RETAINING HISTORY IS NOT MAINTAINING STATE: STATECOMMIT FOR PERSISTENT WORLD MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Persistent world modeling under partial observability requires maintaining interaction-established states that may remain relevant after they are no longer directly observable. Existing approaches commonly preserve past interaction evidence through long context, summaries, retrieval, or recurrent memory, and reconstruct the needed state when a query or prediction is made. However, retaining evidence does not guarantee a coherent representation of what is currently true, as accurate recovery of an individual state can coexist with substantial whole-state inconsistency. To this end, we introduce **`StateCommit`**, which shifts state resolution from read time to interaction time and incrementally maintains a query-independent *Active State*. Specifically, interaction evidence is converted into explicit state transactions that are verified before a deterministic runtime commits persistent updates. For visual prediction, slot-indexed *Visual Bindings* further ground the maintained semantic state in the prediction view induced by the known next action. We also introduce **`StateReturn`**, a multimodal benchmark for whole-state maintenance and state-consistent future prediction. On **`StateReturn`**, Full-History retains 98.61% target-state accuracy after 16 intervening interactions but only 13.89% full-state exact match, whereas **`StateCommit`** achieves 98.67–100.00% full-state exact match and reduces state-sensitive prediction error by 58.1% relative to the best memory baseline. A quick visual overview of this work is available at https://statecommit.github.io/.

1 INTRODUCTION

Interactive world models seek to predict how an environment evolves in response to an agent's actions (Hafner et al., 2019a;b; Bruce et al., 2024; Alonso et al., 2024; Chen et al., 2026). Unlike passive video prediction, interaction can alter the underlying world state rather than only its immediate appearance (Oh et al., 2015; Finn et al., 2016). Such changes may persist beyond the moment of interaction, so the current observation may no longer reveal a state that still affects what happens next (Kaelbling et al., 1998). As illustrated in Figure 1, two trajectories can reach the same current observation and receive the same next action yet require different futures because earlier interactions established different persistent states. Persistent world modeling therefore requires maintaining interaction-established states even after they are no longer directly observable.

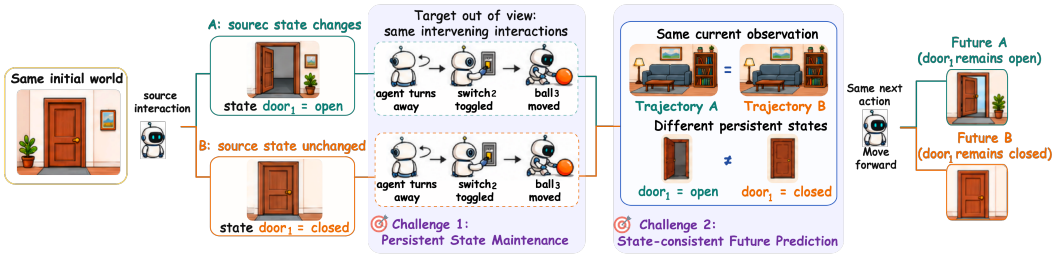


Figure 1: Persistent state under partial observability. Two trajectories establish different states for the same door and later reach the same current observation. Under the same next action, the persistent state established earlier leads to different futures.

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.

A natural strategy is to retain past interaction evidence through full context, retrieval, summaries, or recurrent memory (Dai et al., 2019; Park et al., 2023; Packer et al., 2023; Hu et al., 2025). However, retaining evidence is not the same as maintaining state: the model must still resolve accumulated interactions into a coherent representation of what is currently true. Recovering a single queried variable can be easier because only the interactions relevant to that variable need to be resolved, whereas reconstructing the tracked state as a whole requires determining the latest valid value of every tracked variable across interleaved and potentially superseding interactions. Our results show that this gap persists even when no history is discarded: at the longest interaction setting in our visual benchmark, Full-History retains 98.61% target-state accuracy while achieving only 13.89% full-state exact match. This leads to our first research question: ❶ **How can a persistent world model maintain a coherent current state as interactions unfold?**

To address this question, we introduce `StateCommit`, which maintains state as interactions unfold rather than reconstructing it from accumulated history. After each interaction, `StateCommit` interprets the newly available evidence and incrementally updates the tracked variables in a query-independent *Active State*. Because an incorrect update can affect later reasoning and prediction, state changes are proposed explicitly and verified before a deterministic runtime commits them.

Maintaining a coherent current state addresses only part of persistent world modeling: the maintained state must also be reflected in future observations. Active State records what is currently true about a tracked entity, but it does not specify how that state should appear in the prediction view induced by the next public action (Oh et al., 2015; Wu et al., 2022; Villar-Corrales et al., 2023; Kipf et al., 2021). This raises our second research question: ❷ **How can maintained semantic state be grounded in the corresponding prediction view so that it is correctly reflected in the predicted future?** To address this question, `StateCommit` associates tracked state with slot-indexed *Visual Bindings* and projects these bindings into the prediction view before rendering.

These two questions also reveal corresponding gaps in evaluation: correctly answering an individual state query does not establish whole-state coherence, while favorable full-frame reconstruction does not ensure that a predicted future reflects the correct persistent state. Existing benchmarks have evaluated long-term memory and future prediction from complementary perspectives (Pasukonis et al., 2022; Yi et al., 2019; Bear et al., 2021; Maharana et al., 2024; Tan et al., 2025), but do not directly couple whole-state maintenance with the requirement that the maintained persistent state be reflected in a subsequent prediction. We therefore introduce `StateReturn`, a multimodal benchmark for evaluating whole-state maintenance and state-consistent future prediction under controlled partial observability. *State Maintenance* tests whether the tracked current state can be recovered coherently, and is instantiated from both textual streams and RGB observations with actions. *State-Consistent Prediction* further tests whether the predicted next observation reflects the persistent state established earlier in the trajectory. Our main contributions are summarized as follows:

- ***Current-State Perspective.*** We distinguish retaining past interaction evidence from maintaining a coherent current state, and identify whole-state coherence as a central requirement for persistent world modeling beyond accurate per-query state recovery.
- `StateCommit` ***Framework.*** We introduce `StateCommit`, which maintains a query-independent *Active State* through verified interaction-time updates and uses slot-indexed *Visual Bindings* to ground the maintained semantic state in the prediction view induced by the known next action. This design separates persistent state maintenance from on-demand reconstruction while connecting semantic state to future rendering.
- `StateReturn` ***Benchmark.*** We introduce `StateReturn`, a multimodal benchmark for evaluating whole-state maintenance and state-consistent future prediction under controlled partial observability. On `StateReturn`, Full-History retains 98.61% target-state accuracy after 16 intervening interactions but achieves only 13.89% full-state exact match, whereas `StateCommit` achieves 98.67–100.00% full-state exact match and reduces state-sensitive prediction error by 58.1% relative to the best memory baseline.

2 Related Work

Memory under Partial Observability. Under partial observability, agents rely on information from earlier interactions when the current observation is insufficient. Existing approaches preserve and reuse past experience through recurrent representations, extended context, external read–write

memory, retrieval, summarization, reflection, and adaptive memory management (Hausknecht & Stone, 2015; Dai et al., 2019; Park et al., 2023; Packer et al., 2023; Wang et al., 2023; Xu et al., 2026). Across these works, the emphasis is primarily on how past experience is stored, organized, retrieved, or reused. `StateCommit` instead focuses on how interaction evidence can be incrementally resolved into a query-independent representation of what is currently true as interactions unfold.

World Models and State Representations. Existing approaches represent partially observed environments through probabilistic belief states, learned latent dynamics, or structured and object-centric representations (Kaelbling et al., 1998; Hafner et al., 2019b; Bruce et al., 2024; Locatello et al., 2020; Wu et al., 2022; Hafner et al., 2019a). These approaches primarily focus on latent state estimation, predictive dynamics, or structured scene representations, rather than explicitly maintaining the latest interaction-established values of tracked variables as interactions unfold. `StateCommit` targets this complementary problem of explicit current-state maintenance.

Evaluation of Persistent World State. Existing embodied and state-tracking benchmarks evaluate memory and environmental state through task success or individually queried facts (Das et al., 2018; Pasukonis et al., 2022; Majumdar et al., 2024). Such metrics can verify a requested state without testing whether the tracked state remains jointly consistent as a whole. Visual prediction benchmarks commonly emphasize full-frame fidelity (Oh et al., 2015; Finn et al., 2016), which may underweight localized but state-critical differences. `StateReturn` targets these gaps by evaluating both whole-state coherence and whether persistent state is correctly reflected in the predicted future.

3 `StateCommit`: Active World-State Maintenance

3.1 Problem Formulation

Let W_t denote the underlying world state at time t. The agent receives a partial observation o_t and takes an action a_t, with

$$o_t = \Omega(W_t), \qquad W_{t+1} = \mathcal{F}(W_t, a_t), \tag{1}$$

where Ω and $\mathcal{F}$ denote the observation and transition processes, respectively (Kaelbling et al., 1998). The interaction history available at time t is $H_t = (o_0, a_0, \ldots, a_{t-1}, o_t)$. Rather than estimating the complete latent world state W_t, we track interaction-established entity-level variables that may remain relevant after they are no longer directly observable. A tracked slot is denoted by $q = (e, c)$, where e is a persistent entity identifier and c is a typed state component, such as *openness*, *place*, or *toggle state*, with value domain $\mathcal{V}_c$. Let Q_t denote the set of slots tracked by time t, and let $z_t(q) \in \mathcal{V}_c$ denote the current value of slot q. The current tracked state is

$$Z_t = \{q \mapsto z_t(q) \mid q \in Q_t\}. \tag{2}$$

A read-time formulation reconstructs a requested state value on demand from memory M_t, which may be the full interaction history H_t or a representation derived from it, as $\hat{z}_t(q) = R(M_t, q)$. Because this readout is conditioned on q, it can recover individual slots without maintaining a joint estimate of the tracked state. By contrast, interaction-time state maintenance updates the tracked state whenever new interaction evidence x_t arrives, independently of any future query:

$$\hat{Z}_t = U_Z(\hat{Z}_{t-1}, x_t), \qquad \hat{z}_t(q) = \hat{Z}_t[q]. \tag{3}$$

We refer to $\hat{Z}_t$ as the *Active State*. State queries are then answered by direct readout from $\hat{Z}_t$, without conditioning the update on a particular query. Whole-state correctness requires

$$\hat{Z}_t[q] = z_t(q), \qquad \forall q \in Q_t. \tag{4}$$

The objective of our work is therefore to maintain an Active State whose tracked variables remain simultaneously correct as interactions unfold.

Overview. `StateCommit` maintains persistent world state through two complementary components. Transactional active-state maintenance converts interaction evidence into explicit state transactions and commits only runtime-verified updates to the *Active State*. The resulting Active State supports direct state readout, but visual prediction additionally requires the maintained semantics to be grounded in the prediction view. To this end, the state-to-render interface uses slot-indexed *Visual Bindings* to ground the maintained semantic state in the prediction view induced by the known action a_t. Together, these components connect interaction evidence to a maintained current state and, when required, to state-consistent future observations.

3.2 TRANSACTIONAL ACTIVE-STATE MAINTENANCE

Maintaining a persistent Active State requires separating semantic interpretation from persistent-state mutation. Given new interaction evidence x_t, `StateCommit` first converts the state change supported by the evidence into an explicit transaction, which is then verified and committed by a deterministic runtime. This separation allows model-based reasoning to determine what should change without directly modifying persistent state.

Evidence-Grounded Transaction Construction. This module converts interaction evidence into an explicit, typed description of the state change it supports. Given x_t, the Grounder extracts typed *EventFacts* E_t that capture the state-relevant consequences of the interaction. Conditioned on E_t and the current Active State $\hat{Z}_{t-1}$, the Programmer constructs a candidate update over the affected tracked slots. When necessary, the Reviewer repairs a first-pass candidate, while the Auditor checks that the resulting proposal remains consistent with the grounded interaction evidence.

Rather than regenerating the entire Active State, `StateCommit` represents the candidate update as a sparse state transaction (Liang et al., 2023; Singh et al., 2022)

$$P_t = (b_t, \Delta_t), \qquad \Delta_t = \left\{(q_i, v_i^-, v_i^+)\right\}_{i=1}^{N_t}, \tag{5}$$

where N_t is the number of proposed slot updates and b_t denotes the Active State version attached to the transaction. Each tuple (q_i, v_i^-, v_i^+) specifies a local transition for slot q_i, from its expected current value v_i^- to the proposed value v_i^+. Including both the expected old value and the proposed new value makes each candidate update an explicit state transition rather than an unconstrained rewrite of persistent state. The transaction therefore exposes the information needed for the runtime to verify the proposal against the current Active State before any persistent state is modified.

Runtime-Verified State Commit. An explicit transaction specifies what state change should occur, but does not by itself authorize modification of persistent state. `StateCommit` therefore delegates the final commit to a deterministic runtime, which verifies the proposed transition against the current Active State. For a slot $q_i = (e_i, c_i)$, a transaction P_t is valid only if

$$b_t = \mathrm{ver}(\hat{Z}_{t-1}), \qquad \hat{Z}_{t-1}[q_i] = v_i^-, \qquad v_i^+ \in \mathcal{V}_{c_i}, \qquad \forall i. \tag{6}$$

These checks ensure that the proposal corresponds to the Active State version, matches the expected pre-update value of each affected slot, and respects the typed value domain of each state component.

Only when all proposed slot transitions satisfy these checks does the runtime commit the transaction:

$$\hat{Z}_t = \begin{cases} \mathrm{Apply}(\hat{Z}_{t-1}, \Delta_t), & \text{if } P_t \text{ is valid,} \\ \hat{Z}_{t-1}, & \text{otherwise.} \end{cases} \tag{7}$$

The commit is atomic: either all transitions in Δ_t are applied together, or the Active State remains unchanged (Gray & Reuter, 1992), preventing a partially valid proposal from leaving the maintained state in an intermediate configuration. The committed Active State $\hat{Z}_t$ then provides the semantic state used by the subsequent state-to-render interface.

3.3 STATE-TO-RENDER INTERFACE

For visual prediction, the maintained semantic state alone does not specify how each tracked value should appear in the next view. `StateCommit` therefore associates visually grounded slots with *Visual Bindings*, aligns these bindings to the prediction view induced by the known next action a_t, and conditions the renderer on both the Active State and the aligned visual evidence.

Prediction-View Binding Construction. For each visually grounded slot q, let τ_q denote the time of the most recent committed update that establishes the current value of q. The corresponding slot-indexed *Visual Binding* $B_{\tau_q}(q)$ references the complete public RGB frame observed immediately after that interaction, together with its public frame index and an availability indicator. When q is updated again, its binding is refreshed together with the Active State so that the stored visual evidence remains synchronized with the currently committed value.

To use this evidence for predicting a later observation, `StateCommit` aligns the stored binding to the prediction view induced by the known action:

$$\widetilde{B}_{t+1}(q) = \mathcal{A}\left(B_{\tau_q}(q); o_{\tau_q:t}, a_{\tau_q:t}\right), \tag{8}$$

where $\mathcal{A}$ maps the binding from its source view to the coordinate frame of the prediction view. The view transformation is estimated from the observed RGB sequence and public actions along the intervening trajectory, with the known action a_t included in the final projection. The aligned binding $\widetilde{B}_{t+1}(q)$ therefore provides view-specific visual grounding for the current semantic value of slot q.

State-Conditioned Future Rendering. With the semantic state and its prediction-view grounding in place, the renderer predicts the next observation as

$$\hat{o}_{t+1} = G_\theta \left(o_t,\, a_t,\, \hat{Z}_t,\, \widetilde{\mathcal{B}}_{t+1} \right), \tag{9}$$

where $\widetilde{\mathcal{B}}_{t+1} = \{\widetilde{B}_{t+1}(q)\}$ collects the aligned bindings for visually grounded slots. The Active State provides the current semantic values, while the aligned bindings provide the view-specific grounding needed to express them in the predicted observation. Together, they enable the renderer to produce a future observation consistent with the maintained state.

4 `StateReturn`: Evaluating Persistent World State

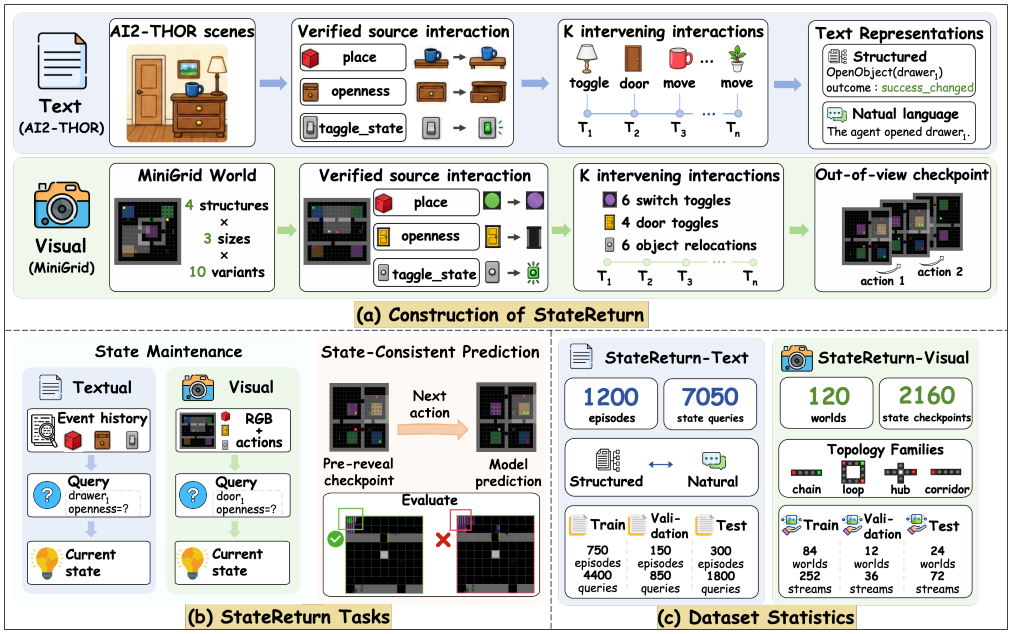


Figure 2: **Overview of `StateReturn`.** (a) Controlled persistence protocol. (b) Evaluation tasks. (c) Dataset statistics for the text and visual tracks.

`StateReturn` operationalizes the two evaluation requirements introduced above: whether interaction-established state remains coherent after leaving direct observation, and whether that state is correctly reflected in a predicted future. It therefore evaluates whole-state maintenance and state-consistent prediction under controlled partial observability.

Controlled Persistence Protocol. All tasks follow a shared trajectory protocol that explicitly separates state establishment from later evaluation. A verified source interaction first establishes the value of a designated tracked variable, referred to as the *source state*. We then introduce $K \in \{0, 1, 2, 4, 8, 16\}$ intervening interactions before evaluation, during which the source state is not directly observed again. Varying K controls the distance between state establishment and evaluation, allowing us to test whether interaction-established state remains available after leaving direct observation and through subsequent interactions. Across the text and visual tracks, we evaluate three shared state components: `place`, `openness`, and `toggle_state`.

Evaluation Tasks. `StateReturn` evaluates two complementary capabilities: *State Maintenance* and *State-Consistent Prediction*. State Maintenance tests whether a model can produce a coherent estimate of the current tracked state at the evaluation checkpoint. We instantiate this capability

Table 1: Textual state maintenance on `StateReturn`-Text. We report FSEM (%) for $K \in \{0, 1, 2, 4, 8, 16\}$ intervening interactions. Best and second-best capacity-matched results are shown in **bold** and underlined; †GPT-6 Astra is reported separately as a strong Full-History control.

Method	Intervening interactions K					
	0	**1**	**2**	**4**	**8**	**16**
Structured representation						
Full-History	**100.00**	95.00	92.67	86.00	73.00	53.00
Bounded-History	**100.00**	95.00	92.67	3.67	0.33	0.33
Rolling Summary	98.00	89.67	85.67	77.33	59.00	57.67
Recurrent Memory	89.00	63.67	55.33	42.00	7.33	0.33
STATECOMMIT	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**
GPT-6 Astra†	100.00	100.00	99.33	100.00	99.33	99.33
Natural-language representation						
Full-History	88.00	81.33	54.67	27.33	7.67	3.00
Bounded-History	88.00	80.00	58.67	2.33	0.00	0.00
Rolling Summary	57.67	12.00	10.00	6.33	3.33	2.67
Recurrent Memory	91.33	37.33	18.00	17.00	4.00	0.33
STATECOMMIT	**98.67**	**98.67**	**98.67**	**98.67**	**98.67**	**98.67**
GPT-6 Astra†	99.67	99.00	99.00	99.33	99.00	99.00

in two modalities: *Textual State Maintenance*, which operates on textual interaction streams, and *Visual State Maintenance*, which infers the same tracked variables from RGB observations and actions. *State-Consistent Prediction* further tests whether the predicted next observation reflects the persistent state established earlier in the trajectory.

`StateReturn`-Text. The text track contains 1,200 AI2-THOR-derived interaction episodes and 7,050 state queries. Each episode is provided in two aligned textual representations: *Structured* encodes each interaction using explicit fields for the action type, entity arguments, and execution outcome, while *Natural* describes the same interaction sequence in natural language. Both representations correspond to the same underlying state transitions and share the same evaluation queries.

`StateReturn`-Visual. The visual track contains 120 MiniGrid worlds and 360 source-state streams, yielding 2,160 evaluation checkpoints across the six values of K. Each stream begins with a visible source entity and a verified interaction that establishes one of its tracked state values. The agent then moves until the source leaves view and continues interacting with other entities without directly re-observing the source state. For *State-Consistent Prediction*, each factual future is paired with a counterfactual future that has the same current observation and next action but differs in the persistent source state established earlier. By holding the immediate prediction context fixed, this pairing isolates the effect of persistent state on the predicted future.

Evaluation Metrics. For state maintenance, *Target State Exact Match* (TSEM) measures whether the queried target slot is correct, *Tracked Slot Accuracy* (TSA) measures average correctness across all tracked slots, and *Full-State Exact Match* (FSEM) requires the entire tracked state to be correct simultaneously. TSEM therefore captures target-level correctness, while TSA and especially FSEM assess the coherence of the tracked state beyond the queried slot. For *State-Consistent Prediction*, evaluation focuses on pixels that differ between the paired factual and counterfactual futures. *SourceMAE* measures reconstruction error against the factual future over these state-sensitive pixels, while *Counterfactual Preference* (CFP) measures whether the prediction is closer to the factual future than to its matched counterfactual.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Tasks and Evaluation. Following the task definitions and metrics in Sec. 4, we evaluate `StateCommit` on *State Maintenance* and *State-Consistent Prediction*. State Maintenance is evaluated on Structured and Natural text as well as RGB observations, while State-Consistent Prediction is evaluated on the visual track. All settings use $K \in \{0, 1, 2, 4, 8, 16\}$ intervening interactions.

Table 2: Visual state maintenance on `StateReturn`-Visual. We report TSEM and FSEM over $K \in \{0, 1, 2, 4, 8, 16\}$ intervening interactions. Higher is better. Best and second-best results are shown in **bold** and underlined, respectively.

Method	Overall	Intervening interactions K					
		0	1	2	4	8	16
Target State Exact Match (TSEM ↑)							
Bounded-History	41.90	52.78	44.44	33.33	47.22	36.11	37.50
Full-History	99.54	**100.00**	**100.00**	98.61	**100.00**	**100.00**	98.61
Recurrent Memory	81.94	**100.00**	95.83	86.11	79.17	75.00	55.56
StateCommit	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**
Full-State Exact Match (FSEM ↑)							
Bounded-History	15.28	52.78	30.56	4.17	4.17	0.00	0.00
Full-History	69.44	**100.00**	97.22	94.44	80.56	30.56	13.89
Recurrent Memory	64.12	**100.00**	95.83	83.33	58.33	40.28	6.94
StateCommit	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**

Baselines. We compare `StateCommit` with representative baselines appropriate to each modality. Context-based baselines include *Full-History*, which retains the complete interaction prefix and reconstructs the current state at evaluation time, and *Bounded-History*, which retains only the four most recent interactions. In the textual setting, *Rolling Summary* incrementally compresses past interactions into a fixed 1,600-character summary. *Recurrent Memory* instead incorporates interaction evidence online into a GRU hidden state (Cho et al., 2014). We additionally evaluate Full-History with *GPT-6 Astra*[1] on the textual track as a strong-model control for read-time state reconstruction.

Implementation Details. For *Textual State Maintenance*, all LLM-based methods use Qwen2.5-Coder-7B-Instruct (Hui et al., 2024) with greedy decoding. For *Visual State Maintenance*, the memory baselines share the ResNet-18 backbone (He et al., 2016) used by the `StateCommit` Grounder. For *State-Consistent Prediction*, the history-based baselines share the same U-Net renderer (Ronneberger et al., 2015). For `StateCommit`, we freeze the pretrained Bounded-History renderer as the base predictor and implement the state-to-render interface as a residual correction conditioned on the maintained Active State and Visual Bindings. Full details are provided in Appendix C.

5.2 Persistent State Maintenance

Textual State Maintenance. Table 1 reports FSEM as the number of intervening interactions increases. We observe that: ***Obs.*** ❶ `StateCommit` ***maintains whole-state coherence as intervening interactions accumulate.*** Under the Qwen2.5 setting, `StateCommit` achieves 100.00% FSEM on Structured inputs and 98.67% on Natural inputs for all values of K, whereas the best memory baseline at $K = 16$ reaches only 57.67% and 3.00%, respectively. ***Obs.*** ❷ ***Complete history does not guarantee reliable read-time state reconstruction.*** Full-History retains the complete interaction prefix but still degrades as K increases. In contrast, the GPT-6 Astra control remains above 99% FSEM, showing that reliable read-time reconstruction can depend strongly on the model.

Visual State Maintenance. We next evaluate the same distinction from RGB observations and actions. Table 2 yields the following observation: ***Obs.*** ❸ ***High target-state accuracy can mask severe whole-state inconsistency.*** `StateCommit` achieves 100.00% TSEM and FSEM across all values of K. In contrast, at $K = 16$, Full-History still recovers the designated target state with 98.61% TSEM, while its FSEM falls to only 13.89%. Thus, correctly recovering the requested state does not imply that the tracked state is coherent as a whole.

5.3 State-Consistent Prediction

Having established that `StateCommit` maintains a coherent current state, we next ask whether this state is correctly reflected in the predicted future. Table 3 yields the following observation: ***Obs.*** ❹ `StateCommit` ***translates coherent state maintenance into state-consistent future prediction.*** Overall, `StateCommit` reduces SourceMAE from the best baseline value of 0.2843 to 0.1190, a relative reduction of 58.1%, while improving CFP from the best baseline value of 0.3773 to 0.8333. The improvement persists across all values of K, while additional reconstruction metrics in Appendix D.3

[1] https://developers.openai.com/api/docs/models/gpt-6-astra

Table 3: State-consistent prediction on `StateReturn`-Visual. We report SourceMAE and CFP overall and across $K \in \{0, 1, 2, 4, 8, 16\}$ intervening interactions. Lower SourceMAE and higher CFP are better. Best and second-best results are shown in **bold** and underlined, respectively.

Method	Overall	Intervening interactions K					
		0	1	2	4	8	16
SourceMAE ↓							
Bounded-History	0.2917	0.3107	0.2970	0.2849	0.2843	0.2873	0.2862
Full-History	0.2901	0.2960	0.2983	0.2827	0.2973	0.2926	0.2736
Recurrent Memory	0.2843	0.3068	0.2915	0.2617	0.2891	0.2881	0.2684
StateCommit	**0.1190**	**0.0832**	**0.1274**	**0.1223**	**0.1220**	**0.1248**	**0.1341**
Counterfactual Preference (CFP ↑)							
Bounded-History	0.3704	0.3056	0.3472	0.3889	0.3889	0.3750	0.4167
Full-History	0.3773	0.3333	0.3333	0.4306	0.3750	0.3611	0.4306
Recurrent Memory	0.3681	0.3333	0.3333	0.4028	0.3750	0.3472	0.4167
StateCommit	**0.8333**	**0.9028**	**0.8611**	**0.8472**	**0.8194**	**0.8056**	**0.7639**

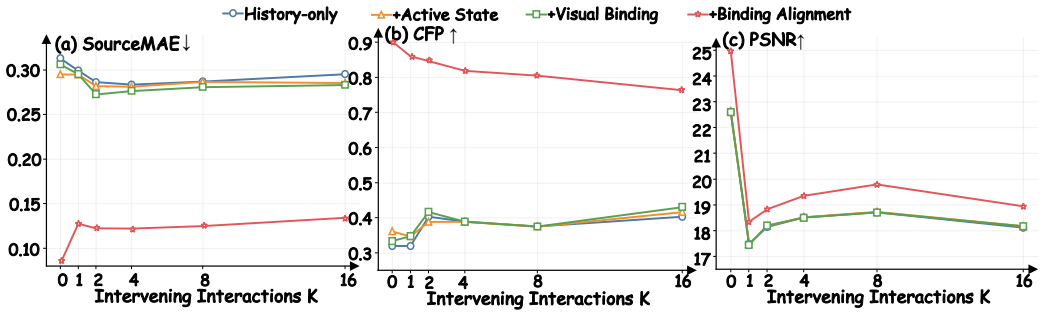


Figure 3: Ablation of the state-to-render interface. We progressively add the Active State, Visual Binding, and Binding Alignment to a history-only control.

show that favorable full-frame PSNR (Huynh-Thu & Ghanbari, 2008) and SSIM (Wang et al., 2004) do not necessarily correspond to state-consistent prediction.

5.4 Mechanism Analysis

Transactional State Maintenance. Table 4 traces transaction construction and validation before commit, yielding the following observation: ***Obs.* ❺ *Selective review repairs first-pass transaction failures before commit.*** For Structured text and visual inputs, all candidate transactions pass Runtime preflight on the first attempt. On Natural inputs, 82.64% are valid on the first pass, while the Reviewer is invoked only for the remaining 718 proposals and repairs all of them into Runtime-valid transactions.

Table 4: State transaction construction.

Setting	1st-Pass Valid (%)	Reviewer Calls(#)	Repair (%)	FSEM
Text-Structured	100.00	0	–	**100.00**
Text-Natural	82.64	718	**100.00**	98.67
Visual State	100.00	0	–	**100.00**

State-to-Render Interface. To isolate the state-to-render interface, all variants share the same frozen Bounded-History renderer and residual architecture while progressively adding the Active State, Visual Bindings, and Binding Alignment. From Figure 3, we observe that: ***Obs.* ❻ *Prediction-view alignment is critical for state-consistent rendering.*** Providing the Active State alone or together with an unaligned Visual Binding produces only modest changes, whereas Binding Alignment brings the largest improvement in both SourceMAE and CFP across values of K.

5.5 Robustness and Further Analysis

Superseding Updates. Increasing K tests intervening interactions but not repeated changes to the same state variable. We therefore construct sequential relocation trajectories and compare one versus three prior updates of the same `place` slot in Table 5. We observe that: ***Obs.* ❼ `StateCommit` *preserves the latest operative state under repeated supersession.*** `StateCommit` maintains 96.00% AnswerEM on Natural inputs and 100.00% on Structured inputs as the supersession depth increases. In contrast, Recurrent Memory drops by 46.00 points on Natural inputs.

Table 5: Superseding updates under sequential relocation. We report AnswerEM (%) after one versus three updates of the same place slot, averaged over K, where D_{sup} denotes supersession depth. Δ_{drop} denotes the performance drop from $D_{\text{sup}} = 1$ to $D_{\text{sup}} = 3$.

Method	Natural			Structured		
	$D_{sup} = 1$	$D_{sup} = 3$	$\Delta \downarrow$	$D_{sup} = 1$	$D_{sup} = 3$	$\Delta \downarrow$
Full-History	**98.67**	91.67	7.00	**100.00**	95.67	4.33
Rolling Summary	48.00	38.67	9.33	97.00	94.67	2.33
Recurrent Memory	98.00	52.00	46.00	98.00	53.67	44.33
StateCommit	96.00	**96.00**	**0.00**	**100.00**	**100.00**	**0.00**

Across State Types and World Structures. Table 6 evaluates three state types and four visual world families. We make the following observation: ***Obs.*** ❽ `StateCommit` ***remains consistent across state types and world structures.*** `StateCommit` achieves 100.00% FSEM in all settings. Its CFP ranges from 0.7431 to 0.8958 across state types and from 0.7500 to 0.9074 across world families, exceeding the strongest history baseline throughout. Complete results are provided in Appendix D.

Table 6: Robustness across state types and world structures. We compare `StateCommit` with the best baseline.

Setting	FSEM (%) ↑		CFP ↑	
	Ours	Hist.	Ours	Hist.
State type				
Openness	**100.00**	70.83	**0.8611**	0.5208
Place	**100.00**	67.36	**0.8958**	0.1181
Toggle state	**100.00**	70.14	**0.7431**	0.5000
World family				
Chain	**100.00**	73.15	**0.8426**	0.3148
Four-Room	**100.00**	64.81	**0.8333**	0.5556
Central Hub	**100.00**	66.67	**0.7500**	0.3611
Corridor	**100.00**	73.15	**0.9074**	0.3056

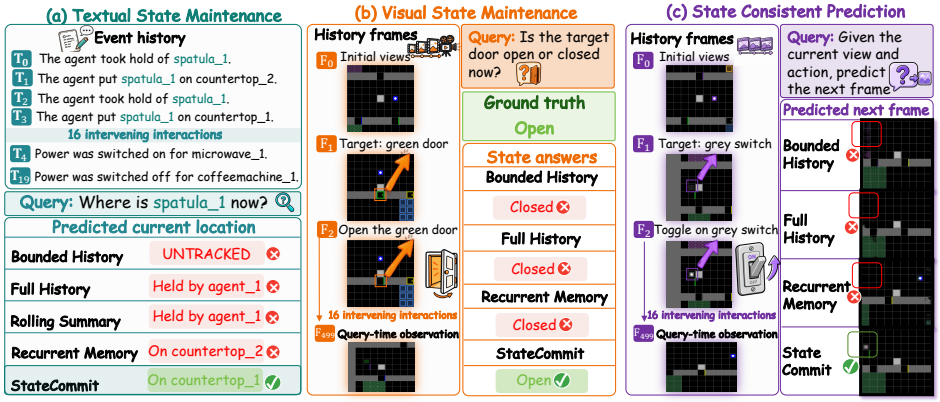


Figure 4: Qualitative examples of persistent state maintenance and prediction.

Qualitative Examples. Figure 4 complements the aggregate results with cases from all three tasks. We observe that: ***Obs.*** ❾ `StateCommit` ***prevents stale readout and state-inconsistent prediction.*** In Textual State Maintenance, `StateCommit` preserves the latest location after repeated relocations. In Visual State Maintenance, it retains the correct open state after 16 interactions out of view. In State-Consistent Prediction, it reflects the earlier switch state in the predicted frame, whereas history-based renderers miss the state-dependent outcome. Additional examples are provided in Appendix D.4.

6 Conclusion

We study persistent world modeling under partial observability and distinguish retaining interaction history from maintaining an explicit representation of the current state. To this end, `StateCommit` maintains a query-independent Active State as interactions unfold and connects this persistent state to future prediction through Visual Bindings. We further introduce `StateReturn` to evaluate whole-state correctness and state-consistent prediction under controlled intervening interactions. Across textual and visual settings, our results show that accurate recovery of individual states does not guarantee a jointly correct current-state estimate, while explicit state maintenance improves state-consistent future prediction. These findings support interaction-time state maintenance as a useful design principle for persistent interactive world models.

AI USE STATEMENT

Generative AI tools were used only to assist with language polishing, grammatical refinement, and the retrieval and organization of publicly available information and references. All technical content, methodology, experiments, analyses, and conclusions were developed and verified by the authors.

ETHICS STATEMENT

This work is conducted entirely in simulated interactive environments using synthetic or simulator-generated trajectories and observations. It involves no human subjects, personally identifiable information, private data, or real-world autonomous deployment. While the experiments reported here therefore raise limited direct privacy or safety concerns, real-world deployment of interactive world models would require additional consideration of reliability, safety, and responsible use.

REPRODUCIBILITY STATEMENT

To support reproducibility, we publicly release the **`StateReturn`** dataset at https://huggingface.co/datasets/StateCommit/StateReturn. We have also released an initial portion of the implementation at https://github.com/StateCommit/StateCommit_codes, including the core components currently prepared for public release. The remaining implementation components will be progressively organized and released to further improve the completeness of the codebase.

Additional resources and a visual overview are available on the project page at https://statecommit.github.io/. Detailed benchmark construction, data organization, and evaluation protocols are provided in Appendix B, while model architectures, implementation details, training procedures, and optimization settings are documented in Appendix C. Together, these resources provide the benchmark, implementation details, and experimental protocols needed to facilitate reproduction and further study of our method.

REFERENCES

Eloi Alonso, Adam Jelley, Vincent Micheli, Anssi Kanervisto, Amos Storkey, Tim Pearce, and François Fleuret. Diffusion for world modeling: Visual details matter in atari. *Advances in Neural Information Processing Systems*, 37:58757–58791, 2024.

Daniel M Bear, Elias Wang, Damian Mrowca, Felix J Binder, Hsiao-Yu Fish Tung, RT Pramod, Cameron Holdaway, Sirui Tao, Kevin Smith, Fan-Yun Sun, et al. Physion: Evaluating physical prediction from vision in humans and machines. *arXiv preprint arXiv:2106.08261*, 2021.

Jake Bruce, Michael D Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, et al. Genie: Generative interactive environments. In *Forty-first international conference on machine learning*, 2024.

Taiye Chen, Xun Hu, Zihan Ding, and Chi Jin. Learning world models for interactive video generation. *Advances in Neural Information Processing Systems*, 38:154456–154483, 2026.

Kyunghyun Cho, Bart Van Merriënboer, Çağlar Gulçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1724–1734, 2014.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 2978–2988, 2019.

Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Embodied question answering. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 2135–213509. IEEE, 2018.

Kuan Fang, Alexander Toshev, Li Fei-Fei, and Silvio Savarese. Scene memory transformer for embodied agents in long-horizon tasks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 538–547. IEEE, 2019.

Stefano Ferraro, Pietro Mazzaglia, Tim Verbelen, and Bart Dhoedt. Focus: object-centric world models for robotic manipulation. *Frontiers in Neurorobotics*, 19:1585386, 2025.

Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. *Advances in neural information processing systems*, 29, 2016.

Kristen Grauman, Andrew Westbury, Eugene Byrne, Zachary Chavis, Antonino Furnari, Rohit Girdhar, Jackson Hamburger, Hao Jiang, Miao Liu, Xingyu Liu, et al. Ego4d: Around the world in 3,000 hours of egocentric video. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 18995–19012, 2022.

Jim Gray and Andreas Reuter. *Transaction processing: concepts and techniques*. Elsevier, 1992.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019a.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pp. 2555–2565. PMLR, 2019b.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640(8059):647–653, 2025.

Nick Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. In *International Conference on Learning Representations*, volume 2024, pp. 47376–47405, 2024.

Matthew J Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *AAAI fall symposia*, volume 45, pp. 141, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 32779–32798, 2025.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Quan Huynh-Thu and Mohammed Ghanbari. Scope of validity of psnr in image/video quality assessment. *Electronics letters*, 44(13):1, 2008.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

Thomas Kipf, Elise Van der Pol, and Max Welling. Contrastive learning of structured world models. *arXiv preprint arXiv:1911.12247*, 2019.

Thomas Kipf, Gamaleldin F Elsayed, Aravindh Mahendran, Austin Stone, Sara Sabour, Georg Heigold, Rico Jonschkowski, Alexey Dosovitskiy, and Klaus Greff. Conditional object-centric learning from video. *arXiv preprint arXiv:2111.12594*, 2021.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International conference on robotics and automation (ICRA)*, pp. 9493–9500. IEEE, 2023.

Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. *Advances in neural information processing systems*, 33:11525–11538, 2020.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13851–13870, 2024.

Arjun Majumdar, Anurag Ajay, Xiaohan Zhang, Pranav Putta, Sriram Yenamandra, Mikael Henaff, Sneha Silwal, Paul Mcvay, Oleksandr Maksymets, Sergio Arnaud, et al. Openeqa: Embodied question answering in the era of foundation models. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16488–16498. IEEE, 2024.

Vincent Micheli, Eloi Alonso, and François Fleuret. Transformers are sample-efficient world models. *arXiv preprint arXiv:2209.00588*, 2022.

Ali Modarressi, Ayyoob Imani, Mohsen Fayyaz, and Hinrich Schütze. Ret-llm: Towards a general read-write memory for large language models. *arXiv preprint arXiv:2305.14322*, 2023.

Steven Morad, Ryan Kortvelesy, Matteo Bettini, Stephan Liwicki, and Amanda Prorok. Popgym: Benchmarking partially observable reinforcement learning. *arXiv preprint arXiv:2303.01859*, 2023.

Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. *Advances in neural information processing systems*, 28, 2015.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Aishwarya Padmakumar, Jesse Thomason, Ayush Shrivastava, Patrick Lange, Anjali Narayan-Chen, Spandana Gella, Robinson Piramuthu, Gokhan Tur, and Dilek Hakkani-Tur. Teach: Task-driven embodied agents that chat. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 2017–2025, 2022.

Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. *arXiv preprint arXiv:1702.08360*, 2017.

Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. Stabilizing transformers for reinforcement learning. In *International conference on machine learning*, pp. 7487–7498. PMLR, 2020.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023.

Jurgis Pasukonis, Timothy Lillicrap, and Danijar Hafner. Evaluating long-term memory in 3d mazes. *arXiv preprint arXiv:2210.13383*, 2022.

Marco Pleines, Matthias Pallasch, Frank Zimmer, and Mike Preuss. Memory gym: Partially observable challenges to memory-based agents. In *The eleventh international conference on learning representations*, 2023.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *2020 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pp. 10737–10746. IEEE, 2020.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*, 2022.

Theodore R Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L Griffiths. Cognitive architectures for language agents. *arXiv preprint arXiv:2309.02427*, 2023.

Haoran Tan, Zeyu Zhang, Chen Ma, Xu Chen, Quanyu Dai, and Zhenhua Dong. Membench: Towards more comprehensive evaluation on the memory of llm-based agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 19336–19352, 2025.

Rishi Veerapaneni, John D Co-Reyes, Michael Chang, Michael Janner, Chelsea Finn, Jiajun Wu, Joshua Tenenbaum, and Sergey Levine. Entity abstraction in visual model-based reinforcement learning. In *Conference on Robot Learning*, pp. 1439–1456. PMLR, 2020.

Angel Villar-Corrales, Ismail Wahdan, and Sven Behnke. Object-centric video prediction via decoupling of object dynamics and interactions. In *2023 IEEE International conference on image processing (ICIP)*, pp. 570–574. IEEE, 2023.

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.

Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

Ziyi Wu, Nikita Dvornik, Klaus Greff, Thomas Kipf, and Animesh Garg. Slotformer: Unsupervised visual dynamics simulation with object-centric models. *arXiv preprint arXiv:2210.05861*, 2022.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38:17577–17604, 2026.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z Pan, et al. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12805–12825, 2026.

Kexin Yi, Chuang Gan, Yunzhu Li, Pushmeet Kohli, Jiajun Wu, Antonio Torralba, and Joshua B Tenenbaum. Clevrer: Collision events for video representation and reasoning. *arXiv preprint arXiv:1910.01442*, 2019.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

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.

APPENDIX CONTENTS

A Extended Related Work **15**

A.1 Memory under Partial Observability 15

A.2 World Models and State Representations 15

A.3 Evaluation of Persistent World State 16

B `StateReturn` Benchmark Details **16**

B.1 Benchmark Construction Protocol 16

B.2 `StateReturn`-Text 17

B.3 `StateReturn`-Visual 17

B.4 Evaluation Metrics 27

C Implementation and Training Details **28**

C.1 Textual State Maintenance 28

C.2 Visual State Maintenance 29

C.3 State-Consistent Prediction 30

C.4 Optimization and Compute 30

D Additional Experimental Results **31**

D.1 Textual State Maintenance 31

D.2 Visual State Maintenance 33

D.3 State-Consistent Prediction 34

D.4 Additional Qualitative Examples 35

E Additional Analysis **35**

E.1 Perception-Boundary Analysis 35

E.2 Binding Alignment Diagnostic 36

A Extended Related Work

A.1 Memory under Partial Observability

Partial observability has long motivated mechanisms that preserve information beyond the current observation (Parisotto & Salakhutdinov, 2017; Fang et al., 2019). Recurrent agents summarize past observations into hidden states, allowing previously observed information to influence later decisions when it is no longer directly available (Hausknecht & Stone, 2015). Long-context architectures provide another route by extending the range over which earlier information can be directly reused (Dai et al., 2019; Parisotto et al., 2020). Although these approaches differ in representation, both aim to retain information that may become relevant again later.

Recent language-model agents have developed more explicit mechanisms for long-term memory. Generative Agents organizes observations, reflections, and higher-level summaries for later retrieval (Park et al., 2023), while MemGPT manages information across limited working context and external memory through explicit operations (Packer et al., 2023). LongMem augments language models with retrievable long-term memory over prior context (Wang et al., 2023), while more recent agentic memory systems adaptively organize and update stored experience as interaction proceeds (Xu et al., 2026). Other work further explores explicit read–write memory, memory consolidation, reflection over past experience. (Zhong et al., 2024; Modarressi et al., 2023; Shinn et al., 2023; Zhao et al., 2024; Sumers et al., 2023; Yan et al., 2026).

These methods substantially improve how past experience is preserved, compressed, organized, retrieved, and reused. Our focus is complementary: even when the relevant interaction evidence remains available, the model must still determine which past state changes remain operative after subsequent interactions and reconcile them into a coherent representation of what is currently true. From this perspective, memory provides evidence for current-state reasoning, whereas `StateCommit` focuses on maintaining the resolved current state itself as interactions unfold.

A.2 World Models and State Representations

Representing hidden environment state is a central problem under partial observability. Classical formulations address this through belief states, which summarize the interaction history as a distribution over possible latent world states (Kaelbling et al., 1998). This provides a principled mechanism for reasoning under uncertainty when the underlying state cannot be observed directly.

Learned world models instead acquire predictive state representations from experience. Early action-conditioned video models predict how visual observations evolve under actions (Oh et al., 2015; Finn et al., 2016), while latent-dynamics models learn compact internal states together with transition dynamics for prediction, planning, and control (Hafner et al., 2019b;a; 2025). More recent work extends this paradigm with transformer-based and scalable latent world models (Micheli et al., 2022; Hansen et al., 2024), as well as generative models of richer interactive environments (Bruce et al., 2024).

A complementary line of research introduces more structured representations of the environment. Object-centric methods decompose scenes into entity-level representations, supporting object discovery, tracking, and modeling of object dynamics and interactions (Locatello et al., 2020; Wu et al., 2022; Villar-Corrales et al., 2023; Ferraro et al., 2025; Kipf et al., 2019; Veerapaneni et al., 2020). Such representations provide a more compositional alternative to a single global latent state and can better preserve entity-specific information across time.

These lines of work primarily study how hidden environment state should be represented and propagated for prediction or control. Our setting focuses on a complementary problem: when interactions establish or modify explicit tracked variables, the model must keep their latest valid values current as subsequent interactions occur and the corresponding entities leave direct observation. `StateCommit` therefore focuses on explicit current-state maintenance rather than on learning a complete latent representation of the world.

A.3 Evaluation of Persistent World State

Persistent state is evaluated indirectly in many embodied and interactive benchmarks. Embodied Question Answering requires an agent to explore an environment and answer questions about it (Das et al., 2018), while more recent benchmarks such as OpenEQA extend embodied question answering to richer environments and foundation-model settings (Majumdar et al., 2024). Long-horizon embodied tasks additionally require agents to retain information across extended interaction sequences. For example, ALFRED includes compositional household tasks with persistent, non-reversible state changes, while TEACh evaluates language-guided interaction and task execution in household environments (Shridhar et al., 2020; Padmakumar et al., 2022). Memory-focused benchmarks more directly isolate long-term information retention, for example by testing navigation under partial observability after relevant information has left the current view (Pasukonis et al., 2022; Morad et al., 2023; Pleines et al., 2023).

These benchmarks capture important forms of embodied memory and state-dependent behavior, but their primary metrics typically concern task completion, navigation performance, or correctness on individual questions. Such evaluation can reveal whether information needed for a particular decision or query remains accessible, without necessarily requiring all tracked state variables to be simultaneously correct. Related egocentric benchmarks also study episodic memory and future forecasting over long visual experience (Grauman et al., 2022), further broadening the range of temporal capabilities under evaluation.

Visual prediction provides a complementary perspective. Action-conditioned video prediction evaluates whether models can generate future observations under interventions or controls (Oh et al., 2015; Finn et al., 2016), while CLEVRER and Physion introduce more diagnostic tests of causal and physical prediction (Yi et al., 2019; Bear et al., 2021). These benchmarks are valuable for assessing visual and physical consistency, but they are not specifically designed to isolate whether a previously established persistent state is retained while hidden and then correctly expressed in a later prediction.

`StateReturn` targets this intersection. It evaluates whether the tracked state remains coherent as a whole after intervening interactions, and whether the hidden persistent state is subsequently reflected in the predicted future. By evaluating these two capabilities under a shared controlled-persistence protocol, `StateReturn` separates target-level correctness from whole-state coherence and full-frame visual quality from state-sensitive predictive correctness.

B `StateReturn` Benchmark Details

`StateReturn` contains two suites built around the same causal protocol: `StateReturn`-Text, which isolates persistent state maintenance from visual perception, and `StateReturn`-Visual, which instantiates persistent state maintenance and future prediction from RGB observations and actions. All experiments use the fixed Base Track, whose tracked components are `place`, `openness`, and `toggle_state`.

B.1 Benchmark Construction Protocol

Common construction protocol. Both `StateReturn`-Text and `StateReturn`-Visual are built around persistent state changes induced by interaction. Each trajectory contains a verified source interaction that changes one tracked state variable. The benchmark then inserts $K \in \{0, 1, 2, 4, 8, 16\}$ intervening interactions and evaluates the world state before the source state is directly revealed again. For a fixed trajectory, the six values of K correspond to nested prefixes of the same interaction sequence, so increasing K changes the amount of intervening experience without changing the source interaction or the preceding trajectory.

Tracked state. Both suites use the same entity-centric representation of current state. A state slot is defined by an entity and a typed component, and stores the latest valid value of that component. The reported benchmark tracks three components: `place`, `openness`, and `toggle_state`. A later valid interaction affecting the same slot supersedes its previous value. The canonical value domains for these components in the Text and Visual suites are summarized in Table 7.

Table 7: Canonical state domains used in the `StateReturn` benchmark.

Component	`StateReturn`-Text	`StateReturn`-Visual
`openness`	`open`, `closed`	`open`, `closed`
`toggle_state`	`on`, `off`	`on`, `off`
`place`	`on`, `inside`, or `held_by` an entity	in a zone identified by its zone ID

B.2 `StateReturn`-TEXT

Episode construction. `StateReturn`-Text is constructed from 120 AI2-THOR indoor scenes, with 80 scenes used for training, 20 for validation, and 20 for testing. Simulator objects are mapped to stable entity identifiers, and each executed trajectory is projected into the Base-Track state representation defined above.

Source interactions are obtained by executing state-changing actions in the environment. They cover three types of persistent change: opening or closing an object, switching a device on or off, and relocating a movable object through pickup and placement. A source interaction is retained only when replay confirms the intended typed state change without unintended changes to other Base-Track slots.

This procedure yields 600 qualified source interactions. Each source is instantiated in two episode forms. In a *single-update* episode, one interaction establishes the source state. In a *sequential-overwrite* episode, a later valid interaction updates the same slot again, requiring the current value to supersede the earlier one. Together, these two forms produce 1,200 episodes.

After the source update, we insert interactions involving other entities while preserving the current value of the source slot. These intervening interactions include unrelated state changes, same-component changes to other entities, and attempted interactions that leave the world state unchanged. For each episode, the resulting interaction sequence is fixed and reused to construct nested prefixes for $K \in \{0, 1, 2, 4, 8, 16\}$.

Structured and Natural representations. Each episode is released in two aligned representations of the same interaction trajectory. The Structured view encodes each interaction with a canonical action, participating entities, and execution outcome, while the Natural view describes the same event sequence in natural language. The two representations share the same episode, split, K, typed query, and ground-truth state, so they differ only in how interaction evidence is presented. Both representations preserve the observed outcome of each interaction, including whether it changes the world state or leaves it unchanged.

Dataset statistics and splits. The train, validation, and test partitions are disjoint at the scene level. State queries are generated at the valid values of K produced for each episode by the construction protocol. Table 8 summarizes the resulting dataset.

Table 8: Statistics of `StateReturn`-Text.

Split	Scenes	Episodes	Queries
Train	80	750	4,400
Validation	20	150	850
Test	20	300	1,800
Total	120	1,200	7,050

Across all splits, $K = 0$ contributes 1,200 queries, while each of $K \in \{1, 2, 4, 8, 16\}$ contributes 1,170. The final 7,050 queries consist of 2,400 `openness`, 2,340 `toggle_state`, and 2,310 `place` queries.

B.3 `StateReturn`-VISUAL

Visual trajectory construction. `StateReturn`-Visual is constructed in MiniGrid and exposes interaction history through local RGB observations and primitive actions. Each observation is a 288×288 RGB rendering of a 9×9 local view, and the action space consists of turning, moving

forward, picking up, dropping, and toggling objects. The observation does not include an inventory overlay. A public entity catalog provides stable entity identifiers and visual attributes, allowing entities to be referred to consistently across an interaction sequence.

State changes are produced through executable environment interactions. Object placement is changed by physically picking up, moving, and dropping an object, while door openness and switch state are changed through toggle interactions. Each source interaction is retained only when execution produces the intended change to its tracked slot. Subsequent interactions affect other entities, while the source entity is kept outside the local view at the evaluation checkpoints. As in the Text suite, the same executed trajectory is used to construct nested evaluation instances for $K \in \{0, 1, 2, 4, 8, 16\}$.

World families. `StateReturn`-Visual contains 120 worlds spanning four topology families, three grid sizes, and ten layout variants. Each topology is instantiated at grid sizes 17×17, 19×19, and 21×21 with ten variants, yielding 30 worlds per topology and 120 worlds in total. The local observation size remains fixed across all world sizes. Variants modify the spatial layout and the placement of doors, zones, interactive objects, switches, and agent starting configurations.

The four topology families are:

- **Chain.** Rooms are connected sequentially, producing a primarily linear spatial structure in which movement proceeds through a chain of connected regions. The 30 Chain worlds are shown in Figs. 5 and 6.
- **Four-Room Loop.** Four rooms are connected in a closed cycle, providing multiple routes through the environment while preserving a loop-shaped global structure. The 30 Four-Room Loop worlds are shown in Figs. 7 and 8.
- **Central Hub.** Multiple surrounding regions connect through a shared central area, creating a hub-and-spoke structure in which many transitions between regions pass through the hub. The 30 Central Hub worlds are shown in Figs. 9 and 10.
- **Corridor with Side Rooms.** A main corridor forms the backbone of the environment, with rooms branching from it at different locations. The 30 Corridor with Side Rooms worlds are shown in Figs. 11 and 12.

Evaluation checkpoints and counterfactual futures. Each world provides three source-state streams, corresponding to `place`, `openness`, and `toggle_state`. The source state is established respectively by relocating a movable object, changing the state of a door, or toggling a switch. The 120 worlds therefore yield $120 \times 3 = 360$ source-state streams.

For each stream, a source interaction is followed by a sequence of non-source state changes involving other entities. Evaluation checkpoints are taken after $K \in \{0, 1, 2, 4, 8, 16\}$ intervening interactions. At every checkpoint, the source remains outside the current local view. The same checkpoint is used for both *Visual State Maintenance* and *State-Consistent Prediction*. The former evaluates the current tracked state before the source is directly observed again, while the latter predicts the next RGB frame under a fixed forward action that brings the source back into view. With six checkpoints per stream, each capability contains $360 \times 6 = 2,160$ evaluation instances.

For State-Consistent Prediction, each factual future is paired with a counterfactual future constructed from the same visible prediction context. The factual and counterfactual branches have the same current RGB observation and the same next action, but differ in the persistent state of the source entity. For doors and switches, the counterfactual branch uses the alternative legal state; for object placement, the source object is assigned an alternative legal zone. The two resulting next frames therefore differ only in the visual consequence of the source-state difference. Their RGB difference identifies the state-sensitive region used by the prediction metrics described in Sec. B.4.

Dataset statistics and splits. World instances are disjoint across the train, validation, and test splits. Seven layout variants are used for training, one for validation, and two for testing, yielding 84 training worlds, 12 validation worlds, and 24 test worlds. All source streams and checkpoints derived from a world remain in the same split. Table 9 summarizes the resulting dataset.

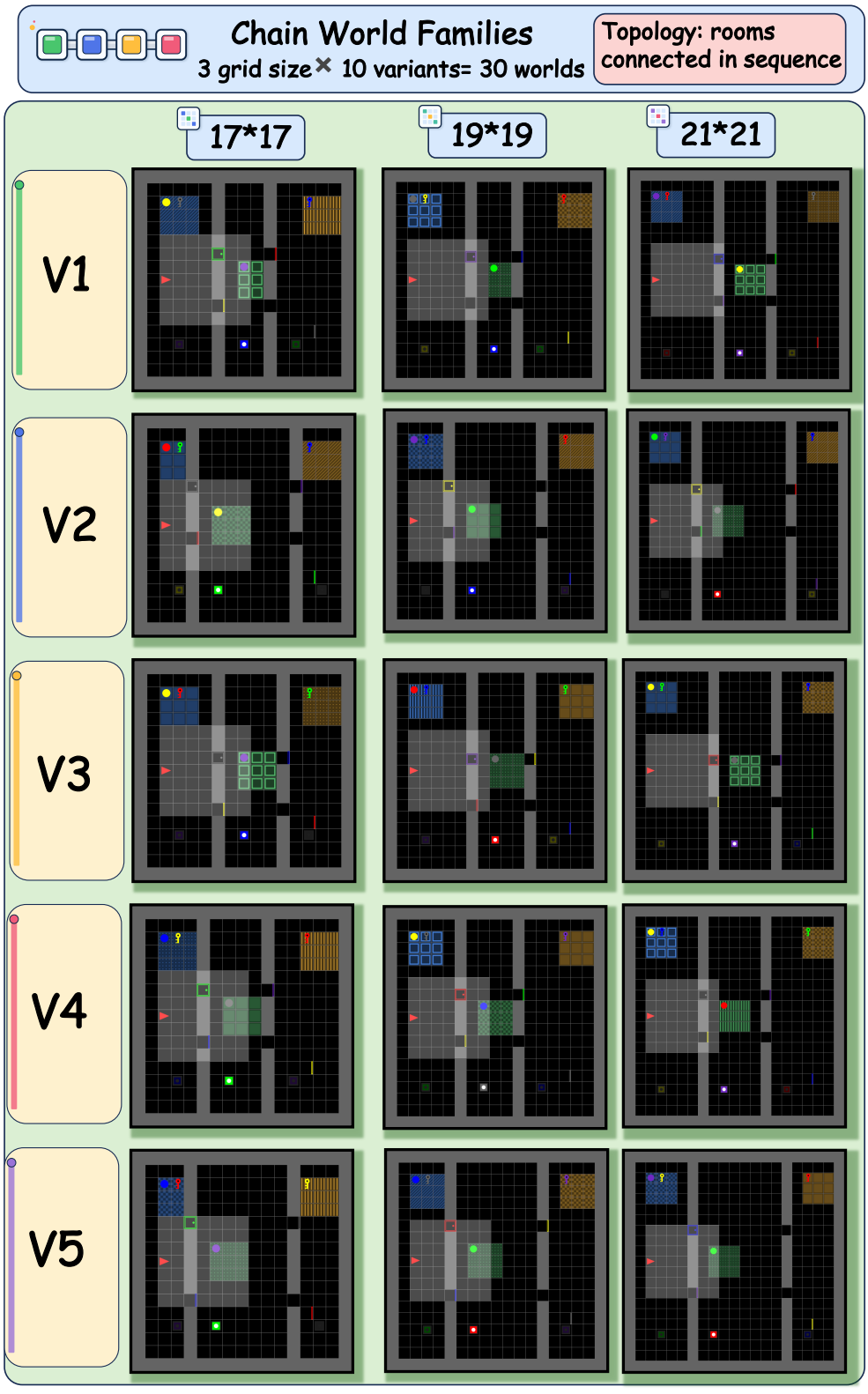


Figure 5: **Chain world layouts, variants V1–V5.** Each variant is shown at grid sizes 17×17, 19×19, and 21×21.

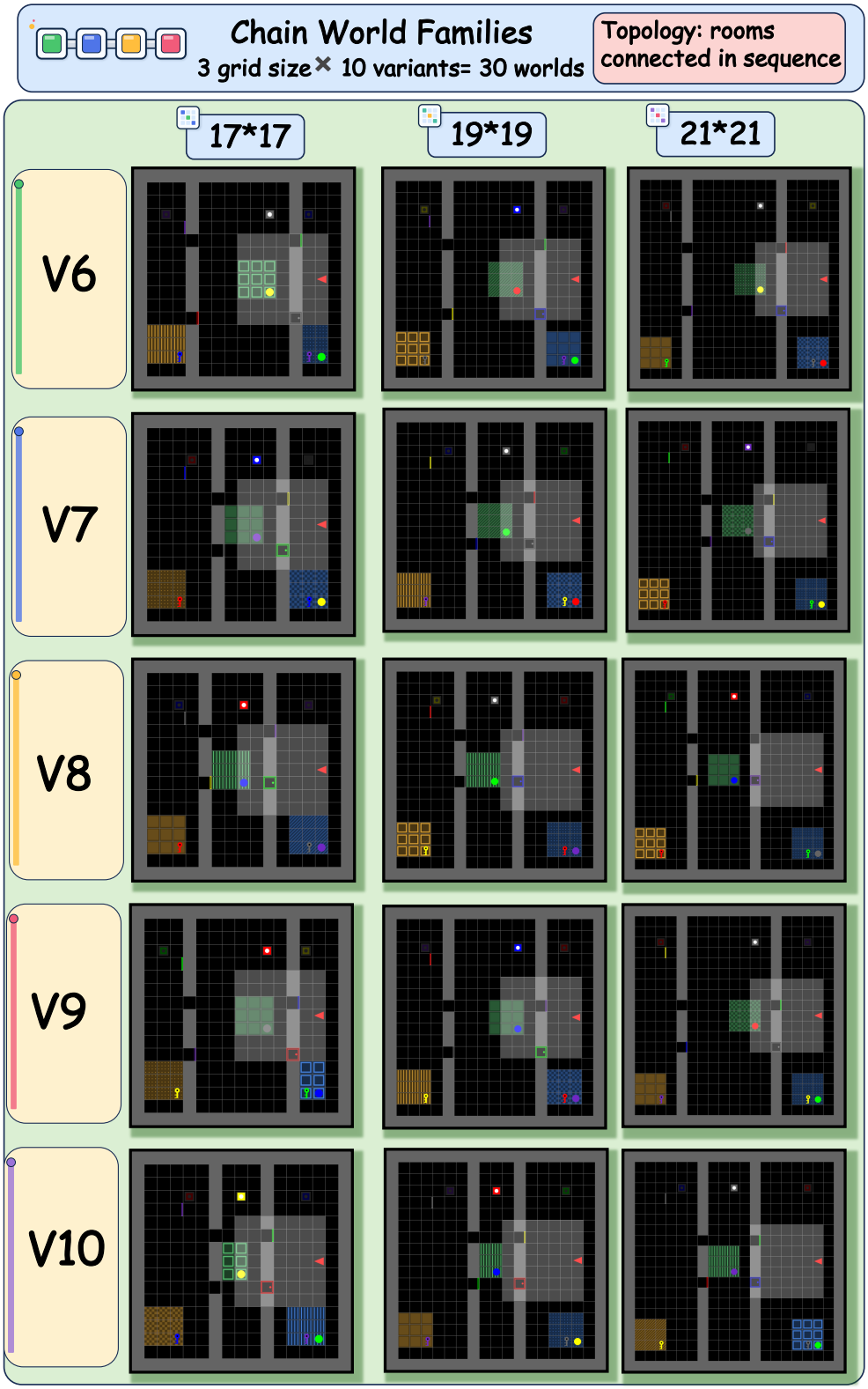


Figure 6: **Chain world layouts, variants V6–V10.** Each variant is shown at the three grid sizes.

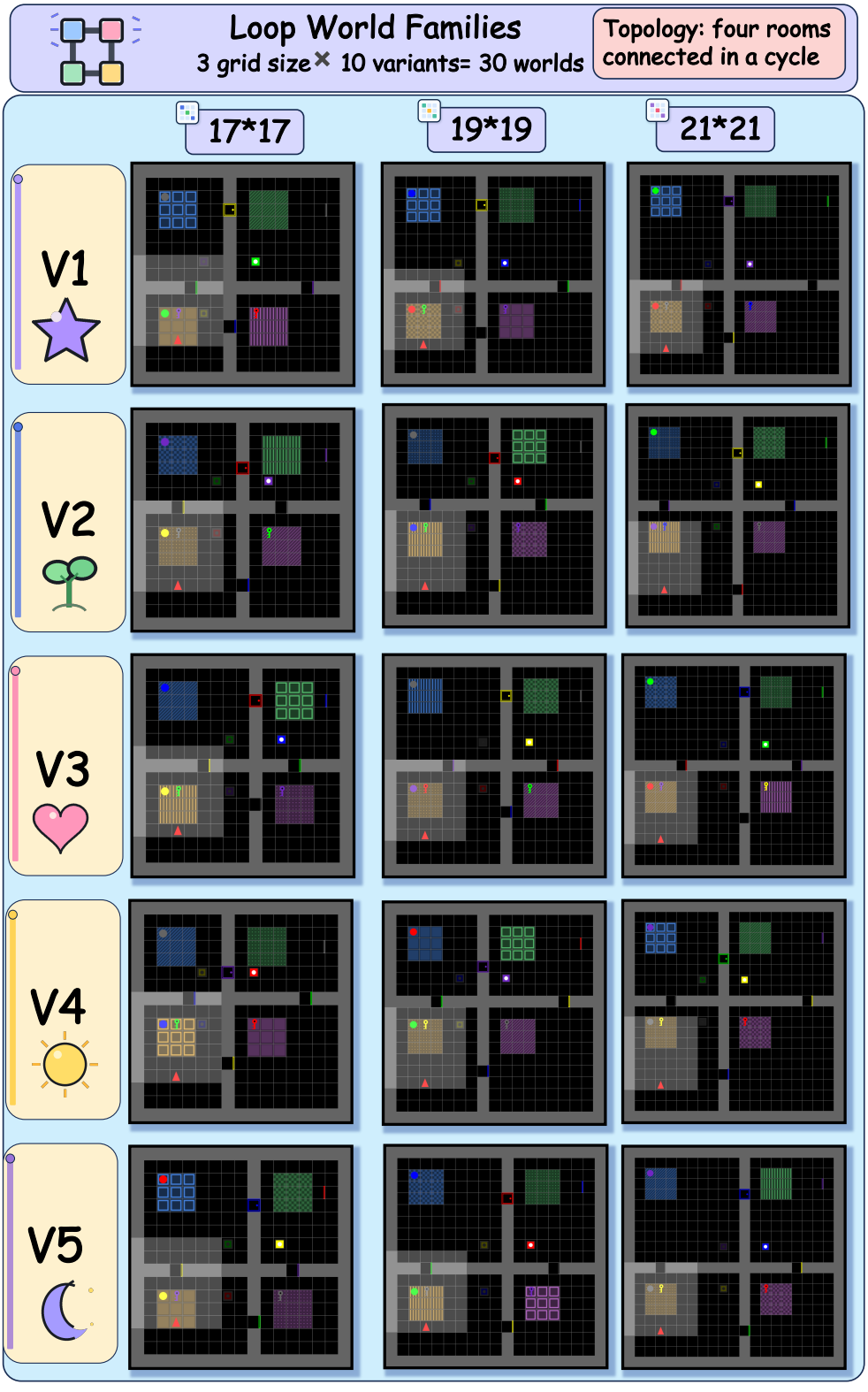


Figure 7: **Four-Room Loop world layouts, variants V1–V5.** Each variant is shown at grid sizes 17×17, 19×19, and 21×21.

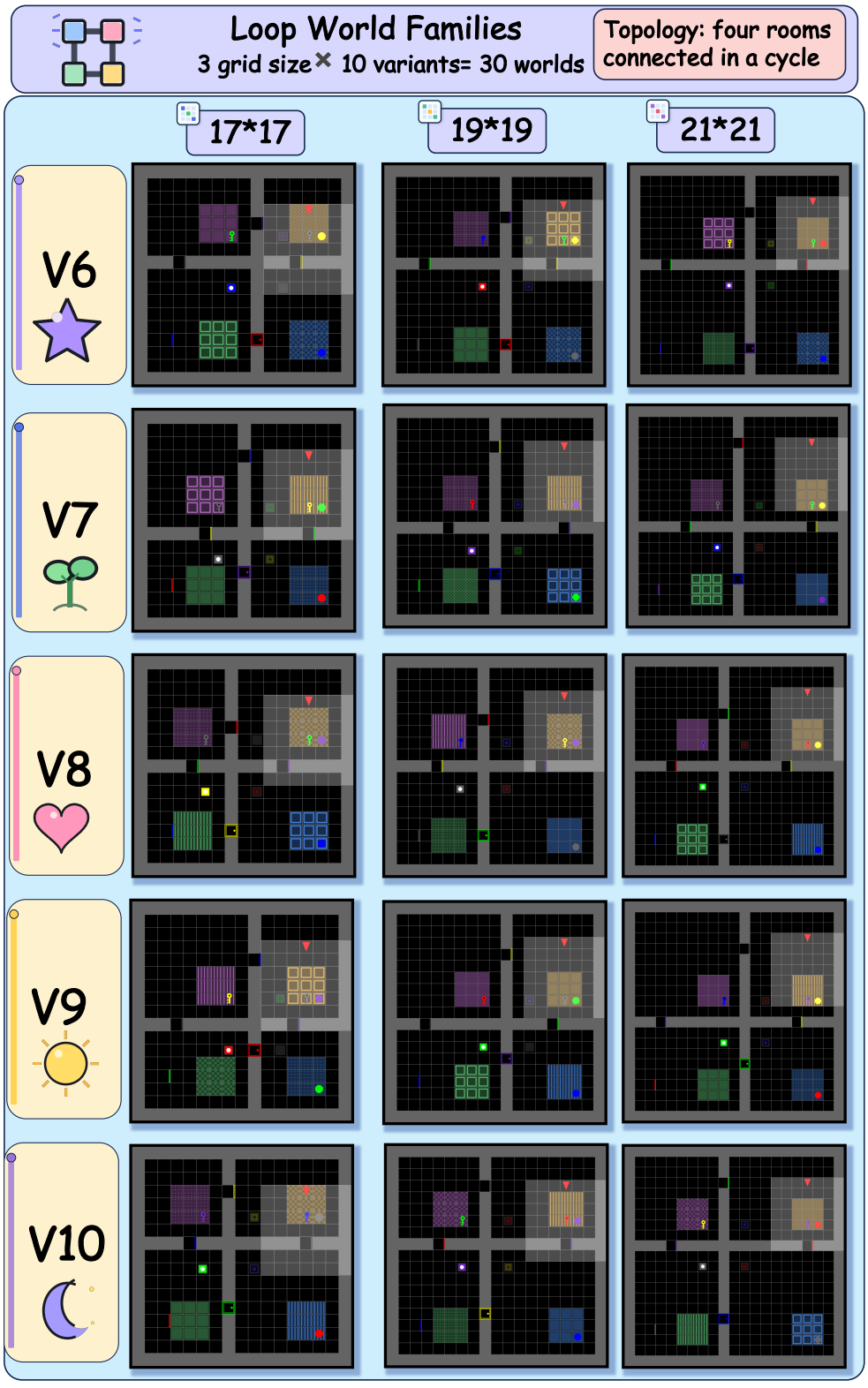


Figure 8: **Four-Room Loop world layouts, variants V6–V10.** Each variant is shown at the three grid sizes.

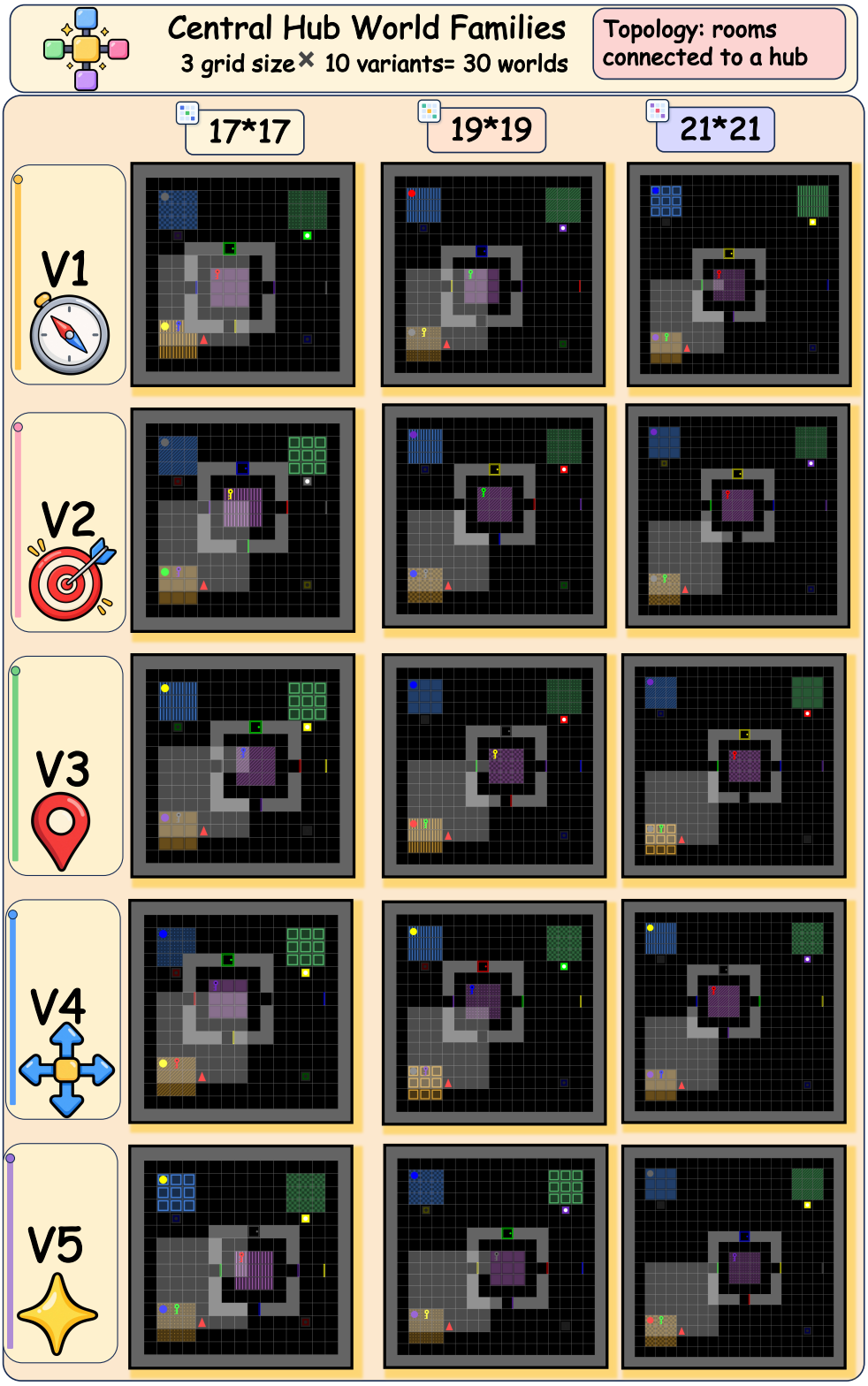


Figure 9: **Central Hub world layouts, variants V1–V5.** Each variant is shown at grid sizes 17×17, 19×19, and 21×21.

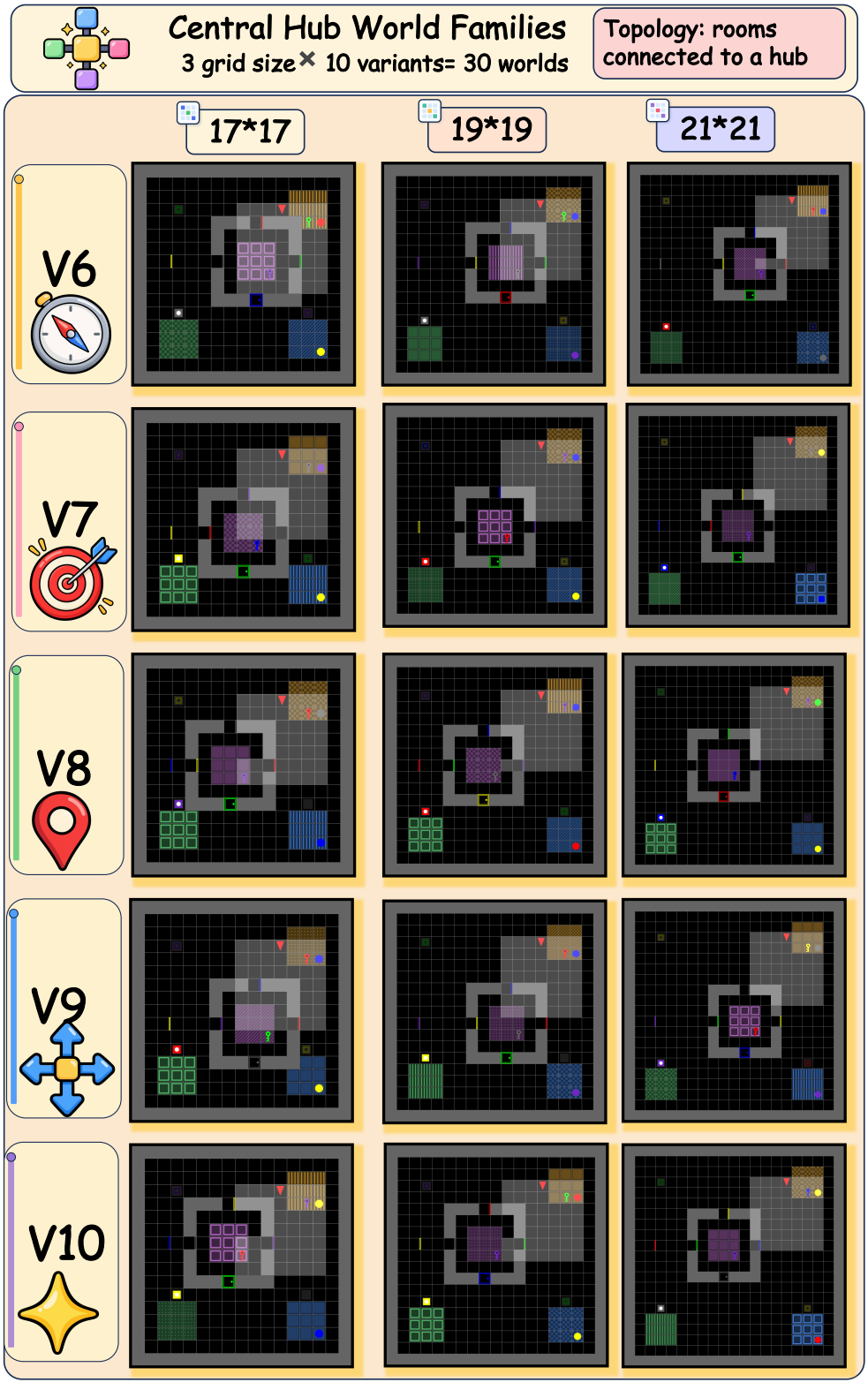


Figure 10: **Central Hub world layouts, variants V6–V10.** Each variant is shown at the three grid sizes.

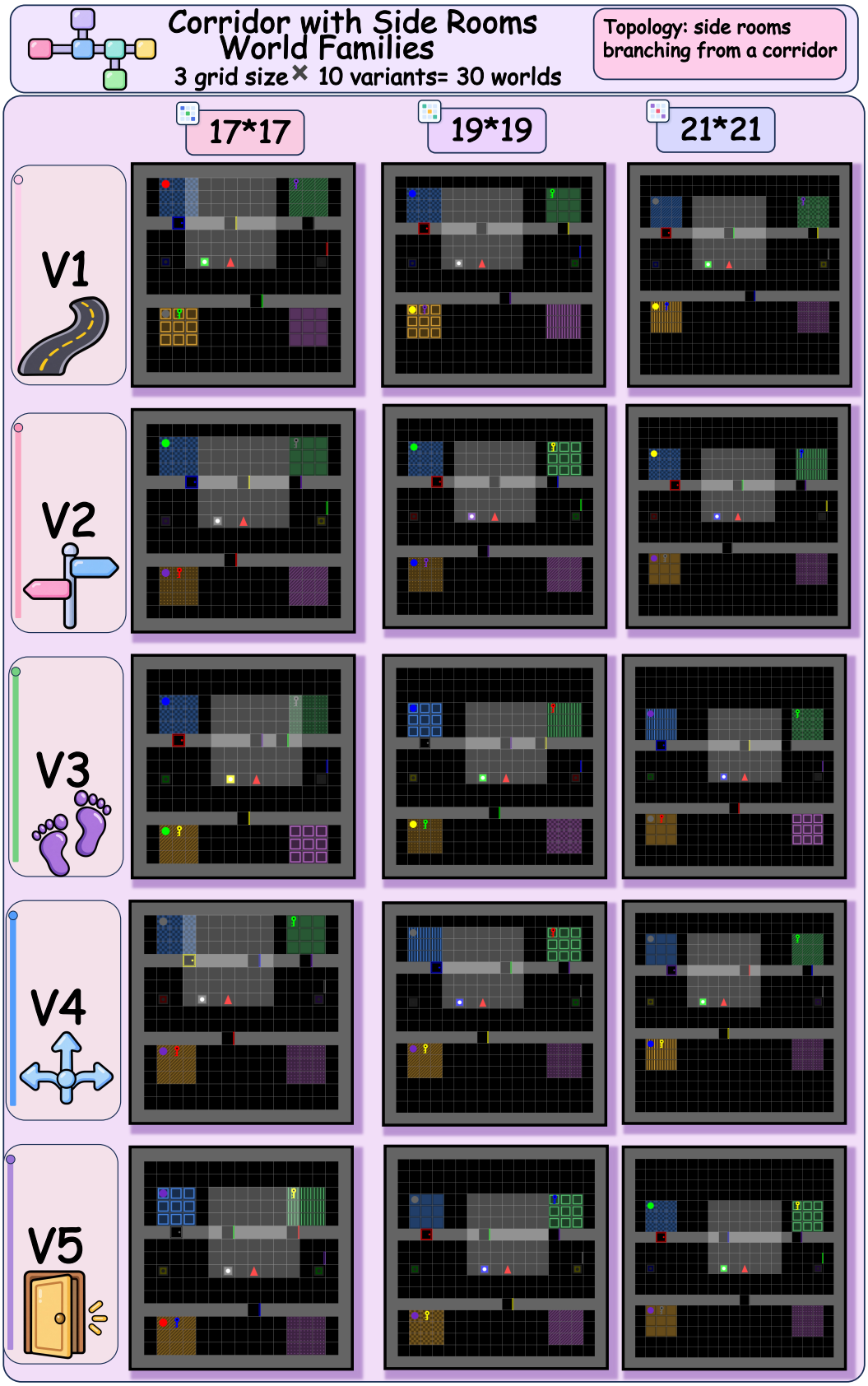


Figure 11: **Corridor with Side Rooms world layouts, variants V1–V5.** Each variant is shown at grid sizes 17×17, 19×19, and 21×21.

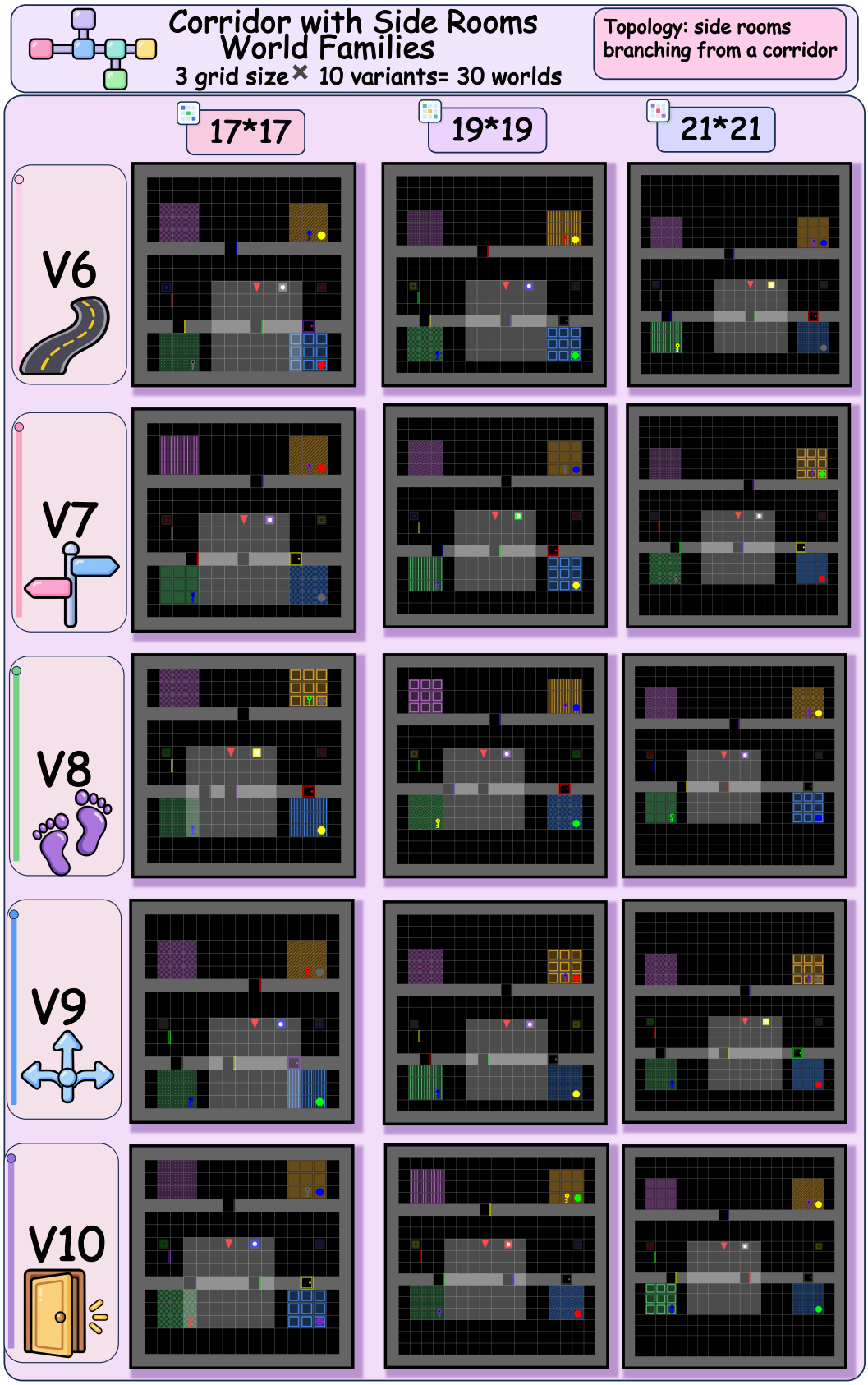


Figure 12: **Corridor with Side Rooms world layouts, variants V6–V10.** Each variant is shown at the three grid sizes.

Table 9: Statistics of `StateReturn`-Visual.

	Train	Validation	Test	Total
Worlds	84	12	24	120
Source streams	252	36	72	360
State-maintenance queries	1,512	216	432	2,160
Prediction queries	1,512	216	432	2,160

Across all splits, `StateReturn`-Visual contains 161,562 RGB frames. The test split contains 432 state-maintenance checkpoints. Because each checkpoint may contain multiple tracked state slots, these checkpoints comprise 1,699 slot-level state labels in total. This count is used for computing TSA and does not represent additional evaluation queries.

B.4 Evaluation Metrics

State-maintenance metrics. At each evaluation checkpoint i, let q_i denote the queried target slot, v_{i,q_i} its ground-truth value, and $\mathcal{S}_i$ the set of tracked state slots at that checkpoint. Predictions are evaluated by exact match over the canonical typed state representation.

Target State Exact Match (TSEM) measures whether the queried slot is predicted correctly:

$$\mathrm{TSEM} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}\left[\hat{v}_{i,q_i} = v_{i,q_i}\right]. \tag{10}$$

Full-State Exact Match (FSEM) requires all tracked slots at a checkpoint to be correct simultaneously:

$$\mathrm{FSEM} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}\left[\forall s \in \mathcal{S}_i,\ \hat{v}_{i,s} = v_{i,s}\right]. \tag{11}$$

Tracked Slot Accuracy (TSA) measures accuracy over individual tracked slots:

$$\mathrm{TSA} = \frac{\sum_{i=1}^{N}\sum_{s\in\mathcal{S}_i} \mathbb{I}\left[\hat{v}_{i,s} = v_{i,s}\right]}{\sum_{i=1}^{N}|\mathcal{S}_i|}. \tag{12}$$

TSEM captures correctness of the requested state, whereas FSEM evaluates whether the complete tracked current state is jointly correct. TSA provides a complementary slot-level measure that distinguishes broad state degradation from failures caused by a small number of incorrect slots.

State-consistent prediction metrics. For prediction checkpoint i, let $\hat{y}_i \in \mathbb{R}^{H\times W\times 3}$ denote the predicted next frame, y_i the factual next frame, and y_i^{cf} its paired counterfactual. We define the state-sensitive pixel set

$$M_i = \left\{p \;\middle|\; y_i(p) \neq y_i^{\mathrm{cf}}(p)\right\}, \tag{13}$$

which contains the pixels whose future appearance changes under the counterfactual source state.

For two RGB frames a and b, the mean absolute error restricted to M_i is

$$\mathrm{MAE}_{M_i}(a,b) = \frac{1}{3|M_i|}\sum_{p\in M_i}\sum_{c=1}^{3}|a_{p,c} - b_{p,c}|. \tag{14}$$

SourceMAE measures prediction error in this state-sensitive region:

$$\mathrm{SourceMAE} = \frac{1}{N}\sum_{i=1}^{N}\mathrm{MAE}_{M_i}(\hat{y}_i, y_i), \tag{15}$$

where lower is better. By restricting evaluation to pixels affected by the source state, SourceMAE avoids having unchanged background regions dominate the reconstruction error.

Counterfactual Preference (CFP) evaluates whether a prediction is closer to the factual future than to the paired counterfactual future. Define

$$e_{T,i} = \mathrm{MAE}_{M_i}(\hat{y}_i, y_i), \qquad e_{\mathrm{CF},i} = \mathrm{MAE}_{M_i}(\hat{y}_i, y_i^{\mathrm{cf}}). \tag{16}$$

Then

$$\mathrm{CFP} = \frac{1}{N}\sum_{i=1}^{N}\left(\mathbb{I}[e_{T,i} < e_{\mathrm{CF},i}] + \frac{1}{2}\mathbb{I}[e_{T,i} = e_{\mathrm{CF},i}]\right). \tag{17}$$

A CFP above 0.5 indicates that predictions more often favor the factual future, while 0.5 indicates no aggregate preference between the factual and counterfactual futures. We additionally report conventional reconstruction metrics such as PSNR and SSIM as diagnostics, while SourceMAE and CFP are the primary metrics for state-consistent prediction.

C Implementation and Training Details

C.1 Textual State Maintenance

Shared setup. Textual State Maintenance is evaluated separately on the aligned Structured and Natural representations of `StateReturn`-Text. For a given episode and checkpoint, all methods receive the same interaction evidence and typed state query; only the representation used to retain and resolve that evidence differs.

Full-History, Bounded-History, Rolling Summary, and all language-model components in `StateCommit` use Qwen2.5-Coder-7B-Instruct with greedy decoding. The Structured and Natural settings use representation-specific task instructions that define the corresponding event format, typed state domains, and execution-outcome semantics. All LLM readouts are decoded into the same canonical typed state format for evaluation. For methods that infer state at query time, target-state prediction and full-state prediction are obtained from separate readouts of the same retained memory representation.

History-based baselines. *Full-History* retains the complete interaction prefix and resolves the requested state from the full sequence at query time. For full-state evaluation, the same prefix is provided together with the tracked-slot set and the model predicts the current values of all tracked slots.

Bounded-History follows the same procedure but retains only the four most recent interaction events. The four-event budget is applied at the event level rather than through token truncation.

Rolling Summary incrementally maintains a textual summary of the interaction history under a 1,600-character budget. After each interaction, the previous summary and the new event are used to produce an updated summary that is intended to preserve currently supported state information and replace superseded values. At query time, the model receives this summary rather than the original event history.

Recurrent Memory. *Recurrent Memory* provides a learned implicit-state baseline that compresses interaction evidence online. Each public interaction is encoded with a frozen Qwen2.5-Coder-7B encoder. The final hidden representations are mean pooled to obtain a 3,584-dimensional event representation x_t, which is incorporated into a 768-dimensional recurrent state:

$$h_t = \mathrm{LayerNorm}\left(\mathrm{GRUCell}(x_t, h_{t-1})\right). \tag{18}$$

Once incorporated into h_t, earlier event text is not revisited at query time.

State prediction is component-specific. `Openness` and `toggle_state` use binary prediction heads, while `place` is decomposed into a three-way relation prediction (*on*, *inside*, or *held by*) and a pointer over the public candidate entities. The pointer uses 128-dimensional query and candidate-key projections, while component-specific adapters use 192-dimensional representations. The Qwen encoder remains frozen; only the recurrent module, projections, adapters, and typed prediction heads are optimized. Training hyperparameters and checkpoint selection are summarized in Sec. C.4.

C.2 VISUAL STATE MAINTENANCE

Visual Grounder. For visual state maintenance, `StateCommit` uses a learned Grounder to infer typed state changes from observed visual transitions. The Grounder is implemented as a ResNet-18 initialized from scratch. For each transition, the before and after RGB frames are concatenated along the channel dimension to form a six-channel input and resized to 160×160. The model is trained on 4,284 visual transitions, with 612 validation transitions used for checkpoint selection, by minimizing the sum of subject and value cross-entropy losses. In the controlled MiniGrid setting of `StateReturn`-Visual, the selected checkpoint achieves 100.0% EventFact exact accuracy on the validation set. Optimization details are reported in Sec. C.4.

The Grounder converts each visual transition into a typed EventFact by predicting both the affected entity and its resulting state. A subject head predicts one of 24 public type–color identities, corresponding to four interactive object types and six colors, while a value head predicts one of ten canonical state values covering door and switch states as well as zone patterns for object placement. The observed primitive action and public entity catalog restrict decoding to compatible entity and value types. Because each type–color identity is unique within a world, the predicted subject can be mapped directly to its stable public entity identifier.

Programmer Configurations. For the Visual State Maintenance results in Table 2, the learned Grounder is followed by a deterministic restricted Programmer. Given an EventFact and the current Active State, this component compiles the corresponding update into the restricted state-transaction format used by the Runtime, without reinterpreting the visual evidence or changing the semantic content of the EventFact.

For State-Consistent Prediction in Table 3 and the state-to-render ablations in Fig. 3, the Programmer is instead instantiated with Qwen2.5-Coder-7B-Instruct using greedy decoding and a maximum generation length of 96 tokens. Both configurations use the same learned Grounder, Auditor, and restricted Runtime. The rendering results are therefore reported under their corresponding Programmer configurations and are not intended to isolate the effect of Programmer capacity.

Visual memory baselines. *Bounded-History*, *Full-History*, and *Recurrent Memory* share the same frozen ResNet-18 transition backbone used by the Grounder. For each public before/after RGB pair, the backbone produces a 512-dimensional transition feature. This feature is combined with a 24-dimensional embedding of the corresponding primitive action and projected to a 256-dimensional transition representation.

A state query is represented using the queried entity's public type–color identity together with its requested component. All three baselines predict from the same canonical value vocabulary, with component-specific legality constraints applied during decoding. For `place`, a predicted zone pattern is mapped to the corresponding public zone identifier. Each baseline is trained to predict all tracked slots at an evaluation checkpoint, rather than only the designated target slot, so the same model outputs support TSEM, FSEM, and TSA evaluation.

The three baselines differ only in how they aggregate the causal sequence of visual transitions:

- **Bounded-History** retains the four most recent transition representations. Transitions outside this window cannot contribute to its state prediction.
- **Full-History** processes the complete causal prefix with a two-layer, eight-head Transformer encoder. A query token attends over the full observed transition sequence to produce the representation used for state prediction.
- **Recurrent Memory** processes transitions sequentially with a single 256-dimensional GRU and uses the final recurrent state for prediction.

All three baselines use 48-dimensional query embeddings, a 256-dimensional memory representation, and dropout of 0.10. Their aggregation modules and prediction heads are trained independently, while the visual transition backbone remains frozen. Optimization settings and checkpoint-selection criteria are summarized in Sec. C.4.

C.3 State-Consistent Prediction

Shared renderer setup. At each prediction checkpoint, the renderer predicts the next RGB frame from the current visual context and the fixed next action. All models are trained only on factual next-frame targets from the training split. Counterfactual futures and state-sensitive masks are used only for evaluation and do not provide training supervision.

All renderers use the same change-aware reconstruction objective. Let c denote the current frame and y the factual next frame. We define a training change mask

$$M_{\text{train}} = \mathbf{1}\left[\text{mean}_{c'} |y - c| > 0.02\right], \tag{19}$$

where c' indexes RGB channels. The rendering loss is

$$\mathcal{L}_{\text{render}} = \text{mean}\left[(1 + 4M_{\text{train}})|\hat{y} - y|\right]. \tag{20}$$

This mask is derived only from the factual training pair and is distinct from the state-sensitive mask used by the evaluator. Checkpoints are selected using whole-frame validation L_1 error.

History-based renderers. *Bounded-History*, *Full-History*, and *Recurrent Memory* share the same conditional U-Net renderer and the same frozen visual transition features used in Visual State Maintenance. They differ only in how the preceding interaction history is summarized into the conditioning representation.

Bounded-History conditions on the four most recent visual transitions, *Full-History* encodes the complete causal prefix with a Transformer, and *Recurrent Memory* uses the final hidden state of a GRU. Each memory encoder produces a 192-dimensional context vector that modulates the shared U-Net through feature-wise linear modulation. The U-Net uses channel widths 32, 64, 96, and 128 with skip-connected downsampling and upsampling blocks. By sharing the same image decoder, these baselines isolate the effect of history representation rather than decoder capacity.

`StateCommit` renderer. The `StateCommit` renderer augments local visual context with the maintained Active State and the slot-indexed Visual Binding associated with the relevant state. The Active State specifies the current semantic value, while the Visual Binding preserves public visual evidence associated with that value. Before rendering, the stored binding is projected from its source-view coordinates into the prediction-view coordinates induced by the known next action.

The renderer builds on a frozen Bounded-History renderer, which provides the same recent-four-transition visual context used by the corresponding history baseline. A separate residual branch receives the current RGB frame, the frozen Bounded-History prediction, the stored Visual Binding, its prediction-view-aligned version, and the typed Active-State condition. Binding features are integrated through cross-attention, while state features modulate the residual network through feature-wise linear modulation. The final prediction is

$$\hat{y} = \text{clip}_{[-1,1]}\left(\hat{y}_{\text{BH}} + \sigma(g)\tanh(\Delta)\right), \tag{21}$$

where $\hat{y}_{\text{BH}}$ is the frozen Bounded-History prediction and the residual branch predicts both a correction Δ and gate g. Only the residual branch is optimized; the underlying Bounded-History renderer remains frozen throughout training.

Binding alignment. A Visual Binding references a public RGB observation expressed in the camera coordinates of the interaction at which the corresponding state was last supported. Because rendering predicts the observation after a known next action, Binding Alignment projects the stored visual evidence into the camera coordinates of the corresponding prediction view. The transformation is estimated only from the observed RGB prefix and public actions. Over the observed prefix, agent rotations update the relative orientation, while forward motion is estimated through local image alignment between adjacent observations. The resulting transformations are composed over the intervening trajectory, with the known next action included in the final projection to the prediction view. The target RGB frame is never used to estimate this transformation.

C.4 Optimization and Compute

All trainable components are optimized on their corresponding training split, with model selection performed on the validation split. Table 10 summarizes the optimization settings and validation criteria used in the reported experiments.

Table 10: Optimization settings and validation criteria for the learned components.

Model	Optimizer	Batch	Epochs	Seed	Validation criterion
Text Recurrent Memory	AdamW, 10^{-3}	16	20	20260803	Macro component accuracy
Visual Grounder	AdamW, 3×10^{-4}	64	35	0	EventFact exact accuracy
Visual Bounded-History / Recurrent Memory	AdamW, 3×10^{-4}	32	50	0	TSEM $\rightarrow$ FSEM $\rightarrow$ TSA
Visual Full-History	AdamW, 3×10^{-4}	16	50	0	TSEM $\rightarrow$ FSEM $\rightarrow$ TSA
History-based renderers	AdamW, 3×10^{-4}	8	80	0	Whole-frame L_1
`StateCommit` residual renderer	AdamW, 3×10^{-4}	8	80	0	Whole-frame L_1

For the Visual state-maintenance baselines, checkpoints are ranked lexicographically by validation TSEM, FSEM, and TSA, in that order; training loss is used only to break any remaining ties.

All learned visual models use weight decay 10^{-4}, cosine learning-rate decay, and global gradient-norm clipping at 1.0. The Text Recurrent Memory instead uses an adaptive learning-rate schedule that reduces the learning rate by a factor of 0.5 after five epochs without validation improvement, together with gradient clipping at 5.0 and label smoothing of 0.05 for its binary and ternary classification heads. Its placement objective combines relation prediction and entity-pointer prediction:

$$\mathcal{L}_{\text{place}} = \mathcal{L}_{\text{kind}} + 0.5\,\mathcal{L}_{\text{pointer}}. \tag{22}$$

All experiments are run on NVIDIA RTX 4090 GPUs.

D Additional Experimental Results

D.1 Textual State Maintenance

Full-state readout across intervening interactions. The main results report FSEM for Textual State Maintenance. Here we additionally examine TSEM and TSA from the same predicted tracked-state map in Table 11. The additional metrics reinforce the distinction between target correctness and whole-state coherence. For example, at $K = 16$, Structured Recurrent Memory retains 87.67% TSEM but falls to 0.33% FSEM, while Natural Recurrent Memory retains 85.33% TSEM but likewise reaches only 0.33% FSEM. Full-History also exhibits a substantial gap in the Structured setting, with 89.33% TSEM versus 53.00% FSEM at $K = 16$. In contrast, `StateCommit` remains exact across all three metrics under Structured inputs and achieves 98.67% TSEM/FSEM with 99.48% overall TSA under Natural inputs.

Direct target answers versus full-state maintenance. The prompted history baselines can also answer the queried slot directly without first producing the complete tracked-state map. We denote the exact-match accuracy of this target-only readout as *AnswerEM* to distinguish it from TSEM computed from the full-state readout above. `StateCommit`, by contrast, reads the queried value directly from its maintained Active State.

Table 12 shows that target-only answering can remain substantially more accurate than reconstruction of the tracked state. The clearest case is Natural Full-History: its overall AnswerEM is 89.11%, compared with 48.67% TSEM and 43.67% FSEM from the full-state readout. At $K = 16$, the gap becomes even larger: Full-History still answers the individual query correctly in 82.33% of cases, while its map-derived TSEM and FSEM fall to 9.67% and 3.00%, respectively. Thus, success on an independently requested state value can substantially overstate the coherence of the current state recovered from the same interaction history.

Per-component results. Table 13 further separates the full-state readout by the component of the queried slot. FSEM and TSA continue to evaluate the complete tracked scope associated with each checkpoint, rather than only slots of the queried component.

Table 11: Textual state-maintenance results from the full-state readout. TSEM, FSEM, and TSA are computed from the same predicted tracked-state map.

Method	Metric	Overall	Intervening interactions K					
			0	1	2	4	8	16
Structured representation								
Full-History	TSEM	92.28	100.00	95.00	93.00	89.33	87.00	89.33
	FSEM	83.28	100.00	95.00	92.67	86.00	73.00	53.00
	TSA	96.21	100.00	97.50	97.44	96.25	94.85	91.24
Bounded-History	TSEM	50.44	100.00	95.00	93.00	4.33	5.00	5.33
	FSEM	48.67	100.00	95.00	92.67	3.67	0.33	0.33
	TSA	74.97	100.00	97.50	97.44	73.42	42.94	38.54
Rolling Summary	TSEM	85.72	98.00	91.33	87.67	83.33	77.00	77.00
	FSEM	77.89	98.00	89.67	85.67	77.33	59.00	57.67
	TSA	92.16	98.00	94.67	93.89	90.72	86.67	89.03
Recurrent Memory	TSEM	89.17	89.00	88.67	91.00	89.67	89.00	87.67
	FSEM	42.94	89.00	63.67	55.33	42.00	7.33	0.33
	TSA	73.60	89.00	81.83	78.56	73.64	66.81	51.76
`StateCommit`	TSEM	100.00	100.00	100.00	100.00	100.00	100.00	100.00
	FSEM	100.00	100.00	100.00	100.00	100.00	100.00	100.00
	TSA	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Natural representation								
Full-History	TSEM	48.67	88.00	82.67	61.67	34.00	16.00	9.67
	FSEM	43.67	88.00	81.33	54.67	27.33	7.67	3.00
	TSA	57.93	88.00	90.67	76.33	50.42	27.68	14.50
Bounded-History	TSEM	40.11	88.00	81.00	65.67	3.00	1.00	2.00
	FSEM	38.17	88.00	80.00	58.67	2.33	0.00	0.00
	TSA	63.04	88.00	89.83	79.22	60.11	32.77	28.31
Rolling Summary	TSEM	19.67	57.67	18.00	12.67	10.00	10.33	9.33
	FSEM	15.33	57.67	12.00	10.00	6.33	3.33	2.67
	TSA	38.86	57.67	49.33	39.22	32.67	29.82	24.48
Recurrent Memory	TSEM	87.17	91.33	87.00	87.00	86.67	85.67	85.33
	FSEM	28.00	91.33	37.33	18.00	17.00	4.00	0.33
	TSA	61.39	91.33	64.33	53.61	52.72	48.40	57.91
`StateCommit`	TSEM	98.67	98.67	98.67	98.67	98.67	98.67	98.67
	FSEM	98.67	98.67	98.67	98.67	98.67	98.67	98.67
	TSA	99.48	98.67	99.33	99.56	99.67	99.81	99.83

Table 12: Direct target-answer exact match (AnswerEM) on `StateReturn`-Text. AnswerEM evaluates the target-only readout and is therefore reported separately from the full-state TSEM/FSEM/TSA metrics above.

Method	Overall	Intervening interactions K					
		0	1	2	4	8	16
Structured representation							
Full-History	93.89	100.00	94.00	93.67	91.33	91.33	93.00
Bounded-History	64.83	100.00	94.33	94.33	34.00	33.33	33.00
Rolling Summary	92.00	99.33	95.33	91.67	89.67	88.67	87.33
Recurrent Memory	89.17	89.00	88.67	91.00	89.67	89.00	87.67
`StateCommit`	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Natural representation							
Full-History	89.11	98.00	93.00	93.00	83.67	84.67	82.33
Bounded-History	64.56	98.00	95.00	94.00	33.33	33.00	34.00
Rolling Summary	54.89	75.00	54.33	52.33	51.00	47.67	49.00
Recurrent Memory	87.17	91.33	87.00	87.00	86.67	85.67	85.33
`StateCommit`	98.67	98.67	98.67	98.67	98.67	98.67	98.67

The component breakdown shows that the target/full-state discrepancy is not restricted to a single state type. For example, Structured Recurrent Memory achieves TSEM of 100.00% on `openness` and 91.67% on `toggle_state`, yet the corresponding FSEM values are only 42.00% and 37.67%. Under Natural inputs, `place` is particularly difficult for the memory baselines: Full-History reaches only 25.83% FSEM and Recurrent Memory 14.50%. `StateCommit` is exact for all three Structured components and retains 100.00% TSEM/FSEM on both Natural binary components; its remaining Natural errors are confined to `place`, where it achieves 96.00% TSEM/FSEM and 98.43% TSA.

Table 13: Textual state-maintenance results by queried component. TSEM, FSEM, and TSA are computed from the same full-state readout.

Method	Metric	openness	place	toggle_state
Structured representation				
Full-History	TSEM	97.17	98.17	81.50
	FSEM	87.33	88.33	74.17
	TSA	97.35	98.06	93.24
Bounded-History	TSEM	52.50	49.83	49.00
	FSEM	50.17	49.83	46.00
	TSA	74.14	75.10	75.68
Rolling Summary	TSEM	81.00	93.33	82.83
	FSEM	70.00	85.83	77.83
	TSA	88.28	95.10	93.11
Recurrent Memory	TSEM	100.00	75.83	91.67
	FSEM	42.00	49.17	37.67
	TSA	72.30	77.66	70.84
`StateCommit`	TSEM	100.00	100.00	100.00
	FSEM	100.00	100.00	100.00
	TSA	100.00	100.00	100.00
Natural representation				
Full-History	TSEM	66.00	27.50	52.50
	FSEM	57.00	25.83	48.17
	TSA	66.47	43.45	63.87
Bounded-History	TSEM	48.17	26.83	45.33
	FSEM	45.00	26.17	43.33
	TSA	66.35	54.22	68.55
Rolling Summary	TSEM	9.67	28.00	21.33
	FSEM	7.33	20.17	18.50
	TSA	31.22	43.01	42.36
Recurrent Memory	TSEM	95.00	75.00	91.50
	FSEM	35.33	14.50	34.17
	TSA	68.37	46.02	69.76
`StateCommit`	TSEM	100.00	96.00	100.00
	FSEM	100.00	96.00	100.00
	TSA	100.00	98.43	100.00

Table 14: AnswerEM (%) for `place` queries under one versus three superseding updates. Each stratum contains 50 source episodes and 300 query prefixes. Δ denotes the change from $D_{\text{sup}} = 1$ to $D_{\text{sup}} = 3$.

View	Method	$D_{\text{sup}} = 1$	$D_{\text{sup}} = 3$	Δ
Structured	Full-History	100.00	95.67	-4.33
Structured	Bounded-History	50.00	50.33	+0.33
Structured	Rolling Summary	97.00	94.67	-2.33
Structured	Recurrent Memory	98.00	53.67	-44.33
Structured	`StateCommit`	100.00	100.00	0.00
Natural	Full-History	98.67	91.67	-7.00
Natural	Bounded-History	50.00	47.67	-2.33
Natural	Rolling Summary	48.00	38.67	-9.33
Natural	Recurrent Memory	98.00	52.00	-46.00
Natural	`StateCommit`	96.00	96.00	0.00

Superseding updates. Following the target-only readout used in the main superseding-update analysis, Table 14 reports AnswerEM additionally. Both strata contain 50 `place` episodes and 300 query prefixes. `StateCommit` is unchanged as the superseding depth increases, whereas Recurrent Memory drops by 44.33 and 46.00 points under Structured and Natural inputs, respectively.

D.2 Visual State Maintenance

Tracked-slot accuracy across intervening interactions. The main paper reports TSEM and FSEM for Visual State Maintenance. Table 15 complements these results with Tracked Slot Accuracy (TSA), which measures correctness over individual slots in the maintained state.

Table 15: Tracked Slot Accuracy (TSA, %) for Visual State Maintenance across different numbers of intervening interactions.

Method	Overall	Intervening interactions K					
		0	1	2	4	8	16
Bounded-History	45.20	52.78	58.33	44.91	47.56	43.44	40.58
Recurrent Memory	86.05	100.00	97.92	94.44	89.33	85.20	75.96
Full-History	88.46	100.00	98.61	98.15	95.73	83.77	79.23
`StateCommit`	100.00	100.00	100.00	100.00	100.00	100.00	100.00

Table 16: Visual state-maintenance performance by queried component (%). Each component contains 144 test checkpoints.

	`openness`			`place`			`toggle_state`		
Method	**TSEM**	**FSEM**	**TSA**	**TSEM**	**FSEM**	**TSA**	**TSEM**	**FSEM**	**TSA**
Bounded-History	46.53	16.67	46.21	25.00	9.72	40.67	54.17	19.44	48.70
Recurrent Memory	85.42	65.28	87.73	66.67	57.64	82.39	93.75	69.44	88.04
Full-History	100.00	70.83	87.18	98.61	67.36	88.56	100.00	70.14	89.60
`StateCommit`	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00

The additional metric further exposes the difference between accurate target readout and coherent whole-state maintenance. At $K = 16$, Full-History still achieves 98.61% TSEM, but its TSA has fallen to 79.23% and its FSEM to 13.89%. Thus, even when the designated target is usually recovered correctly, errors have accumulated elsewhere in the tracked state. Recurrent Memory degrades more gradually, reaching 75.96% TSA at $K = 16$, while Bounded-History falls to 40.58%. In contrast, `StateCommit` maintains every tracked slot correctly across all values of K.

Per-component results. Table 16 reports TSEM, FSEM, and TSA after grouping checkpoints by the component of the queried slot. Each queried component contributes 144 test checkpoints. Although the grouping is defined by the target component, FSEM and TSA are still computed over the complete tracked state at each checkpoint.

The results show that the target/full-state discrepancy is not confined to a single state type. Full-History achieves nearly perfect target accuracy for every component, including 100.00% TSEM on both `openness` and `toggle_state`, yet its corresponding FSEM remains between 67.36% and 70.83%. Persistent object placement is more difficult for the compressed-memory baselines: Bounded-History reaches only 25.00% TSEM on `place`, while Recurrent Memory reaches 66.67%. `StateCommit` maintains all three component groups exactly across TSEM, FSEM, and TSA.

Results across world families. We further group the same frozen test predictions by the four `StateReturn`-Visual topology families introduced in Sec. B.3. Each family contains 108 test checkpoints spanning all three state components, all three grid sizes, and all six values of K. The same trained checkpoint is used for every topology.

Table 17 shows that the main trends are consistent across world structures. Full-History maintains target accuracy above 98% in every family, but its FSEM ranges only from 64.81% to 73.15%. Recurrent Memory shows a smaller target/full-state gap but remains below 66% FSEM across all four families, while Bounded-History performs poorly throughout. `StateCommit` reaches 100.00% TSEM, FSEM, and TSA in every topology, showing that its whole-state performance is not driven by any single world family.

D.3 State-Consistent Prediction

Additional reconstruction metrics. The main paper evaluates state-consistent prediction primarily with SourceMAE and Counterfactual Preference (CFP), which focus on the part of the future that depends on the persistent source state. Table 18 complements these metrics with SourcePSNR and conventional whole-frame PSNR and SSIM.

Table 17: Visual state-maintenance performance across the four held-out world families. Each cell reports TSEM / FSEM / TSA (%), using the same frozen checkpoint for all families.

Method	Chain	Four-Room Loop	Central Hub	Corridor with Side Rooms
Bounded-History	47.22 / 18.52 / 46.15	46.30 / 14.81 / 49.29	36.11 / 14.81 / 41.08	37.96 / 12.96 / 44.34
Recurrent Memory	81.48 / 63.89 / 84.85	82.41 / 63.89 / 87.14	81.48 / 62.96 / 85.92	82.41 / 65.74 / 86.32
Full-History	100.00 / 73.15 / 89.51	98.15 / 64.81 / 85.95	100.00 / 66.67 / 88.50	100.00 / 73.15 / 89.86
`StateCommit`	100.00 / 100.00 / 100.00	100.00 / 100.00 / 100.00	100.00 / 100.00 / 100.00	100.00 / 100.00 / 100.00

Table 18: Additional reconstruction metrics for state-consistent prediction. SourcePSNR is computed over the same state-sensitive region as SourceMAE, whereas PSNR and SSIM are measured over the full frame.

Method	SourceMAE ↓	CFP ↑	SourcePSNR ↑	PSNR ↑	SSIM ↑
Bounded-History	0.2917	0.3704	7.614	18.722	0.8540
Full-History	0.2901	0.3773	7.354	17.344	0.8106
Recurrent Memory	0.2843	0.3681	7.593	20.271	0.8883
`StateCommit`	0.1190	0.8333	12.706	19.663	0.8813

Per-component results. We next separate the 432 paired test futures according to the source-state component, with 144 examples each for `openness`, `place`, and `toggle_state`. Table 19 shows that the advantage of `StateCommit` is consistent across all three state types.

Object placement is particularly difficult for the history-based renderers: their CFP values range from 0.0903 to 0.1181, indicating a strong tendency toward the counterfactual rather than the factual future. `StateCommit` instead reaches a CFP of 0.8958 on placement, while also improving openness from at most 0.5208 to 0.8611 and toggle state from at most 0.5000 to 0.7431. The corresponding SourceMAE reductions show that the improvement is not driven by a single state component.

Results across world families. Finally, we group the same test predictions by the four `StateReturn`-Visual topology families introduced in Sec. B.3. Each family contains 108 paired futures and uses the same globally selected checkpoint for every method.

As shown in Table 20, `StateCommit` achieves the lowest SourceMAE and highest CFP in every world family. Its CFP ranges from 0.7500 on Central Hub to 0.9074 on Corridor with Side Rooms, whereas the history-based baselines remain substantially lower across all topologies. The improvement therefore persists across distinct spatial organizations rather than being concentrated in a particular world structure.

D.4 Additional Qualitative Examples

Figures 13–15 provide additional qualitative examples. The Text examples illustrate failures to preserve the latest operative value after intervening or superseding interactions. The Visual State Maintenance examples further show that history-based methods can lose or misresolve persistent state when the source entity is no longer visible at query time. Finally, the prediction examples visualize the downstream consequence of these state errors: history-based renderers often produce futures inconsistent with the persistent source state, whereas `StateCommit` preserves the corresponding state-dependent visual outcome.

E Additional Analysis

E.1 Perception-Boundary Analysis

To examine the dependence of persistent state maintenance on upstream evidence interpretation, we replace the learned visual Grounder with Qwen2.5-VL-7B while keeping the same deterministic restricted Programmer, Auditor, and Runtime used for Visual State Maintenance unchanged. As shown in Table 21, replacing the Grounder reduces TSEM/FSEM/TSA from $100.00/100.00/100.00$ to $50.69/9.72/42.97$. The Qwen2.5-VL variant produces 39 invalid or `NO_FACT` outputs and leaves 146 target queries untracked.

Table 19: State-consistent prediction by source-state component. Each cell reports SourceMAE ↓ / CFP ↑ on 144 paired test futures.

Method	openness	place	toggle_state
Bounded-History	0.2455 / 0.4931	0.3243 / 0.1181	0.3054 / 0.5000
Full-History	0.2135 / 0.5208	0.3355 / 0.1181	0.3213 / 0.4931
Recurrent Memory	0.2241 / 0.5208	0.3180 / 0.0903	0.3107 / 0.4931
StateCommit	0.0838 / 0.8611	0.1254 / 0.8958	0.1477 / 0.7431

Table 20: State-consistent prediction across world families. Each family contains 108 paired test futures. Each cell reports SourceMAE ↓ / CFP ↑ using the same frozen checkpoint across all families.

Method	Overall	Chain	Four-Room Loop	Central Hub	Corridor with Side Rooms
Bounded-History	0.2917 / 0.3704	0.2899 / 0.3056	0.2518 / 0.5463	0.3209 / 0.3333	0.3043 / 0.2963
Full-History	0.2901 / 0.3773	0.2911 / 0.3148	0.2418 / 0.5556	0.3175 / 0.3611	0.3100 / 0.2778
Recurrent Memory	0.2843 / 0.3681	0.2835 / 0.3056	0.2570 / 0.5278	0.3106 / 0.3333	0.2859 / 0.3056
StateCommit	0.1190 / 0.8333	0.1153 / 0.8426	0.1044 / 0.8333	0.1752 / 0.7500	0.0810 / 0.9074

Table 21: Perception-boundary control for Visual State Maintenance. Only the visual evidence interpreter is changed; the downstream state-update mechanism is held fixed.

Visual Grounder	TSEM ↑	FSEM ↑	TSA ↑
Learned Grounder	100.00	100.00	100.00
Qwen2.5-VL-7B	50.69	9.72	42.97

This control illustrates the boundary between evidence interpretation and state maintenance. StateCommit can preserve and consistently apply state changes that are successfully extracted from the interaction, but it cannot recover a change that is absent from the extracted evidence.

E.2 Binding Alignment Diagnostic

A Visual Binding references slot-indexed public visual evidence from the interaction at which the corresponding state was established. The referenced observation is expressed in the camera coordinates of its source view. We evaluate the same stored binding before and after prediction-view alignment on 72 validation placement queries. As shown in Table 22, alignment reduces SourceMAE from 0.3476 to 0.0584 and increases CFP from 0.1111 to 0.8889. This shows that retaining the relevant visual evidence is not sufficient by itself; the evidence must also be projected into the prediction view induced by the known next action before it can reliably support rendering.

Table 22: Binding Alignment diagnostic on 72 validation placement queries. The same source-sensitive metrics are computed before and after prediction-view alignment.

Binding representation	SourceMAE ↓	CFP ↑
Source-view binding	0.3476	0.1111
Prediction-view-aligned binding	0.0584	0.8889



Figure 13: **Additional qualitative examples for Textual State Maintenance.** Each example shows the source interaction history, intervening interactions, the query, and the current-state predictions of all methods. History-based methods may return stale or untracked values, while `StateCommit` preserves the latest operative state.



Figure 14: **Additional qualitative examples for Visual State Maintenance.** Each example shows the source-state interaction, intervening visual history, query-time observation, ground-truth state, and predictions from the evaluated memory methods.

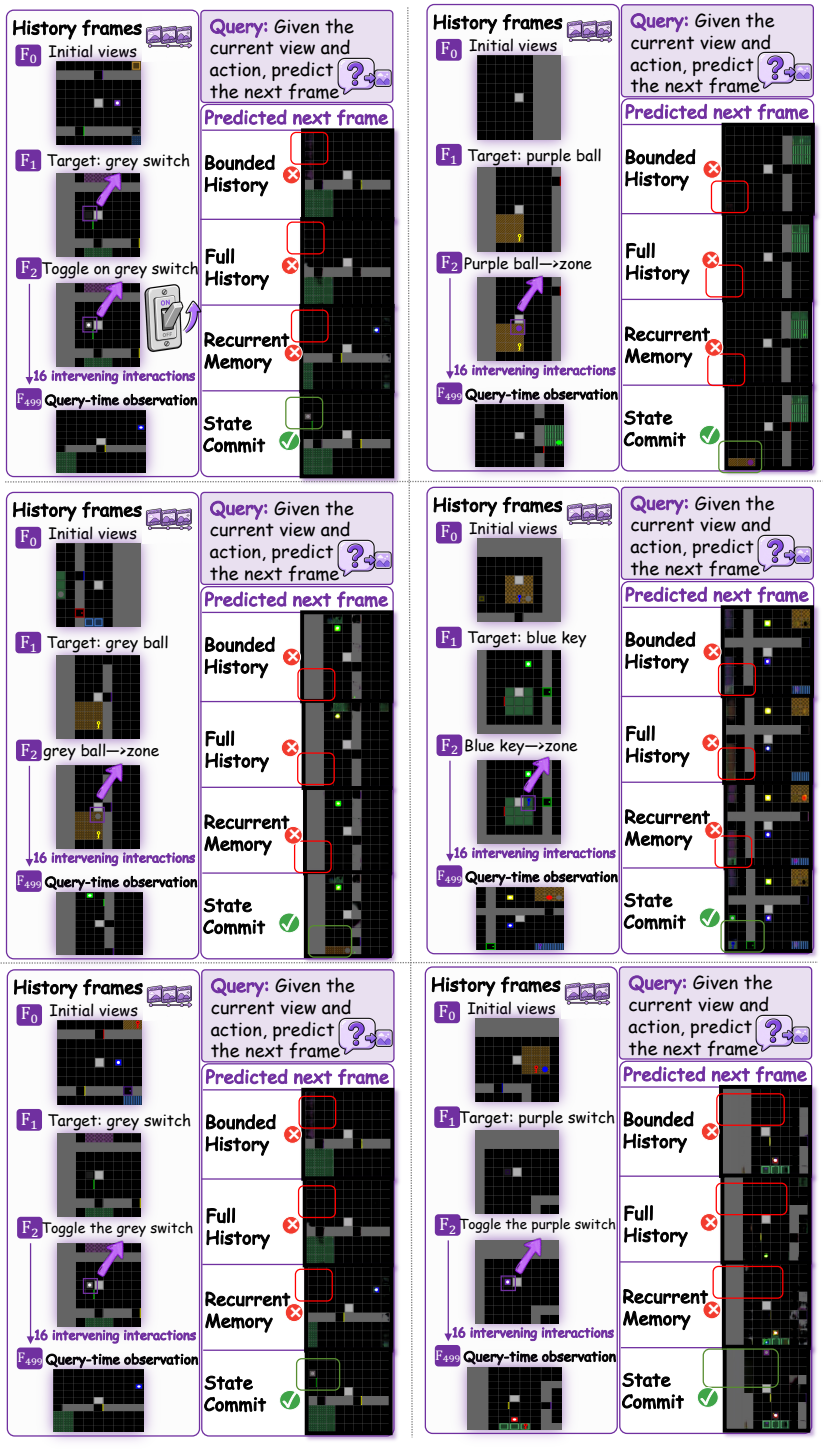


Figure 15: **Additional qualitative examples for State-Consistent Prediction.** Each example shows the earlier state-changing interaction, the query-time observation, and the predicted next frame. The highlighted regions indicate where the persistent source state affects the future observation. History-based renderers often produce a state-inconsistent future, while `StateCommit` better preserves the factual state-dependent outcome.