bar{r}}), \tag{34}$$

where

$$G_t := \sum_{j=0}^{T-t} \gamma^j R_{t+j}. \tag{35}$$

The observable contexts C_t^a and C_t^r are prespecified and held fixed across candidate models. Define $(\widehat{U}_t^e, \widehat{V}_t^e)$ analogously.

A.2 Auxiliary Lemma

Lemma 1 (Identification of one relevant–ancillary split)**.** *Let $\mathcal{U}, \mathcal{V}, \widehat{\mathcal{U}}$, and $\widehat{\mathcal{V}}$ be nonempty connected open subsets of Euclidean spaces. Suppose that P_Z has full support relative to $\mathcal{U} \times \mathcal{V}$, meaning that every nonempty open subset of the product domain has positive probability, and that*

$$h = (\widehat{U}, \widehat{V}) \in \mathcal{H} \tag{36}$$

is a C^1 diffeomorphism between the two product domains, with $\dim \widehat{U} = d_U$ and $\dim \widehat{V} = d_V$. If (X, C) smoothly separates (U, V) relative to $(h, \mathcal{H})$, then there are C^1 diffeomorphisms ϕ and ψ such that

$$\widehat{U} = \phi(U), \qquad \widehat{V} = \psi(V). \tag{37}$$

Proof. By (S2) and the chain rule, for every $\alpha \in \mathcal{I}$,

$$\nabla_z \ell_\alpha(z) = D\widehat{U}(z)^\top \nabla_{\widehat{u}} \widetilde{\ell}_\alpha\big(\widehat{U}(z)\big). \tag{38}$$

Hence, by (S1),

$$E_U^* = \mathscr{S}_{X|C}(z) \subseteq \mathrm{im}\big(D\widehat{U}(z)^\top\big) \qquad P_Z\text{-almost surely.} \tag{39}$$

The rows of $D\widehat{U}(z)$ are linearly independent because they are a subset of the rows of the nonsingular matrix $Dh(z)$. Both spaces above have dimension d_U, so they are equal. Thus

$$D_V \widehat{U}(Z) = 0_{d_U \times d_V} \qquad P_Z\text{-almost surely.} \tag{40}$$

Continuity of $D\widehat{U}$ and relative full support extend this equality to the entire product domain. Since $\mathcal{V}$ is connected, $\widehat{U}(u, v)$ is constant in v; hence $\widehat{U} = \phi(U)$ for a C^1 map ϕ.

By (S3), $X \perp\!\!\!\perp \widehat{V}(Z) \mid C$. Applying (S4) to $h \in \mathcal{H}$ gives

$$D_U \widehat{V}(Z) = 0_{d_V \times d_U} \qquad P_Z\text{-almost surely.} \tag{41}$$

The same argument makes this equality pointwise, and connectedness of $\mathcal{U}$ gives $\widehat{V} = \psi(V)$ for a C^1 map ψ.

Consequently,

$$Dh(u,v) = \begin{bmatrix} D\phi(u) & 0_{d_U \times d_V} \\ 0_{d_V \times d_U} & D\psi(v) \end{bmatrix}. \tag{42}$$

Since Dh is nonsingular, both diagonal blocks are nonsingular. Moreover, $h(u,v) = (\phi(u), \psi(v))$ is bijective between product domains. Its injectivity implies that ϕ and ψ are injective, and its surjectivity implies that both maps are surjective. The inverse function theorem gives local C^1 inverses, which coincide with the global inverses by bijectivity. Thus ϕ and ψ are C^1 diffeomorphisms. □

A.3 Proof of Theorem 1

Proof. Fix a population-equivalent candidate in the class under consideration. By (A2), $\widehat{Z}_t = h(Z_t)$ with $h = \widehat{g}^{-1} \circ g$, a global C^1 diffeomorphism. Applying Lemma 1 to the action split gives

$$\widehat{Z}_t^a = h_a(Z_t^a), \qquad \widehat{Z}_t^{\bar{a}} = h_{\bar{a}}(Z_t^{\bar{a}}). \tag{43}$$

Applying it to the return split gives

$$\widehat{Z}_t^r = h_r(Z_t^r), \qquad \widehat{Z}_t^{\bar{r}} = h_{\bar{r}}(Z_t^{\bar{r}}). \tag{44}$$

Order the input and output blocks as $(ar, \bar{a}r, a\bar{r}, \bar{a}\bar{r})$. The action split implies the Jacobian pattern

$$\begin{bmatrix} * & 0 & * & 0 \\ 0 & * & 0 & * \\ * & 0 & * & 0 \\ 0 & * & 0 & * \end{bmatrix}, \tag{45}$$

whereas the return split implies

$$\begin{bmatrix} * & * & 0 & 0 \\ * & * & 0 & 0 \\ 0 & 0 & * & * \\ 0 & 0 & * & * \end{bmatrix}. \tag{46}$$

Their intersection is block diagonal. Therefore, for each $b \in \mathcal{B}$, the derivative of $\widehat{Z}_t^b$ with respect to every other input block is zero. The product of the other three connected domains is connected, so there is a C^1 map h_b such that

$$\widehat{Z}_t^b = h_b(Z_t^b). \tag{47}$$

Thus

$$Dh(z) = \operatorname{diag}\big(Dh_{ar}, Dh_{\bar{a}r}, Dh_{a\bar{r}}, Dh_{\bar{a}\bar{r}}\big). \tag{48}$$

Each Dh_b is nonsingular because Dh is nonsingular. Fixing the other three input blocks and using injectivity of h shows that h_b is injective. Fixing arbitrary target values of the other three candidate blocks and using surjectivity of h onto the candidate product domain shows that h_b is surjective. Hence each h_b is bijective. The inverse function theorem gives local C^1 inverses, which coincide with the global inverse by bijectivity. Therefore every h_b is a C^1 diffeomorphism. □

B Notation

Table 4 summarizes the notation used throughout the paper; symbols introduced only for the identifiability analysis are defined in Appendix A.1.

C Implementation Details

C.1 Environment Details

Figure 5 summarizes the three evaluation environments. They differ in how the agent observes and acts: ALFWorld operates through discrete textual actions, τ^2-Bench through dialogue interleaved with API calls, and DataSearch through catalog-navigation commands. We describe each environment below.

Table 4: Notation used throughout the paper.

Symbol	Description
$\mathcal{M}$	Partially observable tool-interaction process.
s_t, o_t, H_t	Environment state, textual observation, and interaction history at step t.
a_t	Structured tool call at step t.
π_θ	Tool-using policy.
r_t, G_t	Step reward and discounted return.
γ	Discount factor.
$\mathcal{C}_t$	Candidate tool calls considered at step t.
N	Number of candidates considered at test time.
$h_t = f_\varphi(H_t)$	Language-model representation of the interaction history.
q_α	Representation model mapping h_t to the latent state.
z_t	Latent state before factorization.
$z_t^{a r}, z_t^{\bar{a} r}$	Action-dependent and action-independent reward-relevant factors.
$z_t^{a \bar{r}}, z_t^{\bar{a} \bar{r}}$	Action-dependent and action-independent reward-irrelevant factors.
$z_t^r, z_t^{\bar{r}}$	Aggregated reward-relevant and reward-irrelevant states.
$\mathcal{B}$	Block index set $\{ar, \bar{a}r, a\bar{r}, \bar{a}\bar{r}\}$.
p_ω, p_ψ	Transition and reward densities in the world-model formulation of Eq. (5).
T_ω, R_ψ, V_ν	Latent transition, process-reward, and state-value functions.
$\hat{z}_{t+1}^r$	Predicted reward-relevant successor state.
$S(a_t^{(i)})$	Look-ahead score of candidate $a_t^{(i)}$.
$\tilde{r}_t, r^{\text{out}}$	Turn-level process reward and outcome reward for policy optimization.

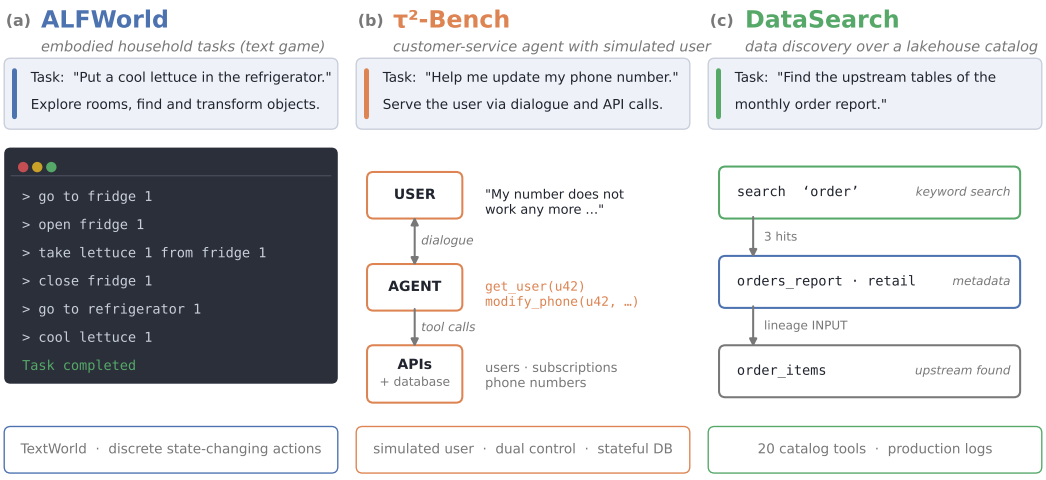


Figure 5: Overview of the three evaluation environments. (a) ALFWorld: a text-game agent completes household tasks through discrete state-changing actions. (b) τ^2-Bench: a dual-control agent serves a simulated user by issuing API calls against a stateful database. (c) DataSearch: a catalog-navigation agent locates tables and traces lineage through catalog tools over production logs.

ALFWorld. ALFWorld (Shridhar et al., 2020) casts embodied household tasks as text games built on TextWorld. The agent receives a natural-language instruction, e.g., *"put a cool lettuce in the refrigerator"*, and completes it by navigating rooms and manipulating objects through discrete textual actions such as `go to fridge 1`, `take lettuce 1`, and `cool lettuce 1`. Episodes are long (16–40 turns) and each action deterministically changes the environment state, so the agent must track task progress over an extended horizon, which makes this environment suitable for evaluating latent dynamics.

τ^2-Bench. τ^2-Bench (Barres et al., 2025) evaluates dual-control customer-service agents. The agent serves a simulated user, e.g., by updating a phone number or managing a subscription, by issuing API calls against a stateful database, while the user follows a policy over the same database and can also act on the environment. Success is judged by whether the database reaches the desired

final state, so the agent must both conduct natural multi-turn dialogue and issue correct calls. We use the telecom split, which contains the largest task pool.

DataSearch. DataSearch is a real-world data-navigation dataset built from logs of a production data-catalog agent. Given a business request, e.g., locating the tables behind a report or tracing the upstream dependencies of a table, the agent explores a lakehouse catalog through 20 tools covering keyword search, hierarchical browsing, metadata retrieval, lineage queries, and content reading, following the workflow `search` $\rightarrow$ `get` $\rightarrow$ `cat`. Tool responses are heterogeneous—search hits, schema listings, lineage graphs, and table contents—and a large fraction of returned fields is irrelevant to the task, which is the setting that most directly exercises the four-block factorization.

C.2 Process-Reward Label Construction

Training data for FOLD is organized into candidate groups. At a decision point t, a group $g_t = \big(H_t, \{(a_t^{(i)}, y_t^{(i)})\}_{i=1}^{n}\big)$ consists of the shared interaction history H_t and $n \in [2, 10]$ candidate calls, each associated with a process label $y_t^{(i)}$. For executed candidates, the group additionally records the resulting post-execution history $H_t^+(a_t^{(i)})$, which supervises the transition model. Train–dev splits are performed at the group level to prevent leakage across shared histories. The process labels are constructed per environment as follows.

ALFWorld. Labels are derived from the symbolic environment state. Each candidate is executed in the simulator, and the fact difference $\Delta F = F^+ \cup \{\neg f : f \in F^-\}$ between the pre- and post-execution predicate sets is mapped by a fixed rule set to an ordinal utility,

$$y = \begin{cases} 0, & \text{invalid or harmful (no effect, or destroying the goal),} \\ 0.5, & \text{neutral (executed without task progress),} \\ 1.0, & \text{helpful subgoal (e.g., taking the target object),} \\ 1.5, & \text{helpful navigation (reaching a required region),} \\ 2.0, & \text{partial goal (correct object, wrong state or receptacle),} \\ 3.0, & \text{full goal completion.} \end{cases} \tag{49}$$

Since the ordinal levels induce a soft ranking rather than a binary one, they support the listwise objective below.

τ^2-Bench. Each step actually executed by the environment receives a step-level score from an LLM judge, combined with the final outcome:

$$y_t = \min\Big(1,\ \frac{\mathrm{judge}_t}{10} + 0.3 \cdot \mathbb{1}\{\text{final step} \wedge \text{task won}\}\Big), \tag{50}$$

where $\mathrm{judge}_t \in [0, 10]$ (0.5 when unavailable). Executed calls are supplemented with negatives: hard negatives keep the tool of the executed call but perturb its arguments, and easy negatives are sampled uniformly from the tool pool; both receive $y = 0$.

DataSearch. Each executed call inherits its session feedback score, which may be negative for redundant operations (e.g., a repeated query that consumes quota), and calls sampled with different tool names receive $y = 0$. Group labels therefore contain frequent ties, which is also why the evaluation protocol operates on strictly ordered pairs.

All three models are trained on the same groups. Writing $s_t^{(i)} := R_\psi(z_t^r, a_t^{(i)})$, the reward model minimizes the listwise cross-entropy

$$\mathcal{L}_{\mathrm{rew}} = -\sum_{i=1}^{n} \mathrm{softmax}(\mathbf{y}_t/\tau)_i \, \log \mathrm{softmax}(\mathbf{s}_t/\tau)_i, \tag{51}$$

with the temperature τ fixed to one, so that ordinal labels induce soft targets over candidates. The transition model regresses the stop-gradient successor encoding $\mathrm{sg}\left(q_\alpha(H_t^+(a_t^{(i)}))\right)$ block by block, and the value model regresses the group-optimal label $\max_i y_t^{(i)}$ as the return proxy. The evaluation metrics computed on these groups are formalized in Appendix C.3.

C.3 Evaluation Metrics

The model-quality metrics of RQ1 are computed on the dev candidate groups (Appendix C.2). We reuse the scores $s_t^{(i)} = R_\psi(z_t^r, a_t^{(i)})$ and labels $y_t^{(i)}$, and write

$$z_t^+ = \mathrm{sg}\left(q_\alpha(H_t^+)\right) \tag{52}$$

for the encoded post-execution state of an executed candidate.

Process reward: pairwise accuracy. Within each group, ranking quality is measured over strictly ordered label pairs, and macro-averaged over the dev groups $\mathcal{G}_{\mathrm{dev}}$:

$$\mathrm{PW}(g_t) = \frac{\left|\{(i,j) \,:\, y_t^{(i)} > y_t^{(j)} \ \wedge \ s_t^{(i)} > s_t^{(j)}\}\right|}{\left|\{(i,j) \,:\, y_t^{(i)} > y_t^{(j)}\}\right|}, \qquad \mathrm{Acc}_{\mathrm{pair}} = \frac{1}{|\mathcal{G}_{\mathrm{dev}}|} \sum_{g \in \mathcal{G}_{\mathrm{dev}}} \mathrm{PW}(g). \tag{53}$$

Strict inequalities exclude tied labels, which are frequent in DataSearch, so each group contributes only pairs with an unambiguous target ordering.

Latent dynamics: cosine similarity. For each executed candidate, the predicted successor $\hat{z}_{t+1} = T_\omega(z_t, a_t)$ is compared with the encoded true successor z_t^+:

$$\cos = \frac{\hat{z}_{t+1}^\top z_t^+}{\|\hat{z}_{t+1}\|_2 \, \|z_t^+\|_2}, \tag{54}$$

averaged over all executed candidates in the dev split.

State value: Spearman correlation. Writing $G_t^\star := \max_i y_t^{(i)}$ for the group-optimal label that serves as the return proxy (Appendix C.2), value quality is measured by the Spearman rank correlation between the predicted values and the proxies across all dev decision points:

$$\rho_s = \rho\big(\{V_\nu(z_t^r)\}_{t \in \mathrm{dev}}, \ \{G_t^\star\}_{t \in \mathrm{dev}}\big). \tag{55}$$

Semantic-event probe. Probing evaluates whether the predicted successor preserves task-relevant state changes rather than merely matching the latent geometry. A linear probe

$$\hat{e}(v) = \arg\max_{e \in \mathcal{E}} \ \mathrm{softmax}(Wv + b)_e \tag{56}$$

is trained on frozen representations to predict the next semantic event $e_{t+1} \in \mathcal{E}$, where $\mathcal{E}$ is an environment-defined event set (e.g., container-state changes in ALFWorld), and we report accuracy and macro-F1. The probe is trained and evaluated in two conditions: on the predicted successor $\hat{z}_{t+1}$ and, as a control, on the current state z_t without applying the transition. A substantially higher accuracy on the former indicates that the transition model injects the information required to anticipate semantic changes rather than copying the current state.

C.4 Turn-Level GRPO with Process Rewards

This section presents the complete formulation for policy optimization with FOLD's process rewards, following the turn-level credit assignment of Ma et al. (2026).

Rollout construction. Given a query, GRPO samples a group $\mathcal{G}$ of rollouts from the current policy $\pi_{\theta_{\mathrm{old}}}$. Rollout $\tau^{(g)} \in \mathcal{G}$ consists of T_g tool-call turns followed by a final-answer turn, with turn-level rewards $\tilde{r}_t^{(g)}$ defined in Eq. (15). Each rollout is generated as a token sequence $x_1^{(g)}, \ldots, x_{L_g}^{(g)}$, where only the tokens produced by the policy—thoughts, tool calls, and the final answer—receive gradients; environment-returned observations are appended to the context without contributing policy tokens. Since all FOLD parameters are frozen, each process reward adds a single forward pass of the encoder and reward head over the interaction history up to the corresponding turn.

Turn-level advantage estimation. For each turn t, the rewards of all rollouts that reach turn t are collected as

$$\mathcal{R}_t = \big\{\tilde{r}_t^{(g)} \;\big|\; \tau^{(g)} \in \mathcal{G},\; t \le T_g\big\}, \qquad \mathcal{R}^{\text{out}} = \big\{\tilde{r}_{T_g+1}^{(g)} \;\big|\; \tau^{(g)} \in \mathcal{G}\big\}, \tag{57}$$

and normalized independently within each set:

$$\hat{A}_t^{(g)} = \frac{\tilde{r}_t^{(g)} - \text{mean}(\mathcal{R}_t)}{\text{std}(\mathcal{R}_t)}, \qquad \hat{A}_{T_g+1}^{(g)} = \frac{\tilde{r}_{T_g+1}^{(g)} - \text{mean}(\mathcal{R}^{\text{out}})}{\text{std}(\mathcal{R}^{\text{out}})}. \tag{58}$$

Process rewards are therefore compared only with process rewards of the same turn, and separately from the binary outcome reward. This per-turn normalization follows the observation of Ma et al. (2026) that jointly normalizing process and outcome rewards causes the policy to prefer fewer tool calls. Turns reached by a single rollout, or whose rewards coincide across $\mathcal{G}$, yield zero advantage and contribute no gradient.

Token-level objective. The turn-level advantage is broadcast to every policy-generated token of the corresponding turn. Writing $A_j^{(g)}$ for the advantage assigned to token $x_j^{(g)}$, the policy is optimized with the clipped GRPO objective

$$\mathcal{J}(\theta) = \mathbb{E}\left[\frac{1}{|\mathcal{G}|}\sum_{\tau^{(g)}\in\mathcal{G}}\frac{1}{L_g}\sum_{j=1}^{L_g}\Bigg(\min\Big(\rho_j^{(g)}A_j^{(g)},\; \text{clip}\left(\rho_j^{(g)},\,1-\epsilon,\,1+\epsilon\right)A_j^{(g)}\Big) - \beta\, D_{\text{KL}}\big[\pi_\theta \,\big\|\, \pi_{\text{ref}}\big]\Bigg)\right], \tag{59}$$

where $\rho_j^{(g)} = \pi_\theta(x_j^{(g)} \mid \tau_{<j}^{(g)})/\pi_{\theta_{\text{old}}}(x_j^{(g)} \mid \tau_{<j}^{(g)})$ is the importance ratio between the current and the sampling policy, $\tau_{<j}^{(g)}$ denotes the query together with the tokens preceding $x_j^{(g)}$, the KL divergence is evaluated at the same context $\tau_{<j}^{(g)}$, π_{ref} is the reference policy, and ϵ and β are the clipping and KL-regularization coefficients.

C.5 Training Configuration

Training frameworks. Policy optimization is implemented with `veRL` (Sheng et al., 2025), building on the asynchronous tool-calling and process-reward integration released with TRM (Ma et al., 2026), with vLLM-based asynchronous rollouts and FSDP model sharding.

Hardware and training time. All experiments run on three machines with eight 64 GB Hygon DCU accelerators each. Training FOLD takes 1.2 hours for 300 steps.

FOLD training configuration. The default configuration uses Qwen3-4B-Instruct-2507 with LoRA rank 16 on the attention and MLP projections, 16 latent dimensions per block (64 in total), learning rate 2×10^{-5} with cosine decay, 300 training steps, gradient accumulation 8, and bf16 precision. The maximum sequence length is 1024 on ALFWorld and 2048 on the other environments. The 8B comparison uses Qwen3-8B with the remaining hyperparameters unchanged.

GRPO training configuration. Policies are trained with learning rate 1×10^{-6} (5×10^{-7} on ALFWorld), KL coefficient 0.001, four rollouts per group, a batch size of 16 on τ^2-Bench and 32 on DataSearch, at most 30 tool turns per episode, and no learning-rate warmup. ALFWorld policies are initialized from Qwen2.5-7B-Instruct (Yang et al., 2024) and policies for the other environments from Qwen3-8B.

Test-time selection configuration. Unless otherwise stated, the policy samples $N{=}5$ candidate calls, and reward-only reranking and look-ahead receive exactly the same candidates.

D Additional Experiments

D.1 Controlled Block Recovery

Because the ground-truth blocks are unavailable in the evaluation environments, we construct a synthetic environment in which the four factors are known by design, and examine whether FOLD recovers them. The results are reported in Figure 4 in the main text; this section describes the environment, the training configuration, and the evaluation protocol.

Synthetic environment. The ground-truth state consists of four blocks A, B, C, D, each six-dimensional. At each step, an action $a = (\mathrm{id}, \mathrm{fx})$ with a tool index id and a scalar intensity fx updates the state by

$$\begin{aligned} A' &= 0.4\,A + 0.8\,e_{\mathrm{id} \bmod 6} + 0.1\,(\mathrm{fx} - 3)\,\epsilon_{\mathrm{id,fx}}, \\ C' &= 0.3\,C + 0.9\,\epsilon'_{\mathrm{id,fx}}, \\ B' &= B, \qquad D' \sim \mathcal{N}(0, I), \\ r &= w^\top A + v^\top B, \end{aligned} \tag{60}$$

where e_k is the k-th standard basis vector, ϵ and ϵ' are standard Gaussian noise, and w, v are fixed random reward weights. By construction, only A and C respond to the action, the reward reads only A and B, and D is related to neither; the four blocks therefore play the roles of z^{ar}, $z^{\bar{a}r}$, $z^{a\bar{r}}$, and $z^{\bar{a}\bar{r}}$, respectively.

Training. FOLD is trained on this environment with the 4B backbone and the same architecture and objective as in the main experiments. The transition model is supervised by Eq. (60) through the rendered observations, without access to the block identities.

Evaluation protocol. After training, the encoder is frozen. For each of the four learned blocks, a linear probe is trained to regress the corresponding ground-truth block vector (six dimensions), and the coefficient of determination R^2 is computed on held-out steps. This yields a 4×4 recovery matrix: diagonal entries measure how well each ground-truth block is linearly decodable from its assigned latent block, and off-diagonal entries measure cross-block leakage. To isolate the contribution of the structured transition, the probes are applied in two conditions: on the encoded state z_t and on the transition prediction $\hat{z}_{t+1}$. Stronger diagonal recovery for $\hat{z}_{t+1}$ than for z_t indicates that block separation emerges from the transition structure rather than from the encoder alone.

D.2 Ablation Studies

This section ablates the training recipe of FOLD to understand how the transition and value objectives interact with process-reward learning. Since the world model is trained offline on candidate groups, we focus on the model-quality metrics of RQ1: reward pairwise accuracy, transition cosine similarity, and value Spearman correlation.

Settings. All variants are trained on τ^2-Bench with the 4B backbone, sharing the same data and optimization budget. Starting from the listwise reward objective, we consider the following variants:

- **Reward only:** train the reward model alone, without the transition and value objectives.
- **Full FOLD (small weights):** add the transition and value objectives with weights $(\lambda_{\mathrm{dyn}}, \lambda_{\mathrm{val}}) = (0.03, 0.1)$; this is the configuration used in the main experiments.
- **Full FOLD (heavy weights):** the same objectives with weights $(0.1, 0.5)$.
- **+ MI regularizer:** the heavy-weight configuration with the mutual-information regularizer enabled ($\lambda_{\mathrm{mi}} = 0.1$); this variant is analyzed in Section D.4.
- **BT loss:** replace the listwise objective with the Bradley–Terry pairwise loss; this variant concerns the reward-model loss form rather than the recipe, and is analyzed in Section D.3.

Results and analysis. Table 5 summarizes the results. Overall, the full FOLD recipe with small auxiliary weights performs best.

Table 5: Ablation of the training recipe on τ^2-Bench. Reward accuracy is the pairwise accuracy (%); transition quality is the cosine similarity; value quality is the Spearman correlation.

Variant	Reward acc. ↑	Trans. cos ↑	Value ρ ↑
Reward only (listwise)	98.4	–	–
Full FOLD (small weights)	**98.8**	**0.81**	0.83
Full FOLD (heavy weights)	91.5	0.74	0.82
+ MI regularizer (see Section D.4)	93.9	0.80	**0.85**
BT loss (see Section D.3)	97.6	–	–

Auxiliary objectives act as lightweight regularization at small weights. With $(\lambda_{\text{dyn}}, \lambda_{\text{val}}) = (0.03, 0.1)$, adding the transition and value objectives slightly improves reward accuracy over the reward-only model (98.8 vs. 98.4) while learning both auxiliary signals to a useful level (cosine 0.81, ρ 0.83). The three heads thus coexist under the shared backbone without competing for capacity, which supports the multi-task formulation behind the full objective.

D.3 Reward-Model Loss Functions

This section compares alternative loss forms for the reward head, holding the architecture, data, and optimization budget fixed. Besides the listwise objective in Appendix C.2, we consider the Bradley–Terry pairwise loss (Bradley & Terry, 1952)

$$\mathcal{L}_{\text{BT}} = -\sum_{y_t^{(i)} > y_t^{(j)}} \log \sigma\big(s_t^{(i)} - s_t^{(j)}\big), \tag{61}$$

which uses only the binary orderings implied by the labels. On τ^2-Bench, the listwise objective reaches 98.8 versus 97.6 for Bradley–Terry (Table 5). The gap is consistent with the label structure: candidate groups carry ordinal labels with frequent ties, and the listwise objective converts them into soft targets over the full group, whereas Bradley–Terry discards the magnitude of the label differences and the tied candidates. Stepwise binary classification in the style of TRM corresponds to a single-candidate form of the pairwise objective and inherits the same limitation, which is why we adopt listwise training for FOLD.

D.4 Mutual-Information Regularization

Block separation in FOLD is enforced architecturally: the transition model receives the current call only for the action-dependent blocks (Eq. (8)), and the reward and value models read only the reward-relevant state (Eq. (9)). As a complement, separation can be encouraged with two conditional mutual-information objectives,

$$\begin{aligned} J_{RS} &= I(G_t; z_t^r \mid c_t^r) - I(G_t; z_t^{\bar{r}} \mid c_t^r), \\ J_{AS} &= I(a_{t-1}; z_t^a \mid z_{t-1}) - I(a_{t-1}; z_t^{\bar{a}} \mid z_{t-1}), \end{aligned} \tag{62}$$

where $c_t^r = (a_{t-1:t}, \text{sg}(z_{t-1}^r))$. The objective J_{RS} encourages return-related information to enter z_t^r rather than $z_t^{\bar{r}}$, while J_{AS} encourages the effects of the preceding action to enter z_t^a rather than $z_t^{\bar{a}}$. Each mutual-information term is estimated with a Mutual Information Neural Estimator (Belghazi et al., 2018); the estimators are optimized alternately with FOLD, and the conditioning variables are detached when updating the latent representation.

In all main experiments this regularizer is disabled ($\lambda_{RS} = \lambda_{AS} = 0$). Enabling it on the heavy-weight configuration of Table 5 improves reward accuracy by 2.4 points (93.9 vs. 91.5) and value correlation to 0.85, indicating that the regularizer is directionally helpful when the auxiliary weights are large. However, the result remains far below the small-weight configuration without it (98.8), suggesting that once the auxiliary objectives are appropriately weighted, separation is obtained primarily through the factorized architecture.

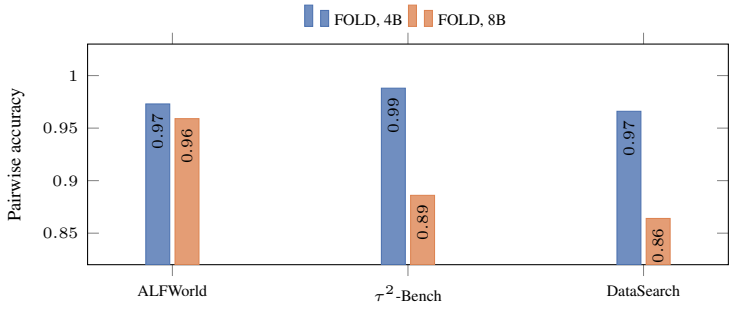


Figure 6: Effect of backbone scale on FOLD reward pairwise accuracy. Both models use the same training data and optimization budget.

D.5 Effect of Backbone Scale

We compare the full FOLD model with Qwen3-4B-Instruct-2507 and Qwen3-8B under the same training data and optimization budget. Figure 6 reports reward pairwise accuracy across the three main evaluation environments.

The 4B backbone matches or exceeds the 8B backbone in every environment, with the largest differences on τ^2-Bench (98.8 vs. 88.6) and DataSearch (96.6 vs. 86.4). Thus, within the evaluated configurations, increasing backbone scale alone does not improve reward modeling. The fixed training budget and latent-head configuration appear sufficient for the smaller backbone, whereas the larger model may require scale-specific tuning to realize its additional capacity.

E Limitations and Discussion

FOLD relies on step-level process supervision and sufficiently diverse tool interactions to distinguish the four functional blocks. Its identifiability result further assumes an informative observation representation, correct structural specification, and adequate variability—conditions that may hold only approximately for pretrained language-model features. The current look-ahead procedure predicts a single successor state rather than performing long-horizon imagined rollouts, and its benefit is bounded by the recall of the candidate generator and the accuracy of the learned transition and value functions. Finally, training and applying multiple latent heads introduces additional computation. Extending FOLD to uncertainty-aware multi-step planning and continually changing tool sets is an important direction for future work.