ENTERPRISE-JEPA: LEARNING LATENT TOOL-USE DYNAMICS FOR EFFICIENT AGENT PLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Enterprise workflows require tool-using agents to coordinate actions across applications, maintain state over long horizons, and avoid costly execution errors. World models can improve planning by predicting the consequences of candidate tool actions, but existing LLM-based approaches rely on autoregressive generation of imagined observations, making repeated planning computationally expensive. We introduce *Enterprise-JEPA*, the first action-conditioned latent world model for enterprise tool-using environments. Given the task context, interaction history, and a candidate action, Enterprise-JEPA predicts the latent representation of the resulting tool observation and decodes seven interpretable operational signals: action type, execution status, error, progress, side effects, information sufficiency, and task completion. A two-stage training procedure first learns latent dynamics from unlabeled trajectories and then fits operational-state heads using annotated transitions. Scaling unlabeled pretraining improves structured next-state prediction and surpasses a similarly sized LLM-based world model. Enterprise-JEPA also delivers strong downstream performance. With beam search, it consistently outperforms a no-world-model baseline across all five benchmarks, while its latent rollout mechanism is substantially faster than an LLM-based world model. These results demonstrate that latent world modeling enables more effective and efficient planning for enterprise tool-using agents, improving decision quality while reducing inference latency. Code is available at `https://anonymous.4open.science/r/enterprise_jepa-8421`.

1 INTRODUCTION

Large language model (LLM) agents are increasingly being deployed to execute complex workflows through external tools (Drouin et al., 2024; Boisvert et al., 2024; Yao et al., 2025). Enterprise environments pose unique challenges for such agents. Workflows often span multiple applications, observations are fragmented across databases, permissions restrict available actions (Malay et al., 2026), and a single erroneous tool invocation can irreversibly alter business state (Ruan et al., 2024). Recent benchmarks such as EnterpriseOps-Gym (Malay et al., 2026) and CRMArena-Pro (Huang et al., 2026) capture these difficulties through stateful, long-horizon tasks involving hundreds of tools, multi-turn interactions, and confidentiality constraints. Despite rapid advances in agent capabilities, performance on these benchmarks remains far from satisfactory, highlighting the need for more effective long-horizon planning.

World models (Ha & Schmidhuber, 2018; Hafner et al., 2020) provide a principled mechanism for anticipating the consequences of proposed actions and supporting deliberation before interacting with the environment. Recent approaches, including RAP (Hao et al., 2023), WebDreamer (Gu et al., 2025), Imagine-Then-Plan (ITP) (Liu et al., 2026), and ECHO (Shrivastava et al., 2026) use an LLM as the world model to predict future observations or intermediate states. While highly expressive, these methods rely on autoregressive generation, making planning computationally expensive. The cost grows with the number of action candidates, rollout horizon, and planning frequency.

Joint-embedding predictive architectures (JEPAs) offer an alternative method: predict the representation of a future observation rather than reconstructing its tokens (Assran et al., 2023; Maes et al., 2026). By avoiding autoregressive token generation, this formulation can predict the next state in a single forward pass, reducing the cost of evaluating candidate actions during planning. This prin-

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.

ciple has been studied primarily in vision and continuous control, where models predict changes in dense perceptual representations and actions are typically continuous and low-level (Assran et al., 2025). In contrast, tool-use agents issue discrete, semantically rich calls whose outcomes may depend on entities and observations introduced many steps earlier.

We introduce *Enterprise-JEPA*, to the best of our knowledge, the first action-conditioned JEPA for text-based tool agents in enterprise environments. Enterprise-JEPA extends latent predictive modeling to discrete, semantically rich tool calls by conditioning on the task context and recent action–observation history. It predicts the next tool observation in latent space and maps its contextual hidden state to seven interpretable operational fields, including execution status, errors, progress, information sufficiency, side effects, and task completion. This interface enables agents to evaluate candidate actions and plans without autoregressively generating intermediate observations.

Enterprise-JEPA is trained in two stages: self-supervised latent-dynamics learning from unlabeled trajectories, followed by supervised training of operational-state heads. Expanding the pretraining corpus improves average next-state macro-F1 and macro-recall, exceeding the LLM-based world model on both metrics. We evaluate downstream task execution across five benchmarks using three inference harnesses: revision, Imagine-Then-Plan, and beam-search planning. With beam search, Enterprise-JEPA exceeds the no-WM baseline on all five benchmarks, achieving an average success rate of 0.418 compared with 0.378. Under optimized serving, its world-model rollouts are up to $44.8\times$ faster than those of the LLM-based world model. Together, these results support latent prediction as a source of useful planning feedback at substantially lower inference cost.

Our contributions are threefold:

- We introduce **Enterprise-JEPA**, the first action-conditioned JEPA world model for text-based LLM agents in enterprise environments, to the best of our knowledge. It predicts tool-induced transitions directly in latent space rather than reconstructing future observations autoregressively.
- We develop a two-stage training procedure that separates latent-dynamics learning from operational-state decoding. Scaling experiments and ablations demonstrate the contribution of latent pretraining to next-state prediction quality.
- Experiments across five benchmarks and three planning harnesses establish better downstream performance with lower world-model latency than autoregressive simulation. Qualitative analysis further identifies argument grounding and task completion as sources of improvement.

2 Related Work

LLM world models and planning. Recent work has explored using LLMs as world models to support deliberative agent behavior (Zhang et al., 2026; Qiao et al., 2024; Shrivastava et al., 2026). RAP treats an LLM as both a reasoning policy and a world model within Monte Carlo tree search (Hao et al., 2023). In interactive environments, WebDreamer predicts the outcomes of candidate web actions before selection (Gu et al., 2025), while ITP learns textual environment dynamics and adaptively selects its lookahead depth (Liu et al., 2026). These approaches complement tool-using and reflective agents that improve decision-making through external interaction, feedback, or self-generated imagined trajectories (Yao et al., 2023; Shinn et al., 2023). However, textual world models require autoregressive decoding for every imagined transition, making repeated rollouts costly and susceptible to compounding generation errors. Enterprise-JEPA replaces generated observations with latent transitions and interpretable operational-state heads, allowing the model to support trajectory imagination efficiently.

Joint-embedding prediction. Joint-embedding predictive architectures (JEPAs) learn predictive representations by matching target embeddings rather than reconstructing high-dimensional observations (Assran et al., 2023). This formulation is related to latent-dynamics world models, which support planning by predicting abstract future states instead of raw observations (Ha & Schmidhuber, 2018; Hafner et al., 2020). LeJEPA further introduces SIGReg to regularize joint-embedding objectives and mitigate representational collapse (Balestriero & LeCun, 2025). Unlike prior JEPA methods developed primarily for visual or continuous domains (Assran et al., 2025), Enterprise-JEPA models discrete, history-dependent tool interactions. It predicts a latent representation of the next observation and decodes the state into operational categorical variables.

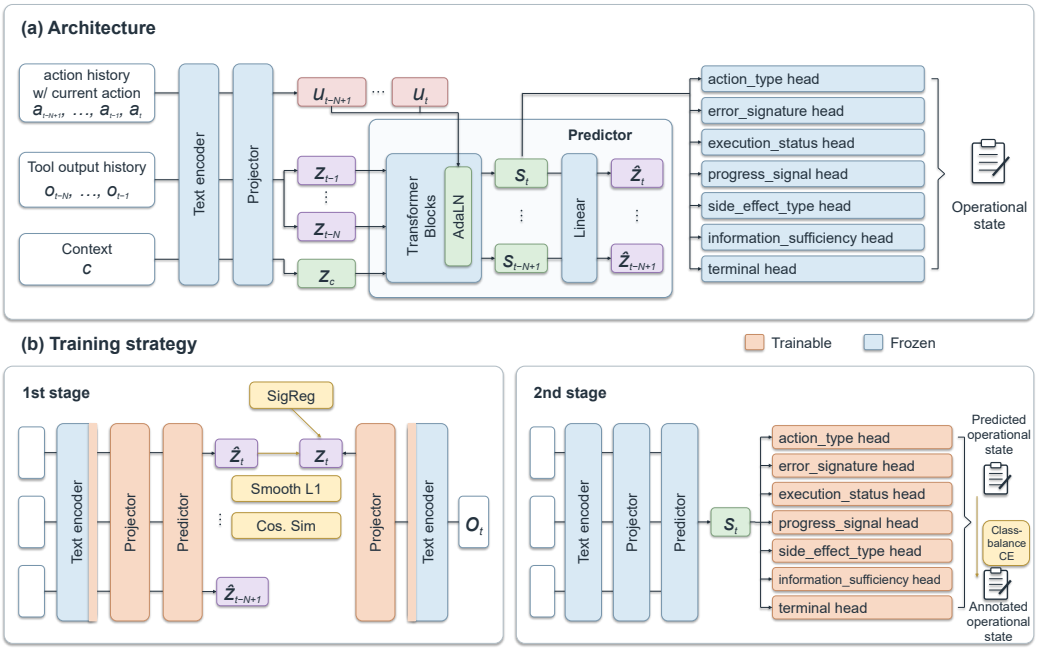


Figure 1: **Enterprise-JEPA architecture and training.** (a) Enterprise-JEPA comprises a text encoder, projector, Transformer-based predictor, and seven classification heads that predict the operational state resulting from the current action, history, and context. (b) Stage 1 learns action-conditioned latent dynamics, after which Stage 2 freezes the dynamics components and trains the seven operational-state heads.

3 METHOD

3.1 PROBLEM FORMULATION AND OVERVIEW

At interaction step t, an agent receives task context c and history $h_t = \{(a_i, o_i)\}_{i<t}$, where a_i is a tool action and o_i is its observed result induced by a_i. An agent proposes $a_t \sim \pi_{\text{base}}(\cdot \mid c, h_t)$, and the environment produces o_t. World models predict the likely outcome of a_t without execution and expose them to the agent through inference harnesses. We first consider an LLM-based world model, an autoregressive world model that predicts o_t or a specific state as text, token by token, conditioned on c, h_t, and a_t. In contrast, Enterprise-JEPA learns action-conditioned dynamics in a latent space and decodes the predicted transition into a structured operational state, trained in two phases (Figure 1 (b)) as described in the next section.

3.2 ENTERPRISE-JEPA

Enterprise-JEPA is trained in two stages that separate representation learning from readout. Stage 1 learns an action-conditioned latent representation from a large corpus of unlabeled trajectories. Because it requires no annotations, it can leverage data at a scale far beyond that of any labeled dataset. Stage 2 then maps that latent space onto the small set of states that matter for agent operation.

Architecture. The architecture overview is shown in Figure 1 (a). Let E_ϕ be a pretrained text encoder and P_ψ its projector. We encode context (system and task prompt), observations, and actions as $\boldsymbol{z_c} = P_\psi E_\phi(c)$, $\boldsymbol{z_i} = P_\psi E_\phi(o_i)$, and $\boldsymbol{u_i} = P_\psi E_\phi(a_i)$, respectively. A causal Transformer (Vaswani et al., 2017) T_θ receives $\boldsymbol{z_c}$ and the N most recent observation representations, ordered from oldest to newest. Actions are not added as separate sequence tokens. Instead, each observation representation $\boldsymbol{z}_i$ is conditioned on the subsequent action representation $\boldsymbol{u}_{i+1}$ through adaptive layer normalization at the corresponding Transformer position. Thus, the candidate action representation $\boldsymbol{u}_t$ conditions the most recent observation representation $\boldsymbol{z}_{t-1}$:

$$(\hat{\boldsymbol{z}}_t, \boldsymbol{s_t}) = T_\theta\big(\boldsymbol{z_c}, \{(\boldsymbol{z_i}, \boldsymbol{u_{i+1}})\}_{i=t-N}^{t-1}\big)\,. \tag{1}$$

Here $\hat{\boldsymbol{z}}_{\boldsymbol{t}}$ predicts the representation of the next observation produced by a_t, whereas it is projected from $\boldsymbol{s}_{\boldsymbol{t}}$, the Transformer's final hidden state.

Stage one: latent dynamics. Stage 1 trains the model to align the predicted latent representation of the next observation, $\hat{\boldsymbol{z}}_{\boldsymbol{t}}$ in Eq. (1), with its target representation, $\boldsymbol{z}_{\boldsymbol{t}} = P_\psi E_\phi(o_t)$, through self-supervised learning. The text encoder receives different inputs in the prediction and target branches: the prediction branch encodes the task context, interaction history, and current action, whereas the target branch encodes only the resulting tool output. This separation prevents the model from minimizing the prediction loss by matching shared input text rather than learning action-conditioned transitions. We minimize the following objective:

$$\mathcal{L}_{\text{dyn}} = \ell_{\text{pt}}(\hat{\boldsymbol{z}}_{\boldsymbol{t}}, \text{sg}(\boldsymbol{z}_{\boldsymbol{t}})) + 1 - \cos(\hat{\boldsymbol{z}}_{\boldsymbol{t}}, \text{sg}(\boldsymbol{z}_{\boldsymbol{t}})) + \lambda_{\text{sig}} \mathcal{R}_{\text{SIG}}(\boldsymbol{z}_{\boldsymbol{t}}). \tag{2}$$

where $\text{sg}(\cdot)$ denotes the stop-gradient operation, ℓ_{pt} is the Smooth-L_1 loss (Girshick, 2015), which reduces sensitivity to outliers, and λ_{sig} controls the regularization strength. SIGReg compares random one-dimensional latent projections with Gaussian samples and penalizes non-zero means and non-unit standard deviations (Balestriero & LeCun, 2025), which discourages representational collapse. The final four encoder layers, projector, and predictor are trainable.

Stage two: operational-state decoding. After latent pretraining, the encoder, projector, and predictor are frozen, and seven classification heads are trained on annotated transitions to predict interpretable operational states.

Structured operational state. Enterprise-JEPA predicts seven operational fields: action type, execution status, error signature, progress signal, side-effect type, information sufficiency, and task completion. The fields and their categories are application-agnostic, enabling transfer across enterprise environments without relying on benchmark-specific details. They describe the operational consequences of an action without reconstructing the full tool observation. Because the schema omits details such as exact identifiers and argument values, it serves as a predictive interface for planning rather than a sufficient statistic of the environment. Full vocabularies appear in Table 1.

Table 1: Operational-state vocabularies. The `information_sufficiency` field indicates whether the information collected so far is sufficient to proceed to the next step.

Field	Categories
action_type	read, search, create, update, delete, run, test, communicate, clarify, validate, unknown
execution_status	success, failure, partial, no-op, unknown
error_signature	none, not-found, permission-denied, invalid-argument, dependency-missing, test-failed, timeout, policy-risk, parse-error, runtime-error, unknown
progress_signal	positive, negative, neutral, unknown
side_effect_type	none, retrieved, modified, created, deleted, sent, installed, validated, failed-validation, executed, unknown
information_sufficiency	sufficient, insufficient
task_completion	finished, not-finished

The hidden state $\boldsymbol{s}_t$ captures task context and interaction history, whereas $\hat{\boldsymbol{z}}_t$ is projected from $\boldsymbol{s}_t$ to represent the predicted tool observation. The classification heads take $\boldsymbol{s}_t$ as input to retain contextual information relevant to operational-state prediction. For each field $f \in \mathcal{F}$, the corresponding head produces

$$\boldsymbol{p}_f = \text{softmax}(C_f(\boldsymbol{s}_t)), \qquad f \in \mathcal{F}, \tag{3}$$

where C_f maps $\boldsymbol{s}_t$ to logits over the categories of field f. The predicted category is $\arg\max_k [\boldsymbol{p}_f]_k$.

The heads are trained using class-balanced cross-entropy:

$$\mathcal{L}_{\text{head}} = -\frac{1}{|\mathcal{F}|} \sum_{f \in \mathcal{F}} w_{f,y_f} \log p_f(y_f \mid c, h_t, a_t), \tag{4}$$

where y_f is the target category for field f. Field-specific class weights w_{f,y_f} address class imbalance using the effective number of samples in each class (Cui et al., 2019).

4 TRAINING AND NEXT-STATE PREDICTION

We first describe how Enterprise-JEPA is trained exactly and how accurately it predicts the next operational state; Section 5 then describes how its predictions are exposed to an agent at inference time and the resulting task execution.

4.1 BENCHMARKS AND DATA

Benchmarks. Enterprise-JEPA is trained and evaluated as a next-state predictor on EnterpriseOps-Gym and CRMArena-Pro, which provide annotated operational states from trajectories generated on their training tasks.

Data. For the data to train and evaluate Enterprise-JEPA, Stage 1 requires only action–observation transitions available in raw trajectories. To assess the effect of data size, we construct two types of corpora. The *Small corpus* contains 4,515 training-task trajectories from in-distribution benchmarks, EnterpriseOps-Gym and CRMArena-Pro. The *extended corpus* augments this set with filtered tool-use and software-engineering trajectories from the Agent Data Protocol repository (Song et al., 2026), yielding 316,200 trajectories in total. The description and the number of trajectories for each source are noted in Appendix B.

Stage 2 uses annotated transitions collected from three agents spanning a range of performance levels—Qwen3-8B (Yang & et al, 2025), GPT-5.1 (OpenAI, 2025), and GPT-5.5 (OpenAI, 2026b)—thereby capturing both successful behaviors and diverse failure modes. For each transition, the annotation input contains the task context, interaction history, proposed action, and observed tool output. Three annotator models—GPT-5.6 (OpenAI, 2026c), Claude Sonnet 5 (Anthropic, 2026b), and Claude Opus 5 (Anthropic, 2026a)—independently assign a category to each field of the schema. Their predictions are aggregated by majority vote; when all three models assign different categories, the label from an earlier GPT-5.4-mini (OpenAI, 2026a) annotation pass is used to resolve the tie. We obtain the label through field-wise majority voting, and automatically reject out-of-schema responses, uninformative all-unknown labels, and conflicting duplicate inputs. This procedure yields 17,066 annotated action–state pairs for training and 5,854 for testing. We provide the annotation prompts, agreement statistics, and the adjudication instrument in Appendix C.

4.2 METRICS AND BASELINES

Next-state prediction asks whether the model anticipates the operational consequence of a proposed action before execution. We report macro-F1 and macro-recall for each prediction head, together with their unweighted mean across heads. We do not report accuracy because the class distributions are imbalanced.

We compare Enterprise-JEPA trained with the small and expanded Stage-1 corpora against a parameter-matched autoregressive LLM-based WM (a state-output LLM-WM) trained to emit the same seven-field state. This world model also follows a two-stage training procedure: it is first trained to predict tool outputs and subsequently fine-tuned to generate operational states as text.

4.3 IMPLEMENTATION DETAILS

Enterprise-JEPA initializes E_ϕ from Qwen3-Embedding-0.6B (Zhang et al., 2025) and uses an $N = 8$ history window. Stage 1 trains for one epoch with the top four encoder layers unfrozen; Stage 2 fits seven categorical heads using imbalance-aware losses for five epochs. The state-output LLM-WM is based on Qwen3-0.6B (Yang & et al, 2025) to match the parameter scale of Enterprise-JEPA. Full optimization, precision, and hardware are provided in Appendix E.1.

4.4 RESULTS

Table 2 reports next-state prediction across all seven outputs. Expanding the unlabeled Stage 1 corpus raises average macro-F1 from 0.545 to 0.603 and macro-recall from 0.574 to 0.623, with the largest gain in execution status (macro-F1: 0.379 to 0.608). Improvements in error signature, action type, side effects, and information sufficiency suggest that broader pretraining particularly benefits

Table 2: **Next-state prediction performance (macro-F1/macro-recall)**. Expanded latent pretraining improves Enterprise-JEPA's average performance beyond the autoregressive LLM-WM, with particularly strong gains in execution-status and error-signature prediction. Omitting Stage 1 substantially reduces overall performance.

Field	LLM-WM, expanded	JEPA, no Stage 1	JEPA, small	JEPA, expanded
Action type	**0.669**/0.687	0.295/0.333	0.567/0.655	0.637/**0.702**
Error signature	0.317/0.312	0.217/0.211	0.366/0.382	**0.421/0.395**
Execution status	0.383/0.361	0.266/0.258	0.379/0.361	**0.608/0.607**
Progress signal	0.601/0.571	0.432/0.441	**0.608/0.579**	0.597/0.573
Side-effect type	0.529/0.571	0.241/0.278	0.489/0.592	**0.532/0.618**
Information sufficiency	0.688/0.703	0.681/0.686	0.695/0.704	**0.711/0.719**
Task completion	0.702/0.688	0.699/0.735	**0.714**/0.746	**0.714/0.750**
Average	0.556/0.556	0.404/0.420	0.545/0.574	**0.603/0.623**

long-tailed transitions, as shown in Figure 3. The expanded Enterprise-JEPA also outperforms the state-output LLM-WM in aggregate macro-F1 (0.603 vs. 0.556) and macro-recall (0.623 vs. 0.556), especially on execution status and error signature.

To verify that these gains come from the encoder, projector, and predictor learning environment dynamics during Stage 1 rather than from Stage 2's labeled fine-tuning alone, we ablate Stage 1 entirely (JEPA, no Stage 1): the encoder is left at its off-the-shelf initialization, and the projector/predictor at random initialization, and only the classification heads are fit to the same annotated operational states. Average macro-F1/recall collapses to 0.404/0.420, below even JEPA, small, and the LLM-WM baseline. The drop is sharpest on exactly the fields Stage 1 was shown above to help most: execution status (0.608→0.266 macro-F1) and error signature (0.421→0.217). Information sufficiency (0.711→0.681) and task completion prediction (0.714→0.699) degrade only slightly, indicating these two comparatively easier, near-binary signals are partly recoverable from the frozen embedding alone. This confirms that most of the structure in Table 2 depends on Stage 1's latent-dynamics objective, not merely on having a large enough labeled Stage 2 fine-tuning set.

These metrics measure predicted-state quality but do not guarantee improved agent behavior, which also depends on the inference harness. We evaluate downstream task execution next.

5 Inference and Agentic Task Execution

5.1 Benchmarks

We evaluate three transfer settings. **In-distribution**, EnterpriseOps-Gym (EOPS, 80 tasks) and CRMArena-Pro (CRM, 428 tasks) provide annotated operational states from trajectories generated on their training tasks for training Enterprise-JEPA. **Out-of-distribution**, WorkBench (Styles et al., 2024) (WB, 690 tasks) and AutomationBench (Shepard & Salimans, 2026) (AB, 400 tasks, excluding marketing and finance domains) assess transfer to unseen enterprise applications and tool schemas. **Out-of-domain**, Terminal-Bench 2.0 (Merrill et al., 2026) (TB, 89 tasks) assesses transfer from general enterprise operational states to a terminal-based task. The number of overall tasks is 1,687. Description of each benchmark is noted in Appendix A.

5.2 World-Model Inference Harnesses

We evaluate the following harnesses so that model quality is not conflated with a single strategy.

Revision. The base agent conducts a single-step lookahead operation: 1) An action is inferred by the agent; 2) The world model predicts the consequence of the action proposed by the base agent. The prediction is returned to the agent as pre-execution feedback, allowing it to retain or revise the action.

Imagine-Then-Plan. Imagine-Then-Plan (ITP-I) (Liu et al., 2026) alternates policy-proposed actions and world-model predictions for K imagined steps, then returns the trajectory to the agent for

reflection. All evaluated world models follow this procedure, differing only in the model supplying the predictions. Prompts appear in Appendix G.4.

Beam-Search Planning. Every K interaction steps, the policy generates B candidate tool-call sequences $A^{(b)} = (a^{(b)}_{t,1}, \ldots, a^{(b)}_{t,H_b})$, where $H_b \leq H$. We use *open-loop rollouts* to accelerate the planning: the policy generates each complete action sequence before the world model evaluates it, without conditioning subsequent actions on intermediate world-model predictions. This differs from closed-loop rollouts, which alternate action generation and world-model prediction at each imagined step. Complete plans are parsed and deduplicated; malformed calls and invalid tools are rejected, while recoverable schema errors receive soft penalties.

Enterprise-JEPA evaluates the candidate sequences using predictions of seven operational states:

$$r_{b,k} = \sum_{f \in \mathcal{F}_s} \alpha_f \sum_{y \in \mathcal{Y}_f} p_{b,k,f}(y) u_f(y), \qquad S(A^{(b)}) = \sum_{k=1}^{H_b} r_{b,k} - \Omega(A^{(b)}), \tag{5}$$

where Ω collects planner penalties. Let $b^\star = \arg\max_b S(A^{(b)})$. The planner recommends $a^{(b^\star)}_{t,1}$ only when its predicted first-step score exceeds that of the agent-proposed action by a margin δ; otherwise, the base action is retained. The selected trajectory, including states, is added to the policy context as advisory guidance, rather than committing the agent to execute the entire sequence. Thus, open-loop planning refers to candidate generation within imagined rollouts, not to execution without environmental feedback. Appendix D provides implementation details and hyperparameter settings.

5.3 Metrics and Baselines

Agentic task execution evaluates whether the world model improves task performance while minimizing computational overhead. We report the task success rate, defined as the fraction of evaluated tasks that are completed successfully.

For agentic task execution, we compare a no-WM baseline, in which the agent operates without access to a world model, with the three harnesses described in Section 5.2. Each harness is paired with two world models: Enterprise-JEPA and the state-output LLM-WM trained in Section 4. Because beam-search scoring requires a probability distribution for each operational field's categories, we convert the generated labels from the LLM-WM into one-hot distributions, assigning a probability of one to the generated category and zero to all other categories.

5.4 Implementation Details

All experiments use Qwen3.6-27B (Qwen Team, 2026) as the agent, with the serving configuration detailed in Appendix E.1. Beam-search planning uses $B = 8$ candidate plans and a rollout horizon of $H = 3$, and is invoked every two interaction steps. Appendix F.5 provides the empirical justification for these settings. We report mean scores and standard deviations across three runs.

5.5 Results

Success rate. Table 3 reports task success rates across world models and planning harnesses under three transfer settings. In the in-distribution setting, Enterprise-JEPA achieves the highest mean success rates on EnterpriseOps-Gym (0.446 with ITP-I) and CRMArena-Pro (0.374 with beam search), corresponding to relative improvements of 20.2% and 19.5% over the no-WM baseline, respectively.

The benefits also extend to unseen enterprise environments. On WorkBench, the state-output LLM-WM achieves 0.845 with revision, a 9.0% relative improvement over the baseline. On AutomationBench, Enterprise-JEPA with beam search achieves the highest mean success rate of 0.192, a 10.3% relative improvement. In the out-of-domain setting, Enterprise-JEPA with ITP-I reaches 0.345 on Terminal-Bench 2.0, exceeding the baseline of 0.258.

Within-harness comparisons show that Enterprise-JEPA achieves higher average scores than the LLM-WM under Revision (0.410 versus 0.408), ITP-I (0.417 versus 0.398), and beam search (0.418 versus 0.391). Under beam search, Enterprise-JEPA achieves higher mean scores on all five benchmarks. These results are consistent with its stronger next-state prediction performance and suggest

Table 3: **Task success rates across world models and inference harnesses (mean ± standard deviation over three runs).** Enterprise-JEPA achieves the highest mean on four of five benchmarks. With beam search, it exceeds both the no-world-model baseline and the state-output LLM-WM on all five benchmarks and achieves the highest average score.

World model	Harness	EOPS	CRM	WB	AB	TB	Ave.
No WM	Baseline	0.371 ± 0.006	0.313 ± 0.051	0.775 ± 0.017	0.174 ± 0.016	0.258 ± 0.000	0.378
State-output LLM-WM	Revision	0.392 ± 0.007	0.320 ± 0.030	$\mathbf{0.845} \pm 0.006$	0.172 ± 0.001	0.311 ± 0.034	0.408
	ITP-I	0.375 ± 0.037	0.315 ± 0.028	0.819 ± 0.006	0.167 ± 0.019	0.315 ± 0.030	0.398
	Beam search	0.371 ± 0.019	0.332 ± 0.028	0.779 ± 0.017	0.182 ± 0.015	0.281 ± 0.030	0.391
Enterprise-JEPA	Revision	0.392 ± 0.007	0.344 ± 0.041	0.842 ± 0.008	0.180 ± 0.019	0.292 ± 0.011	0.410
	ITP-I	$\mathbf{0.446} \pm 0.007$	0.336 ± 0.035	0.823 ± 0.008	0.136 ± 0.013	$\mathbf{0.345} \pm 0.039$	0.417
	Beam search	0.404 ± 0.007	$\mathbf{0.374} \pm 0.040$	0.799 ± 0.018	$\mathbf{0.192} \pm 0.007$	0.322 ± 0.026	**0.418**

that the choice of world model matters for downstream planning. These findings further indicate that agents use world-model feedback to improve task execution.

Overall, Enterprise-JEPA with revision and beam search both exceed the no-WM baseline in mean success rate on all five benchmarks. Beam search achieves Enterprise-JEPA's highest average score of 0.418, approximately a 10.6% relative improvement over the baseline. These findings support the utility of latent operational-state prediction across enterprise and terminal tasks.

Inference-time efficiency. World-model latency directly affects execution time because the agent must wait for candidate plans to be evaluated before issuing the next tool call. Figure 2 illustrates the latency of beam search with eight candidates and a rollout horizon h. We compare Enterprise-JEPA with the state-output LLM-WM. Assuming production serving, we use vLLM (Kwon et al., 2023) for the LLM-based world model and CUDA graphs for Enterprise-JEPA. The measured latency for Enterprise-JEPA ranges from 13– 35 milliseconds, while for the state-output LLM-WM, it is between 0.16–1.57 seconds. This corresponds to a speedup of up to $44.8\times$ at a horizon of 8. More details about the serving environment are described in Appendix E.1. To account for the different backends, Appendix F.2 additionally compares both models using HuggingFace Transformers (Wolf et al., 2020), where Enterprise-JEPA also retains a latency advantage.

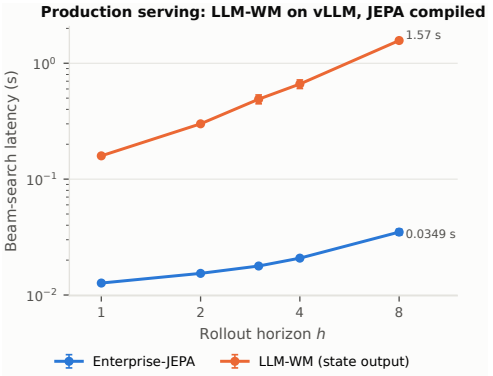


Figure 2: **Beam-search latency under optimized serving.** Enterprise-JEPA achieves up to a $44.8\times$ speedup over the LLM-WM.

Qualitative analysis. We compare assisted and baseline trajectories on tasks where Enterprise-JEPA improves the final-state score. We distinguish *read calls*, which retrieve information, from *write calls*, which modify the environment. Table 9 groups improvements by changes in write calls, with counts reported as Revision/ITP-I/beam search.

Most improvements involve correcting arguments or completing omitted actions. Correcting identifiers, email addresses, or field values accounts for 35/35/21 improved WorkBench tasks and 14/16/24 AutomationBench tasks. Completing batches, adding missing writes, or proceeding to a write after information retrieval accounts for another 31/31/22 and 10/16/8 improvements, respectively. Avoiding unnecessary writes accounts for 14/12/12 improvements on AutomationBench.

Many of these changes begin with better information retrieval. Read calls differ from the baseline in $67/85/51\%$ of improved WorkBench tasks and $93/98/87\%$ of improved AutomationBench tasks. The assisted agent performs missing lookups, reformulates queries, or ends redundant exploration before acting. For example, Appendix F.3 shows how a lookup prompted by reflection corrects the recipient of a subsequent `send_email` call.

The analysis also highlights the importance of verification and completion. On WorkBench, Revision corrects arguments in 35 tasks but changes previously correct arguments in 18, supporting guidance that encourages verification against retrieved information. On AutomationBench, cases

where the agent never performs a required write decrease from 22 and 20 under Revision and ITP-I to 4 under beam search. These findings motivate confidence-based filtering of advice and recommendations focused on gathering evidence and completing required actions.

5.6 Further Analysis and Comparisons

Contribution of world model predictions. To assess whether planning gains depend on world-model predictions rather than on candidate generation and reflection alone, we remove or shuffle the predicted-state feedback while retaining the planning procedure. Appendix F.4 details candidate selection without predictions and the shuffling procedure. Table 4 reports results on EnterpriseOps-Gym with eight candidate plans and a rollout horizon of three, showing mean scores and standard deviations across three runs. Enterprise-JEPA achieves a mean score of 0.404, compared with 0.342 for action-only trajectories without world-model feedback and 0.338 when predictions are shuffled across rollouts. Both controls perform below the no-WM baseline of 0.371. These comparisons suggest that informative world-model feedback contributes to the gains from beam-search planning.

Table 4: **Prediction controls on EnterpriseOps-Gym.** Enterprise-JEPA achieves a higher mean success rate than both removing and shuffling world-model feedback.

Variant	Success Rate
No-WM Baseline	0.371 ± 0.006
No world model feedback	0.342 ± 0.007
Shuffled predictions	0.338 ± 0.035
Enterprise-JEPA (beam search)	$\mathbf{0.404} \pm 0.007$

Comparison with a state-of-the-art world model. To assess whether a compact latent world model can match a substantially larger tool-output world model in downstream task execution, Table 5 compares Enterprise-JEPA (0.6B) with Qwen-AgentWorld (Zuo et al., 2026), a state-of-the-art 35B world model that generates predicted tool outputs. Enterprise-JEPA instead predicts latent observation representations and decodes operational states. Both models support the same planning harnesses and budgets. Enterprise-JEPA is trained, while Qwen-AgentWorld is applied zero-shot. For beam-search planning with Qwen-AgentWorld, the agent model acts as an LLM judge, evaluating the predicted tool outputs to select the best candidate trajectory.

Table 5: **Enterprise-JEPA versus Qwen-AgentWorld-35B-A3B.** Enterprise-JEPA achieves higher mean success rates across all harnesses on EnterpriseOps-Gym and broadly comparable performance on AutomationBench, despite its substantially smaller model size.

World model	Harness	EOPS	AB
No WM	Baseline	0.371 ± 0.006	0.174 ± 0.016
Qwen-AgentWorld (Tool-output LLM-WM)	Revision	0.375 ± 0.000	0.169 ± 0.015
	ITP-I	0.367 ± 0.007	0.151 ± 0.014
	Beam search	0.333 ± 0.014	0.181 ± 0.008
Enterprise-JEPA	Revision	0.392 ± 0.007	0.180 ± 0.019
	ITP-I	$\mathbf{0.446} \pm 0.007$	0.136 ± 0.013
	Beam search	0.404 ± 0.007	$\mathbf{0.192} \pm 0.007$

Despite its substantially smaller size, Enterprise-JEPA achieves competitive or superior task performance. On EnterpriseOps-Gym, it outperforms Qwen-AgentWorld by 7.9 percentage points with ITP-I. On AutomationBench, which is unseen during world-model training for both models, performance is comparable. These results demonstrate that a compact latent world model can compete with a state-of-the-art tool-output world model, with stronger performance in environments represented in its training distribution.

6 Conclusion

We introduced Enterprise-JEPA, an action-conditioned latent world model for enterprise tool-using agents without autoregressively generating future observations. Expanded latent pretraining improves structured next-state prediction, while downstream evaluations demonstrate better task performance at substantially lower planning latency than an LLM-based world model. These results demonstrate the effectiveness and efficiency of latent prediction for planning in enterprise tool-use environments.

REPRODUCIBILITY STATEMENT

We provide code and software dependencies in the anonymous GitHub repository. Implementation details (infrastructure, hyperparameters, LLM models, and fair-comparison settings) are described in Appendix E.1 and the source code. The prompts for benchmark construction and agents are provided in Appendix G. Additionally, the list of data sources is included in Appendix B.

AI USE STATEMENT

We used AI tools for grammar correction and word refinement to enhance the quality of the paper. We also leveraged Claude Code or Codex to implement the source code for executing experiments.

REFERENCES

Anthropic. Introducing claude opus 5. Blog Post, July 2026a. URL https://www.anthropic.com/news/claude-opus-5. Accessed: September 9, 2026.

Anthropic. Introducing claude sonnet 5. Blog Post, June 2026b. URL https://www.anthropic.com/news/claude-sonnet-5. Accessed: September 9, 2026.

Mahmoud Assran, Quentin Duval, Ishan Misra, Piotr Bojanowski, Pascal Vincent, Michael Rabbat, Yann LeCun, and Nicolas Ballas. Self-supervised learning from images with a joint-embedding predictive architecture. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

Mido Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Mojtaba, Komeili, Matthew Muckley, Ammar Rizvi, Claire Roberts, Koustuv Sinha, Artem Zholus, Sergio Arnaud, Abha Gejji, Ada Martin, Francois Robert Hogan, Daniel Dugas, Piotr Bojanowski, Vasil Khalidov, Patrick Labatut, Francisco Massa, Marc Szafraniec, Kapil Krishnakumar, Yong Li, Xiaodong Ma, Sarath Chandar, Franziska Meier, Yann LeCun, Michael Rabbat, and Nicolas Ballas. V-jepa 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint arXiv:2506.09985*, 2025.

Randall Balestriero and Yann LeCun. Lejepa: Provable and scalable self-supervised learning without the heuristics. *arXiv preprint arXiv:2511.08544*, 2025.

Léo Boisvert, Megh Thakkar, Maxime Gasse, Massimo Caccia, Thibault Le Sellier de Chezelles, Quentin Cappart, Nicolas Chapados, Alexandre Lacoste, and Alexandre Drouin. Workarena++: Towards compositional planning and reasoning-based common knowledge work tasks. In *Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie. Class-Balanced Loss Based on Effective Number of Samples . In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. Workarena: how capable are web agents at solving common knowledge work tasks? In *International Conference on Machine Learning*, 2024.

Joseph L. Fleiss. Measuring nominal scale agreement among many raters. *Psychological Bulletin*, 76(5):378–382, 1971.

Ross Girshick. Fast r-cnn. In *International Conference on Computer Vision*, 2015.

Yu Gu, Kai Zhang, Yuting Ning, Boyuan Zheng, Boyu Gou, Tianci Xue, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, Huan Sun, and Yu Su. Is your LLM secretly a world model of the internet? model-based planning for web agents. *Transactions on Machine Learning Research*, 2025.

Kilem Li Gwet. Computing inter-rater reliability and its variance in the presence of high agreement. *British Journal of Mathematical and Statistical Psychology*, 61(1):29–48, 2008.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Kung-Hsiang Huang, Akshara Prabhakar, Onkar Thorat, Divyansh Agarwal, Prafulla Kumar Choubey, Yixin Mao, Silvio Savarese, Caiming Xiong, and Chien-Sheng Wu. CRMArena-pro: Holistic assessment of LLM agents across diverse business scenarios and interactions. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=EPlpe3Fx1x.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Youwei Liu, Jian Wang, Hanlin Wang, Beichen Guo, and Wenjie Li. Imagine-then-plan: Agent learning from adaptive lookahead with world models. *arXiv preprint arXiv:2601.08955*, 2026.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Lucas Maes, Quentin Le Lidec, Damien Scieur, Yann LeCun, and Randall Balestriero. Leworld-model: Stable end-to-end joint-embedding predictive architecture from pixels. *arXiv preprint arXiv:2603.19312*, 2026.

Shiva Krishna Reddy Malay, Shravan Nayak, Jishnu Sethumadhavan Nair, Sagar Davasam, Aman Tiwari, Sathwik Tejaswi Madhusudhan, Sridhar Krishna Nemala, Srinivas Sunkara, and Sai Rajeswar. Enterpriseops-gym: Environments and evaluations for stateful agentic planning and tool use in enterprise settings. *DATA-FM Workshop at ICLR*, 2026.

Mike A Merrill, Alexander Glenn Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Kumar Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Kwesi Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Jenia Jitsev, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, 2026.

Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, Robert Berry, Saumya Malik, Saurabh Shah, Scott Geng, Shane

Arora, Shashank Gupta, Taira Anderson, Teng Xiao, Tyler Murray, Tyler Romero, Victoria Graf, Akari Asai, Akshita Bhagia, Alexander Wettig, Alisa Liu, Aman Rangapur, Chloe Anastasiades, Costa Huang, Dustin Schwenk, Harsh Trivedi, Ian Magnusson, Jaron Lochner, Jiacheng Liu, Lester James V. Miranda, Maarten Sap, Malia Morgan, Michael Schmitz, Michal Guerquin, Michael Wilson, Regan Huff, Ronan Le Bras, Rui Xin, Rulin Shao, Sam Skjonsberg, Shannon Zejiang Shen, Shuyue Stella Li, Tucker Wilde, Valentina Pyatkin, Will Merrill, Yapei Chang, Yuling Gu, Zhiyuan Zeng, Ashish Sabharwal, Luke Zettlemoyer, Pang Wei Koh, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. Olmo 3, 2025.

OpenAI. Gpt-5.1: A smarter, more conversational chatgpt. Blog Post, November 2025. URL https://openai.com/index/gpt-5-1/. Accessed: September 9, 2026.

OpenAI. Introducing gpt-5.4. Blog Post, March 2026a. URL https://openai.com/index/introducing-gpt-5-4/. Accessed: September 9, 2026.

OpenAI. Introducing gpt-5.5. Blog Post, April 2026b. URL https://openai.com/index/introducing-gpt-5-5/. Accessed: September 9, 2026.

OpenAI. Gpt5.6: Frontier intelligence that scales with your ambition. Blog Post, July 2026c. URL https://openai.com/index/gpt-5-6/. Accessed: September 9, 2026.

Shuofei Qiao, Runnan Fang, Ningyu Zhang, Yuqi Zhu, Xiang Chen, Shumin Deng, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Agent planning with world knowledge model. In *International Conference on Neural Information Processing Systems*, 2024.

Qwen Team. Qwen3.6-27b: Flagship-level coding in a 27b dense model. Blog Post, April 2026. URL https://qwen.ai/blog?id=qwen3.6-27b. Accessed: September 15, 2026.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. Identifying the risks of LM agents with an LM-emulated sandbox. In *International Conference on Learning Representations*, 2024.

Daniel Shepard and Robin Salimans. Automationbench. *arXiv preprint arXiv:2604.18934*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Neural Information Processing Systems*, 2023.

Vaishnavi Shrivastava, Piero Kauffmann, Ahmed Awadallah, and Dimitris Papailiopoulos. Echo: Terminal agents learn world models for free. *arXiv preprint arXiv:2605.24517*, 2026.

Yueqi Song, Ketan Ramaneti, Zaid Sheikh, Ziru Chen, Boyu Gou, Tianbao Xie, Yiheng Xu, Danyang Zhang, Apurva Gandhi, Fan Yang, Joseph Liu, Tianyue Ou, Zhihao Yuan, Frank F. Xu, Shuyan Zhou, Xingyao Wang, Xiang Yue, Tao Yu, Huan Sun, Yu Su, and Graham Neubig. Agent data protocol: Unifying datasets for diverse, effective fine-tuning of LLM agents. In *International Conference on Learning Representations*, 2026.

Olly Styles, Sam Miller, Patricio Cerda-Mardini, Tanaya Guha, Victor Sanchez, and Bertie Vidgen. Workbench: a benchmark dataset for agents in a realistic workplace setting. In *Conference on Language Modeling*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In Qun Liu and David Schlangen (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, 2020.

Zhangchen Xu, Adriana Meza Soria, Shawn Tan, Anurag Roy, Ashish Sunil Agrawal, Radha Poovendran, and Rameswar Panda. Toucan: Synthesizing 1.5m tool-agentic data from real-world mcp environments. *arXiv preprint arXiv:2510.01179*, 2025.

An Yang and et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Chen Yang, Ran Le, Yun Xing, Zhenwei An, Zongchao Chen, Wayne Xin Zhao, Yang Song, and Tao Zhang. Toolmind technical report: A large-scale, reasoning-enhanced tool-use dataset. *arXiv preprint arXiv:2511.15718*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R Narasimhan. {τ}-bench: A benchmark for \underline{T}ool-\underline{A}gent-\underline{U}ser interaction in real-world domains. In *International Conference on Learning Representations*, 2025.

Kai Zhang, Xiangchao Chen, Bo Liu, Tianci Xue, Zeyi Liao, Zhihan Liu, Xiyao Wang, Yuting Ning, Zhaorun Chen, Xiaohan Fu, Jian Xie, Yuxuan Sun, Boyu Gou, Qi Qi, Zihang Meng, Jianwei Yang, Ning Zhang, Xian Li, Ashish Shah, Dat Huynh, Hengduo Li, Zi Yang, Xuefei Cao, Lawrence Keunho Jang, Shuyan Zhou, Jiacheng Zhu, Huan Sun, Jason E Weston, Yu Su, and Yifan Wu. Agent learning via early experience. In *International Conference on Machine Learning*, 2026.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025.

Yuxin Zuo, Zikai Xiao, Li Sheng, Fei Huang, Jianhong Tu, Yuxuan Liu, Tianyi Tang, Xiaomeng Hu, Yang Su, Qingfeng Lan, Yantao Liu, Qin Zhu, Yinger Zhang, Bowen Yu, Haiquan Zhao, Haiyang Xu, Jianxin Yang, Jiayang Cheng, Junyang Wang, Lianghao Deng, Mingfeng Xue, Tianyi Bai, Yang Fan, Yubo Ma, Yucheng Li, Zeyu Cui, Zhihai Wang, Zhihui Xie, Zhuorui Ye, An Yang, Dayiheng Liu, Jingren Zhou, and Ning Ding. Qwen-agentworld: Language world models for general agents. *arXiv preprint arXiv:2606.24597*, 2026.

A Benchmark Details

- **EnterpriseOps-Gym (Malay et al., 2026).** 1150 enterprise tasks across 8 domains (Calendar, CSM, Drive, Email, HR, ITSM, Teams, and Hybrid) via 512 tools targeting 164 database tasks, requiring 9.15 steps on average. Among the publicly available 649 tasks for the oracle setting, we extracted 80 tasks (10 tasks for each domain) for evaluation, and the remaining are used to train the world model.
- **CRMArena-Pro (?).** Read-only analysis over a B2B CRM snapshot by using three tools (SQL execute, describe, and respond). Among 2140 tasks, we extracted 428 tasks, and the remaining are used to train the world model.
- **WorkBench (Styles et al., 2024).** Workplace tasks over Email, Calendar, Analytics, Project Management, CRM, and multi-domain with 26 read/write tools. We used all 690 tasks.
- **AutomationBench (Shepard & Salimans, 2026).** Cross-application business-workflow automation over a simulated SaaS world of 47 applications exposed through 549 Zapier-style tools (Salesforce, HubSpot, Google Sheets, Slack, Gmail, Airtable, DocuSign, etc.). Each task supplies a trigger message, an initial world state, and a per-task tool allow-list, and is scored by state-based assertions over the final world state, with partial credit for the fraction of assertions satisfied. If all the assertions are satisfied, the task is regarded as a success. Tasks are long-horizon, requiring 12.9 model steps and 26.5 tool calls on average in our baseline runs. Among the 600 publicly available tasks across 6 domains (Sales, Marketing, Operations, Support, Finance, and HR; 100 each), we use report results with 400 tasks excluding Marketing and Finance (400 tasks).
- **Terminal-Bench-2.0 (Merrill et al., 2026).** Terminal tasks executed in isolated Docker environments. We use all 89 tasks. The tool is a bash command.

B Trajectory Sources and Agent Data Protocol

We use the Agent Data Protocol (ADP) repository (`https://github.com/neulab/agent-data-protocol`) and its published `neulab/agent-data-collection` and `neulab/adp-v2` on Hugging Face. ADP converts heterogeneous raw logs through the Agent Trajectory Interchange Format (ATIF):

$$\text{raw dataset} \rightarrow \text{dataset-specific ATIF} \rightarrow \text{normalized ATIF} \rightarrow \text{world-model trajectory.}$$

The expanded Stage-1 setting uses native EnterpriseOps-Gym and CRMArena-Pro training trajectories with the nine ADP sources in Table 6.

C Annotation Protocol

Two-pass design. Stage-2 labels were produced by two passes over the same transitions. A first pass labeled every transition with a single rater, GPT-5.4-mini (OpenAI, 2026a). A second pass independently re-labeled the same transitions with three stronger raters—Claude Opus 5 (Anthropic, 2026a), Claude Sonnet 5 (Anthropic, 2026b), and GPT-5.6 (OpenAI, 2026c)—using the identical system prompt, user prompt, payload, and JSON schema. Holding the instructions fixed is what makes the raters comparable: the first-pass label then enters the analysis as an additional rater rather than as a reference truth, and is used only to break ties (below).

Annotation input. Each rater receives one JSON object per transition, containing the benchmark and trajectory identifiers, the system and task prompts, the last eight action–observation pairs, the previous state, the candidate action, and the observed tool output. So that the task-relative fields can be judged against what the episode actually required rather than from local evidence alone, the object additionally carries the whole episode as a list of (step, action, observation) triples of at most 30 steps, the steps that follow the candidate action, and the trajectory-level outcome (completion flag and verifier pass rate). Every field is truncated to 4,000 characters.

Table 6: Stage-1 trajectory sources for expanded corpus.

Source	#Trajectories	Group	Content and construction
EnterpriseOps-Gym	1,153	Native enterprise	Multi-model agent executions against live MCP tools in eight business domains; actual tool calls and returned observations are reconstructed and split by task ID.
CRMArena-Pro	3,362	Native enterprise	Multi-model SQL/response trajectories over the B2B CRM snapshot; task-disjoint train and held-out sets retain database observations and final answers.
TOUCAN (Xu et al., 2025)	38,226	ADP tool use	Enterprise-filtered MCP trajectories. The training subset contains the seeded 25k draw plus failure-bearing trajectories; reasoning-only pseudo-tools are collapsed, yielding 38.2k trajectories and a less skewed failure distribution.
Dolci (Olmo et al., 2025)	25,000	ADP tool use	Multi-turn synthetic function-calling conversations with explicit tool schemas, parallel calls, environment results, and final summaries; a seeded 25k-trajectory subset is used.
ToolMind (Yang et al., 2025)	163,180	ADP tool use	Synthetic reasoning-enhanced API trajectories with per-example tool definitions and environment messages.
SWE-smith	17,380	ADP SWE/OS	Agent trajectories for synthesized bug fixes in real Python repositories, including shell/editor actions and test output.
Code-Feedback	66,383	ADP SWE/OS	Multi-turn code generation, execution, feedback, and iterative repair using Python execution observations.
SWE-Play	704	ADP SWE/OS	OpenHands-style software-engineering trajectories with shell, Python, file-edit, reasoning, and completion actions.
SWE-Gym OpenHands	491	ADP SWE/OS	Sampled OpenHands trajectories solving executable GitHub tasks with repository state and unit-test feedback.
OpenHands Feedback	127	ADP SWE/OS	Recorded OpenHands coding and tool-use sessions containing environment feedback and human-guided recovery behavior.
AgentTuning OS	195	ADP SWE/OS	Operating-system interaction trajectories; terminal control sequences and prompts are removed while command outputs are preserved.

Prompt. The system prompt is

> "You are an expert trajectory labeler for enterprise agent world-model training. Your job is to label the observed effect of exactly one candidate action using only the provided categorical schema. Use the observed resulting state/tool output as evidence for the label. Do not solve the task, do not invent benchmark-specific fields, and do not write explanations. Return JSON only."

The user prompt concatenates an output contract, labeling guidance, and the payload. The output contract enumerates every field with its allowed values and requires a single JSON object. The guidance fixes the readings that are otherwise ambiguous:

- `execution_status` is whether the concrete action executed operationally, not whether the whole task is solved; trajectory-level success is not converted into an action-level status unless the action is itself a final evaluator or task wrapper.
- `progress_signal` is judged by the step's contribution to the task after seeing the observation. An action that returned cleanly but retrieved nothing useful, repeated an earlier step, or was later undone is neutral or negative.
- `information_sufficiency`, `information_gain`, and `missing_information_type` are judged against what the later steps reveal was needed: if a subsequent step had to look something up, that information was missing here.
- `risk_signal` marks confidentiality, policy, destructive, or irreversible risk visible from the action or observation.
- `recommended_abstract_action` is the best next behavior given the whole episode. `finalize` requires explicit completion evidence in the observation, and is not licensed merely because the trajectory stopped or a final response was emitted while errors remain.
- Later steps and the task outcome must *not* change `execution_status`, `error_signature`, `action_type`, `object_type`, or `side_effect_type`: a failed task can contain successful actions and a completed task can contain failed ones.
- Per-benchmark evidence rules: EnterpriseOps-Gym actions are typed tool calls whose return status is usually the best execution evidence; CRMArena-Pro actions are SQL, describe, and respond actions, where successful SQL execution is operational success even when the result set is empty.
- `unknown` is preferred only when the provided evidence is genuinely insufficient.

Inter-rater agreement. Table 7 reports, per field, the fraction of transitions on which all three raters agree, the fraction on which at least two agree, Fleiss' κ (Fleiss, 1971), and Gwet's AC1 (Gwet, 2008). We report AC1 alongside κ because κ is deflated by the severe class imbalance of these fields: `risk_signal` is approximately 96% a single category, giving $\kappa = 0.386$ against AC1 $= 0.953$ at 93.3% unanimity. The fields Enterprise-JEPA reports are the more reliable ones. `execution_status` and `error_signature` are close to deterministic ($\kappa = 0.957$ and 0.850), whereas the fields dropped from the reported set are exactly those the raters cannot agree on: `information_gain`, `recommended_abstract_action`, and `missing_information_type` all fall below $\kappa = 0.51$, so their achievable prediction accuracy is bounded by the labeling protocol rather than by the model. Although annotators show some agreement on `object_type`, this field is largely inferable from the tool name and arguments and offers limited utility for beam-search planning compared with `action_type`. We therefore exclude it from the operational-state schema in Table 1.

Table 7: Inter-rater agreement over the three ensemble annotators on the training split . Upper block: fields reported in Table 2. Lower block: dropped fields.

Field	Unanimous	Majority	Fleiss κ	Gwet AC1
execution_status	0.970	0.999	0.957	0.977
error_signature	0.883	0.997	0.850	0.916
progress_signal	0.841	0.991	0.806	0.866
action_type	0.767	0.989	0.728	0.830
information_sufficiency	0.721	1.000	0.564	0.677
side_effect_type	0.684	0.997	0.677	0.774
risk_signal	0.933	0.997	0.386	0.953
object_type	0.698	0.975	0.702	0.780
missing_information_type	0.504	0.836	0.499	0.610
recommended_abstract_action	0.488	0.923	0.504	0.595
information_gain	0.391	0.980	0.395	0.503

Label resolution. Each field is resolved independently by the following ladder, and the rule that fired is recorded per field so that any downstream analysis can exclude tie-broken values. (i) *Unanimous*: all three raters agree. (ii) *Majority*: exactly two agree. (iii) *Tie-break*: on a three-way split, the first-pass GPT-5.4-mini label acts as a fourth rater, and a value it shares with one of the three becomes a two-of-four majority. (iv) *No majority*: a genuine four-way split keeps the first-pass label and is flagged. Across the ensemble-labeled training transitions and eleven annotated fields, 71.8% of field values are unanimous, 25.3% are two-of-three majorities, 1.5% are tie-broken, and 1.3% have no majority.

Automatic filtering. Before training, we discard uninformative labels in which every state field is `unknown`, and deduplicate records with identical (system prompt, task prompt, action, input history, previous state), keeping the member with the fewest `unknown` fields. Over the full labeled pool, this removes 243 uninformative and 148 duplicate training records from 18,149, and 57 and 73 respectively from 6,410 evaluation records, with no leakage detected; restricting the result to the two in-distribution benchmarks gives the $17{,}066$ and $5{,}854$ pairs used here.

Task Completion label. The `task_completion` field is not annotated. It is derived deterministically as whether a transition is the last interaction index of its trajectory that leads to succeeding the task, so it carries no rater disagreement and does not appear in Table 7.

D Planning Details

Candidate generation and latent rollout. At a planning event, the action model generates B plans $b = (a_{b,1}, \ldots, a_{b,H})$. Open-loop generation emits complete plans in one batched request; closed-loop generation alternates action generation and WM prediction. Malformed JSON, unknown tool names, and non-object argument payloads are rejected. Missing required arguments and type mismatches are retained as soft schema warnings because plans are advisory. Enterprise-JEPA

then predicts a categorical distribution $p_{b,k,f}$ for every step k and state field f in one batched latent rollout.

Core expected-utility score. For category utility $u_f(c)$ and field weight w_f, the unregularized step and trajectory scores are

$$r_{b,k} = \sum_f w_f \sum_{c \in \mathcal{C}_f} p_{b,k,f}(c) u_f(c), \qquad S_b^{(0)} = \sum_{k=1}^{H_b} r_{b,k}. \tag{6}$$

No temporal discount is applied. The active beam configuration uses weights $1.0, 1.2, 0.5, 0.4, 0.6$ for execution, progress, information sufficiency, error, and side effect, respectively. Success and positive progress have utility $+1$; failure, negative progress, and concrete errors have utility -1. Neutral progress has utility -0.3. Intended creation, modification, execution, installation, retrieval, and validation are neutral rather than penalized merely for changing state. Failed validation, externally visible sending, and deletion have utilities $-0.5, -0.3, -1$, respectively.

Scoring trajectory. The score used to rank a generated plan is

$$\begin{aligned}\widetilde{S}_b = S_b^{(0)} + \lambda_1 r_{b,1} - \lambda_R \sum_{k=1}^{H_b} q_{b,k}^R \rho_{b,k} + \lambda_C \frac{|\mathcal{R} \cap \mathcal{A}_b|}{|\mathcal{R}|} + \lambda_F \mathbb{I}[\mathcal{R} \cap \mathcal{A}_{b,1} \neq \varnothing] \\ - \lambda_M N_b^{\text{miss}} - \lambda_T N_b^{\text{type}} - \lambda_D C_b.\end{aligned} \tag{7}$$

Here $q_{b,k}^R = p(\texttt{read}) + p(\texttt{search})$ from the action-type head. The read pressure is

$$\rho_{b,k} = \max\left\{\mathbb{I}[n_{b,k}^R \geq K],\ p_{b,k}(\texttt{sufficient}),\ \eta \mathbb{I}[n_{b,k}^P > 0]\right\}, \tag{8}$$

where n^R counts distinct consecutive successful reads, n^P counts prior successful non-read actions, $K = 2$, and $\eta = 0.5$. Thus, a read is penalized only after evidence has accumulated, information is predicted to be sufficient, or task-changing progress has already occurred. $\mathcal{R}$ is the benchmark-agnostic set of operation types explicitly requested in the task text (create, update, delete, run, test, communicate, or validate), and $\mathcal{A}_b$ contains predicted action types in the plan. N_b^{miss} and N_b^{type} count soft schema warnings. C_b counts how often the first-step tool signature has recently been selected, discouraging repeated replans with the same first move.

The hyperparameters are $\lambda_1 = 0.5$, $\lambda_R = 1.0$, $\lambda_C = 1.0$, $\lambda_F = 0.75$, $\lambda_M = 0.15$, $\lambda_T = 0.25$, and $\lambda_D = 0.75$. If no required operation is identified, the coverage term is zero.

Veto and acceptance. A trajectory is removed if any step has $p(\texttt{failure}) > 0.6$ or $p(\texttt{deleted}) > 0.5$. Among surviving candidates, probabilities are normalized as $\pi_b = \exp(\widetilde{S}_b) / \sum_{j \in \mathcal{V}} \exp(\widetilde{S}_j)$. The winning first step must exceed the similarly adjusted base proposal by margin δ. In either rollout mode, acceptance additionally requires a complete horizon and a mean normalized score of at least $1.5/B$. In advisory mode, the accepted trajectory is inserted into the policy context; revision asks the policy to arbitrate at each real step.

E Experimental Protocol

E.1 Implementation Details

Architecture. We use Qwen3-Embedding-0.6B as the text encoder and pass its final-token representation to the projector. The projector consists of a linear layer with input and output dimensions of 1024, followed by layer normalization. The predictor comprises eight Transformer blocks with adaptive layer normalization, 16 attention heads, and a hidden dimension of 1024. Its input sequence consists of nine tokens: eight tokens representing recent tool outputs and one context token. The seven classification heads share a linear layer that maps the predictor's hidden state to 512 dimensions, followed by a GELU (Hendrycks & Gimpel, 2016) activation and dropout. Each head then applies a separate linear layer to produce logits over its corresponding categories.

Stage 1 training hyperparameters. We train the final four layers of the text encoder, the projector, and the predictor, while keeping the remaining encoder layers frozen. We use AdamW (Loshchilov & Hutter, 2019) with learning rates of 10^{-5} for the projector and predictor and 10^{-6} for the text encoder, with a weight decay of 0.01. The learning rates follow a cosine decay schedule with an initial warmup period. We apply dropout with a probability of 0.1 to the predictor and clip the gradient norm to 1.0. Training uses eight H200 NVIDIA GPUs with a per-GPU batch size of eight and BF16 mixed precision. The maximum input length of the text encoder is 8,192 tokens.

Stage 2 training hyperparameters. We train the shared classification layer and the seven field-specific output layers while freezing the text encoder, projector, and predictor. We use AdamW with a learning rate of 5×10^{-4} and a weight decay of 0.01, following a cosine decay schedule with an initial warmup period. The classification module uses a dropout probability of 0.1. Training uses two L40S GPUs with a per-GPU batch size of 24 and BF16 mixed precision. We optimize a class-balanced cross-entropy loss, with field-specific class weights computed from the effective number of samples using $\beta = 0.9999$.

Serving environment. All agents and served world models run on a single NVIDIA H200 NVL (140 GiB HBM, driver 595.71.05) under vLLM 0.20.0 with PyTorch 2.11.0+cu130 (CUDA 13.0). The policy is Qwen3.6-27B, served with tensor parallelism 1 on one GPU. The context length is 200k tokens, FP8 KV cache at 92% of device memory, prefix caching and chunked prefill on, scheduler admitting up to 256 concurrent sequences and 8192 batched tokens per step. Thinking mode is disabled through the chat template, so latency reflects tool-use decoding rather than chain-of-thought; automatic tool choice with the Qwen3-coder parser because the ReAct executors need structured tool calls recovered from the output.

LLM models. All open-source models are sourced from the HuggingFace's `transformers` library (Wolf et al., 2020), Table 8 lists the API names of closed-source models and the HuggingFace model names of open-source models.

Table 8: List of LLM model names in the experiments

Model Name	API name or Huggingface model name
Qwen3-0.6B	`Qwen/Qwen3-0.6B`
Qwen3-8B	`Qwen/Qwen3-8B`
Qwen3.6-27B	`Qwen/Qwen3.6-27B`
Qwen3-Embedding-0.6B	`Qwen/Qwen3-Embedding-0.6B`
Qwen-AgentWorld-35B-A3B	`Qwen/Qwen-AgentWorld-35B-A3B`
GPT-5.1	`gpt-5.1-2025-11-13`
GPT-5.4-mini	`gpt-5.4-mini-2026-03-17`
GPT-5.5	`gpt-5.5-2026-04-23`
GPT-5.6	`gpt-5.6-sol`
Claude Sonnet 5	`claude-sonnet-5`
Claude Opus 5	`claude-opus-5`

F RESULTS

F.1 NEXT STATE PREDICTION

Figure 3 reports per-category recall on 5,854 transitions, covering 39 categories across seven operational fields. Categories are ordered by their frequency within each field. Expanded pretraining primarily improves less frequent outcomes: mean recall for categories representing less than 5% of their field increases from 0.380 to 0.459, while recall for categories representing at least 20% changes only slightly, from 0.792 to 0.798. Notable gains include partial execution (+0.234), not-found errors (+0.201), and object creation (+0.134).

Compared with the state-output LLM-WM, Enterprise-JEPA achieves substantially higher recall on infrequent categories (0.459 versus 0.284), while remaining comparable on frequent cate-

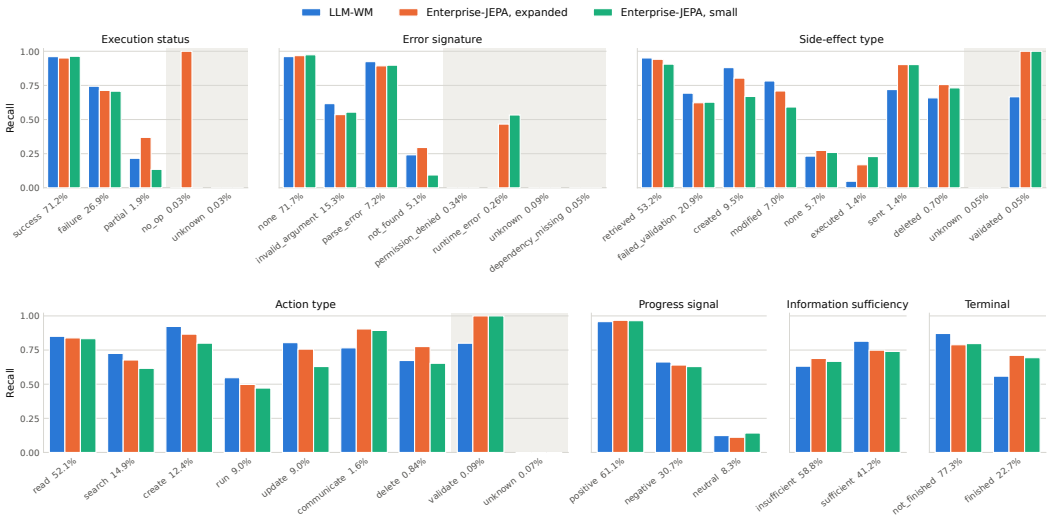


Figure 3: **Per-category recall across operational-state fields.** Expanded pretraining improves Enterprise-JEPA's recall for several less frequent categories, particularly in execution status and error signature, although gains vary across fields and categories.

gories (0.798 versus 0.805). Failure detection remains an important exception: recall is 0.713 for Enterprise-JEPA versus 0.744 for the LLM-WM, motivating further evaluation of the failure probabilities used by the planner's veto. Overall, Enterprise-JEPA improves coverage of infrequent operational outcomes while reducing inference cost (Section F.2).

F.2 Inference Time Efficiency

In the main text, the LLM world model is served with vLLM (Kwon et al., 2023), whereas Enterprise-JEPA uses CUDA graphs with bucketed input shapes. Because Enterprise-JEPA's custom architecture, which combines an embedding encoder with a latent predictor and classification heads, is not directly supported by vLLM, the two models use different optimized backends. To provide a backend-controlled comparison, we additionally run both models with Hugging Face Transformers (Wolf et al., 2020) on the same hardware and without backend-specific optimizations. Figure 4 shows that Enterprise-JEPA requires 0.05–0.08 s per planning step for horizons $h = 1$–8, compared with 3.0–12.1 s for the state-output LLM-WM, speeding up to $158\times$. Although this setting does not reflect the most efficient deployment of the LLM-WM, it isolates the latency difference arising from latent prediction rather than from the serving backend.

F.3 Qualitative Analysis

Each example pairs the baseline and world-model runs on the same task; only the differing calls are shown.

Corrected write arguments (WorkBench `email_0022`, Revision, 0.0→1.0). Task: "send an email to dmitri saying 'Hey dmitri, …' and title it 'Update on performance evaluation'". Baseline: a single `send_email(recipient="dmitri@example.com", ...)` to a guessed address. World model: `find_email_address("dmitri")` followed by `send_email(recipient="dmitri.ivanov@atlas.com", ...)`; subject and body are byte-identical, and the side-effect flag clears. The world model's own record predicted the wrong call would succeed (`execution_status: success`, not vetoed); the fix came from the agent inserting a lookup after being prompted to reflect. The same class on EnterpriseOps: `create_group.members_odata_bind` gains the owner `james.wilson` omitted in all three baseline runs.

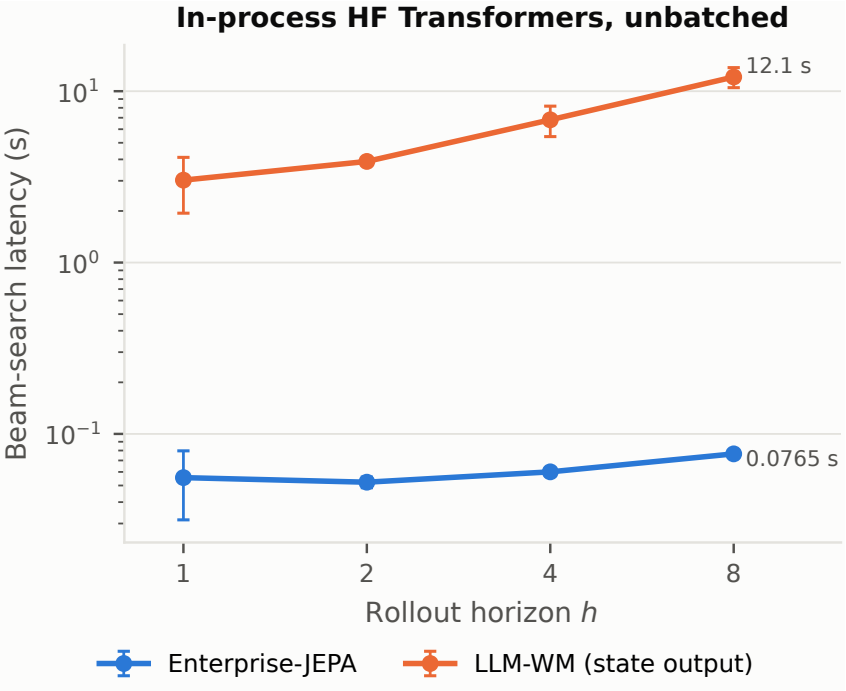


Figure 4: **Beam-search latency versus rollout horizon using Hugging Face Transformers on the same hardware.** Enterprise-JEPA remains below 0.1 s across the tested horizons, while the state-output LLM-WM requires 3.0–12.1 s, demonstrating a substantial latency advantage under a shared inference backend.

Completed write batch (EnterpriseOps *drive* task, ITP-I, 0→1). Baseline writes `create_permission`×1, `create_replies`×1; world model `create_permission`×2, `create_replies`×1. Verifier repaired: "Check if permissions have been granted". On WorkBench, the same class completes "delete all emails from amir, last 6 days": 1 delete → 3 (ground truth 3).

Added a missing write (AutomationBench `sales.full_sales_cycle_orchestrator`, Beam search, 0.25→0.75). The baseline schedules the calendar event but never creates the meeting it refers to. The world-model run issues the same five writes plus `zoom_create_meeting(topic="TechVentures – Technical Q&A", duration=60, start_time="2026-01-16T15:00:00Z", ...)`; scored assertions 2/8 → 6/8.

Acted where the baseline only read (WorkBench `email_0041`, ITP-I, 0.0→1.0). Task: "Forward all emails from Anaya sent last week about 'Update on Board of Directors Conclave' to Nadia." The baseline performs two directory lookups and two searches before terminating without a write. With world-model guidance, the agent invokes `forward_email` for email IDs `00000120` and `00000346`, using Nadia's verified address as the recipient, exactly matching the ground-truth actions.

Removed spurious write (AutomationBench `sales.negative_selection`, Revision, 0.46→1.00). Baseline `salesforce_contact_add_to_campaign`×12; world model ×6. The task is a negative-selection filter: the six extra contacts each broke a pre-satisfied assertion, which the benchmark's free-assertion rule converts into scored failures (6/13 → 7/7 scored).

Reordered writes (EnterpriseOps *csm* task, ITP-I, 0→1). Baseline `update_contract`, `update_contract`, `update_user_details`; world model `update_contract`,

Table 9: How Enterprise-JEPA improves task outcomes. Each entry reports the number of improved tasks attributed to a change in state-modifying tool calls, shown as Revision / ITP-I / beam-search counts.

Help mechanism	WB	AB	EOPS	Description
Corrected mutation arguments	35/35/21	14/16/24	1/5/1	The agent invokes the same state-modifying tool as the baseline but corrects one or more arguments, such as an identifier, address, or field value. The correction often follows an additional or revised lookup.
Completed write batch	15/13/12	2/7/4	0/2/1	The agent applies the required state-modifying operation to all relevant items, whereas the baseline updates only a subset of them.
Acted where baseline only read	13/17/8	0/2/0	–	The baseline performs only search or lookup operations and terminates, whereas the agent proceeds to execute the required state-modifying actions.
Removed spurious mutation	4/1/1	14/12/12	–	The agent omits unnecessary state-modifying calls made by the baseline, such as actions on items that should have been excluded by a task constraint.
Added a missing mutation	3/1/2	8/7/4	1/0/0	The agent invokes a required state-modifying tool that the baseline never calls, causing the corresponding task condition to be satisfied.
Reordered mutations	–	2/2/1	0/1/0	The agent executes the same set of state-modifying calls as the baseline but changes their order to satisfy dependencies between operations.

`update_user_details`, `update_contract`. Identical tool multiset; the "Verify User location" verifier passes.

Baseline crash avoided (Terminal-Bench-2.0 `crack-7z-hash`, ITP-I, fail→pass). The baseline's model emitted a password literal long enough to break its own JSON exec envelope after 30 commands ("could not parse a JSON object from ..."); the world-model run passed in 17 commands. Three of ITP-I's ten Terminal-Bench gains and three of Beam search's seven follow this pattern.

Category gain (CRMArena-Pro task 795, `sales_amount_understanding`, Beam search, 0→1). Query: "During the 2020 Winter season, which representative achieved the maximum total sales? Return only the ID of the agent." Baseline answer `CompanySignedDate` (a field name); world model `005Wt000003NJ53IAG`, the correct ID.

F.4 Details on Prediction Control

Without predicted states, candidate scores are tied, and the planner's stable sort preserves their generation order. The first candidate sampled by the policy is therefore selected; selection is not random. In the shuffled control, complete prediction records are permuted across candidate plans and rollout steps before scoring and feedback rendering. Consequently, the substituted predictions determine both candidate ranking and the predicted-state feedback shown to the policy, including terminal advice. These components move together, preserving their within-record relationships; individual fields are never shuffled independently.

F.5 Ablation Study

Planning budget. Table 10 examines the number of candidate plans, rollout horizon, and rollout strategy, varying one factor at a time from our default beam-search configuration of eight candidates, horizon three, and open-loop rollouts.

Increasing the number of candidates from eight to sixteen reduces performance by 2.9% points on EnterpriseOps-Gym and 1.0% on AutomationBench. Using four candidates also reduces performance on EnterpriseOps-Gym, while yielding a slightly higher mean on AutomationBench. Longer rollouts provide no additional benefit: increasing the horizon from three to four reduces performance by 5.8 and 0.4%, respectively, whereas horizon two produces similar results. These trends may reflect errors in predicted scores: more candidates increase the opportunity to select an overly optimistic prediction, while longer rollouts may accumulate prediction errors.

In open-loop rollouts, the policy generates each complete action sequence before the world model evaluates it. In closed-loop rollouts, action sampling and world-model prediction alternate: each subsequent action is conditioned on the predicted outcomes of earlier actions, until the rollout horizon is reached. Compared with open-loop rollouts, closed-loop rollouts reduce performance by 2.9% on EnterpriseOps-Gym and 0.6% on AutomationBench, while increasing latency per executed step by 23% and 83%, respectively. Conditioning action generation on intermediate predictions would propagate world-model errors into later action choices.

Table 10: **Planning-budget ablation.** Increasing the candidate count or rollout horizon beyond the default (bold) lowers mean performance on both benchmarks, as does switching to closed-loop rollouts. The default achieves the highest mean on EnterpriseOps-Gym and remains close to the best result on AutomationBench.

Configuration			**Success Rate**	
Cand.	Hor.	Rollout	EnterpriseOps-Gym	AutomationBench
4	3	open	0.367 ± 0.007	0.191 ± 0.001
8	**3**	**open**	$\mathbf{0.404 \pm 0.007}$	$\mathbf{0.192 \pm 0.007}$
16	3	open	0.375 ± 0.013	0.182 ± 0.008
8	2	open	0.392 ± 0.026	0.188 ± 0.007
8	4	open	0.346 ± 0.029	0.188 ± 0.018
8	3	closed	0.375 ± 0.013	0.186 ± 0.002

We therefore use eight candidates, horizon three, and open-loop rollouts, which achieve the highest mean score on EnterpriseOps-Gym and AutomationBench.

G Prompts

This section collects every prompt used in the experiments. The annotation prompt is given separately in Appendix C. Braced placeholders are substituted at runtime. Overlong lines are wrapped for typesetting; the prompt text is otherwise unmodified.

G.1 Acting policy

The acting policy keeps each benchmark's native tool-calling prompt unchanged in every condition, including the no-WM baseline. No harness replaces or rewrites that prompt: each one appends a block to the policy's context and lets the policy emit the real action through its normal tool-calling interface. The prompts below are therefore the complete difference between the baseline and each world-model condition.

G.2 State-output LLM world model

Fine-tuning and next-state evaluation. The state-output LLM-WM is trained and evaluated (Table 2) with the following system prompt, which enumerates the seven fields and their categories in a fixed key order:

```
You are an enterprise world model. Given the system prompt, user task,
recent
history, and the current action, predict the resulting annotated state.
Return ONLY a JSON object with exactly these keys, in this order, and no
other text:
```

```
- action_type: one of [clarify, communicate, create, delete, read, run,
search,
  test, unknown, update, validate]
- execution_status: one of [failure, no_op, partial, success, unknown]
- progress_signal: one of [negative, neutral, positive, unknown]
- information_sufficiency: one of [insufficient, sufficient]
- error_signature: one of [dependency_missing, invalid_argument, none,
not_found,
  parse_error, permission_denied, policy_risk, runtime_error,
  test_failed,
  timeout, unknown]
- side_effect_type: one of [created, deleted, executed,
failed_validation,
  installed, modified, none, retrieved, sent, unknown, validated]
- terminal: one of [not_finished, finished]; use finished only when this
action is
  the final step that completes or ends the task
```

The user turn is

```
System prompt:
{system_prompt}

User prompt:
{task_prompt}

Recent action/observation history (oldest to newest):
{history_text}

Action:
{action_json}

Predict the annotated state as JSON. /no_think
```

G.3 REVISION

The predicted state of the policy's own proposed action is returned before execution:

```
[WM_ACTION_REVISION]
The world model evaluated your proposed action before execution:
{predicted_rollout}

Decide whether to PROCEED with the same action or REVISE it. Return the
actual
next action using the normal tool-call format. Treat the prediction as
uncertain
guidance, not an observed result.
[/WM_ACTION_REVISION]
```

`predicted_rollout` is produced by the renderer shared with beam-search planning (Section G.5), instantiated on the single proposed action.

G.4 IMAGINE-THEN-PLAN

Lookahead decision. The policy first chooses the lookahead depth K.

```
[system] Choose how many future interaction steps a world model should
imagine
before the next real action. Harder, uncertain, or risky states merit
more
lookahead. Return exactly one integer from 0 through {max_k}, with no
other text.

[user] Task:
```

```
{task}

Current interaction history:
{history}
```

Imagination (generative world model). With a generative world model, the whole K-step trajectory is requested in one call.

```
[system] You are a world model simulating an interactive tool
environment. Predict
plausible future agent actions and the environment observations they
cause. Do not
claim these imagined actions have actually happened.

[user] Task:
{task}

Available tool names:
{tool_names}

Observed interaction history:
{history}

Imagine the next {k} action-observation steps. Number the steps, include
important
arguments and likely outputs/errors, and remain concise.
```

Hypothetical action proposal (Enterprise-JEPA). Enterprise-JEPA predicts a next state only once an action is supplied, so the policy and world model alternate for K steps. At each imagined step, the policy is asked for one hypothetical action, conditioned on the imagined prefix rendered by Section G.5.

```
[system] Propose a hypothetical tool-action trajectory for a world model
to
simulate. Return only a JSON array. Do not claim any action was executed.

[user] Task:
{task}

Available tools (JSON):
{tools_json}

Observed interaction history:
{history}

Imagined trajectory so far:
{imagined_prefix}

Propose hypothetical action {step} of {k}. Return exactly one JSON
object:
{"name":"<tool>","arguments":{...}}. Use only available tools. Do not
invent
identifiers that are absent from the real or imagined state.
```

Reflection. Either way, the imagined trajectory is returned to the policy as implicit feedback:

```
[ITP_I_IMPLICIT_WORLD_MODEL_FEEDBACK]
Adaptive lookahead: K={k}
Imagined trajectory (hypothetical, not executed):
{foresight}

Reflect on the task, the real interaction history, and this hypothetical
trajectory. Identify progress, risks, contradictions, and the next
bottleneck.
```

```
Then choose the next real action using the available tools. Do not
blindly copy
the first imagined action, and do not treat imagined outputs as observed
facts.
[/ITP_I_IMPLICIT_WORLD_MODEL_FEEDBACK]
```

When $K = 0$, the imagined-trajectory slot reads "No lookahead was requested for this state."

G.5 BEAM-SEARCH PLANNING

Candidate generation. One plan is requested per sample; the same prompt is sampled B times so, a multi-sample endpoint can share its prefill across candidates.

```
You are generating ONE candidate action trajectory for beam search. From
the
CURRENT STATE, propose a sequence of up to {n} tool calls to try next.

OUTPUT FORMAT:
Return ONLY a JSON array (no prose or markdown fences) of up to {n}
steps.
Example: [{"name":"toolA","arguments":{"x":1}},
          {"name":"toolB","arguments":{"id":"$step1.id"}}]
Each step is {"name": "<tool>", "arguments": {...}}.

Rules:
- For any argument whose value depends on an earlier step's result, use a
symbolic
  reference string like "$step1.field" instead of guessing a concrete
  value.
- If a lookup can return multiple records, add a bind selector on that
step and
  reference it with "$vars.name"; for example
  {"name":"find_user","arguments":
  {"name":"Ethan
  Well"},"bind":{"ethan_user_id":{"field":"sys_id","match":
  {"name":"Ethan Well"}}}} followed by
  {"name":"assign_incident","arguments":
  {"user_id":"$vars.ethan_user_id"}}.
- Do not list alternatives inside the plan. Return only ONE single plan.
- The selected diversity option must materially affect the first useful
action,
  subgoal order, or information-gathering strategy; a paraphrase does not
  count as
  a different trajectory.
- Do not invent entity IDs, record identifiers, or tool arguments merely
to appear
  different.
- If the selected diversity option is impossible, use the closest
feasible
  interpretation while preserving task correctness.
- Use ONLY these tools: {tool_names}

CONTEXT:
{context}

CURRENT STATE:
{state}

DIVERSITY PROCEDURE:
- Select exactly one option from 1-{B} before planning. Sample among the
options
  rather than always choosing option 1.
- Use the selected option to shape the trajectory, but output only the
JSON plan.
```

```
DIVERSITY OPTIONS:
{diversity_options}

JSON array of up to {n} steps:
```

With $B = 8$ the diversity menu is

```
1. best_overall: Use the most direct, high-confidence action sequence
that covers
   the task requirements.
2. different_first_tool: Start with a materially different useful tool
from the
   most obvious first action, then pursue a valid route to the same goal.
3. read_or_verify_first: Start by reading, searching, listing, or
verifying state
   before making a mutation, if that is plausible for the task.
4. write_or_update_first: If the current state contains the required
identifiers,
   start with a justified create, update, assign, or other
   progress-making
   mutation.
5. alternate_entity_order: Handle entities or subrequirements in a
different order
   from the most obvious plan while still aiming to finish the same task.
6. minimal_plan: Use the shortest viable plan. Avoid redundant
verification unless
   the state is ambiguous.
7. recovery_plan: Use a robust recovery-oriented plan that accounts for
missing
   IDs, lookup failures, or partial prior progress before proceeding.
8. dependency_first: Resolve the most important prerequisite or
dependency first,
   then use its result through symbolic references in later actions.
```

Advisory injection. The selected trajectory and its predicted states are inserted into the policy context by the renderer shared with the revision harness and with ITP-I's imagined prefix:

```
World-model imagined rollout for the next steps (predicted action ->
predicted
resulting state). Use it as guidance, not ground truth:
  step 1 action: {tool}({arguments})
  step 1 predicted state: execution_status=..., progress_signal=...,
    side_effect_type=..., error_signature=...,
    information_sufficiency=...,
    terminal_probability=0.NN
  step 2 action: ...
Guidance: these predicted states are the world model's guesses, NOT
confirmed
results. Do NOT give a final answer or give up prematurely. A predicted
'finalize'
only means the task MAY be near done -- first verify from the actual
state that
every requirement is met. If you are blocked, uncertain, or a step
failed, examine
the current state and try a different action or approach before
concluding.
```

Fields whose predicted value is uninformative (`none` or `unknown`) are omitted from the state line, and `terminal_probability` appears only for a checkpoint with a termination head. The renderer emits one `action`/`predicted state` pair per horizon step; the revision harness instantiates it with the single proposed action.