TEXTUAL PLANNING WITH EXPLICIT LATENT TRANSITIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Planning requires a transition model that predicts how each action changes the current state. When a large language model (LLM) plays this role, every next state is generated token by token, which makes searching over many possible futures slow and expensive. Existing alternatives either still query an LLM at every step or require a symbolic model of the domain. We propose EMBEDPLAN, a transition model built on frozen text embeddings: it embeds natural language descriptions of the state and the action with a frozen LLM, predicts the embedding of the next state with a lightweight learned network, and returns the closest real state. Because this network can be trained on top of any encoder, EMBEDPLAN also provides a controlled way to compare text representations for learning transitions. We evaluate it on 9 classical planning domains, under six settings that hold out progressively more of the data, from transitions to entire domains, and against baselines ranging from predicting no change to learning symbolic action rules. On planning problems seen during training, EMBEDPLAN almost always ranks the true next state among its top five guesses, still does so for most queries even when every observed state is a candidate, and retains 92–99% of its single-step accuracy when predicting several steps ahead from its own outputs. Given the same candidate states as GPT-5.4, it picks the true next state more often while taking about 0.17 ms per transition with cached embeddings. Accuracy is lower on unseen problems and near chance on unseen domains, and the controlled comparison traces this limit to the state representation rather than to the learned transition.

1 INTRODUCTION

When a large language model (LLM) serves as the world model of a planner, each next state is generated token by token, with a full forward pass per token. This makes multi-step lookahead and rollout-based search, which robust long-horizon planning requires, prohibitively expensive in latency and cost. Without a compact transition function that predicts how actions transform states, planners cannot cheaply evaluate alternatives or backtrack from errors.

Prior work avoids this cost only in part. Prompting-based reasoning (Wei et al., 2022; Yao et al., 2023a;b) still queries the LLM at every step. Compilation to the Planning Domain Definition Language (PDDL) (Liu et al., 2023; Guan et al., 2023; Oswald et al., 2024; Tantakoun et al., 2025; Zuo et al., 2025) requires a domain that can be expressed as a symbolic model. Latent world models (Ha & Schmidhuber, 2018; Hafner et al., 2020; Schrittwieser et al., 2020) learn dynamics over perceptual or game observations rather than text. Whether frozen pre-trained language embeddings can host a learned transition function, turning each planning step into a cheap vector operation, has not been systematically tested.

We introduce EMBEDPLAN, which learns explicit transition dynamics over frozen LLM embeddings of text-described planning domains (Figure 1). It encodes a natural language state description (e.g., "Block A is on B, Block C is clear") and an action (e.g., `pick-up(C)`) with a frozen LLM. A small learned head projects both into a low-dimensional space and predicts the next-state embedding there, and the nearest real state is retrieved as the successor. Two contrastive objectives train it: state prediction, which identifies the correct next state among candidates, and action disambiguation, which separates the effects of different actions on the same state. Semantic encoding thus stays with the frozen LLM, and only the cheap dynamics are learned. EMBEDPLAN is a transition component,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

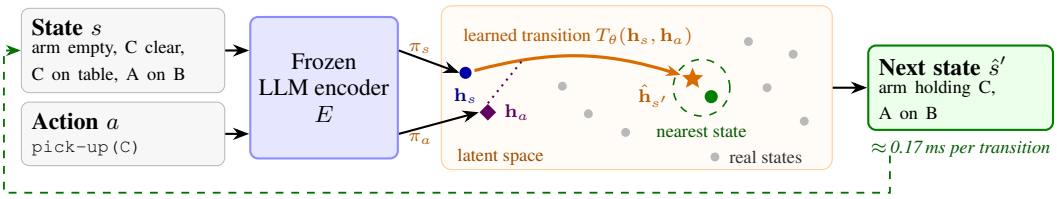


Figure 1: EMBEDPLAN. A frozen LLM encoder E embeds the state s and the action a (Blocksworld, `pick-up(C)`), and learned heads π_s, π_a project them into a latent space where the transition is computed: a small network T_θ predicts the next-state embedding $\hat{\mathbf{h}}_{s'}$, and the nearest real state is retrieved as the successor $\hat{s}'$. Fed back as the next input, it supports multi-step rollout. Search itself is external. Latency is per transition with state embeddings cached (Appendix B.5).

not a complete planner: search composes many transitions, and we study the single transition that search queries repeatedly. Because the same head can be trained over any encoder, the framework also serves as a controlled probe of what a frozen text representation supports for learning dynamics.

We evaluate EMBEDPLAN on 9 classical planning domains from ACPBENCH (Kokel et al., 2025), with states rendered as natural language (nearly 3 million transitions), and 4 frozen encoders from MPNet to Llama-3.3-70B. Six protocols form a ladder of exposure (Table 1). Test transitions come from observed problems in Interpolation (held out at random) and Plan-Variant (along unseen optimal plans), from unseen problems of an observed domain in Extrapolation and Multi-Domain (one model for all nine domains), and from unseen domains in Cross-Domain (trained on one other domain) and Leave-One-Out (trained on the other eight). We report Hit@k, the fraction of queries whose true next state ranks in the top k of 128 candidates (chance: 3.9% at k=5).

Our experiments show that, within observed problems, EMBEDPLAN predicts transitions accurately and fast enough for search (Section 4.1). Its predictions are near-perfect (99.7% Hit@5), keep the true next state in the top five for most queries even when every observed state is a candidate, and stay accurate when each prediction becomes the input to the next transition, as when rolling out a plan. Given the same candidates, EMBEDPLAN also picks the true next state more often than GPT-5.4, at about 0.17 ms per transition with cached state embeddings. This accuracy depends on exposure, however: Hit@5 falls to 54.6% on unseen problems of an observed domain and to 6.6%, near chance, on unseen domains (Section 4.2). Varying only the state representation shows that this boundary lies in the representation, not in the learned transition (Section 4.3).

Contributions.

- EMBEDPLAN, a transition model that runs on frozen LLM embeddings: a lightweight head predicts the next-state embedding, and nearest-neighbor retrieval returns a real state that can be fed back for multi-step rollout (Section 3).
- A controlled evaluation methodology for learned transition models over text: six protocols that hold out progressively more, from transitions to entire domains, across 9 domains and 4 encoders, with reference methods from no-change floors to symbolic action-model induction (Table 1).
- A systematic study that maps where such transitions work and where they fail, and identifies the frozen state representation as the main limit on transfer (Section 4).

2 RELATED WORK

Prompting methods frame planning as iterative generation, using chain-of-thought (Wei et al., 2022), tree search (Yao et al., 2023a), or environment interaction (Yao et al., 2023b; Shinn et al., 2023). Other approaches treat LLMs as world models for Monte Carlo tree search (MCTS) (Hao et al., 2023) or as text-based world simulators (Wang et al., 2024), or compile text into PDDL for classical solvers (Liu et al., 2023; Guan et al., 2023). However, the prompting and world-model approaches query an LLM at every decision, compilation requires a domain that can be expressed as a symbolic model, and LLM outputs in either role can be hallucinated or unexecutable (Kambhampati et al., 2024; Katz et al., 2024). EMBEDPLAN addresses this compute bottleneck by efficiently computing next states via latent transitions in frozen embedding spaces.

Early work demonstrated that planners can operate in a learned latent space rather than a hand-crafted symbolic state space. Asai & Fukunaga (2018) introduced LATPLAN, which learns discrete state encodings from visual inputs via autoencoders and applies classical planners in latent space. Model-based reinforcement learning extended this by learning dynamics models that predict latent states under actions (Ha & Schmidhuber, 2018; Hafner et al., 2019; 2020; Schrittwieser et al., 2020), including transformer world models (Micheli et al., 2023), latent tree search (Gieselmann & Pokorny, 2022), and language-conditioned world models (Lin et al., 2024). Closest to our design, C-SWM (Kipf et al., 2020) learns latent transitions contrastively and evaluates them by ranking, and DINO-WM (Zhou et al., 2025) learns dynamics over frozen pretrained visual features. However, these methods typically model visual or game observations with small or low-dimensional action spaces, whereas EMBEDPLAN builds on a frozen text encoder and must generalize over action *text* in combinatorial grounded action spaces (487 grounded actions on Ferry, 311 on Logistics). A parallel line of work keeps the state in text: Li et al. (2026) survey *text world models* (transition functions over textual states) and organize them by state representation into natural language, structured formats, and executable code, a taxonomy in which a learned latent vector is absent. Belief-graph agents for text games (Adhikari et al., 2020) likewise keep an explicit symbolic state. Huang et al. (2026) further argue that single-step state-matching metrics understate how a text world model behaves over a trajectory, motivating our Plan-Variant and plan-level evaluations alongside single-step Hit@k. Structured approaches offer complementary inductive biases for cross-domain transfer, each presupposing a symbolic decomposition of the state: object-centric architectures that factor states into entities and relations (Battaglia et al., 2018), neural generalized planners over lifted PDDL structure (Toyer et al., 2018; Shen et al., 2020; Ståhlberg et al., 2022), action-model learners that induce symbolic transition models from traces (Yang et al., 2007; Cresswell et al., 2013; Aineto et al., 2019; Juba et al., 2021), of which our lifted STRIPS reference is an instance, and symbolic compilation to PDDL (Liu et al., 2023; Guan et al., 2023). Our study quantifies the gap that these structured approaches must close and provides a diagnostic methodology to measure it.

Sentence embedding methods target semantic similarity, often using contrastive objectives (Reimers & Gurevych, 2019; Gao et al., 2021; Radford et al., 2021; Oved et al., 2025). Representation learning for control uses similar contrastive signals to induce state structure (Oord et al., 2018; Laskin et al., 2020; Schwarzer et al., 2021), while JEPA-style methods predict held-out representations in an embedding space rather than reconstructing raw inputs (Assran et al., 2023). Retrieval-augmented models similarly rely on embedding space lookups for prediction (Khandelwal et al., 2020) and control (Humphreys et al., 2022). However, these approaches largely focus on visual/RL settings or general-purpose similarity, and are not evaluated as action-conditioned transition operators for planning in frozen language embedding spaces. EMBEDPLAN bridges this gap by learning transition functions in a frozen LLM embedding space and systematically evaluating generalization across planning domains and problem configurations.

Classical planning benchmarks provide controlled testbeds (McDermott et al., 1998). ACPBENCH measures predictive transition accuracy and generalization across held-out problems (Kokel et al., 2025). Compositional generalization benchmarks reveal that neural models often memorize training patterns (Lake & Baroni, 2018; Kim & Linzen, 2020). We evaluate transition learning in embedding space with explicit within-distribution and out-of-distribution metrics, connecting failures to embeddings that cluster by surface form rather than by generalizable structure.

3 METHOD AND EXPERIMENTAL SETUP

Task formulation Given a state s and an action a described in natural language, we predict the next state s' with one learned vector step followed by retrieval, so that every prediction is a real state. Let E be a frozen LLM encoder mapping text to $\mathbb{R}^d$, and let $\pi_s, \pi_a : \mathbb{R}^d \to \mathbb{R}^{d'}$ be learned state and action projection heads with $d' = 128$. With $\mathbf{h}_s = \pi_s(E(s))$ and $\mathbf{h}_a = \pi_a(E(a))$, a lightweight transition network $T_\theta : \mathbb{R}^{d'} \times \mathbb{R}^{d'} \to \mathbb{R}^{d'}$ predicts the next-state embedding

$$\hat{\mathbf{h}}_{s'} = T_\theta\big(\mathbf{h}_s,\, \mathbf{h}_a\big). \tag{1}$$

At inference, the next state is the nearest neighbor in a candidate pool $\mathcal{C}$ of encoded states:

$$\hat{s}' = \arg\max_{c \in \mathcal{C}} \; \mathrm{sim}\big(\hat{\mathbf{h}}_{s'},\, \pi_s(E(c))\big), \tag{2}$$

where $\mathrm{sim}(\cdot,\cdot)$ denotes cosine similarity. Only π_s, π_a, and T_θ (together, the head) are trained.

Table 1: Evaluation protocols, grouped by exposure: what the test data share with training. Every query is ranked against 128 states, the true successor and 127 distractors (definitions in Appendix A).

Exposure	Protocol	Trained on → tested on	Distractors from
Observed problems	Interpolation	80% → 20% of transitions	the whole domain
	Plan-Variant	some → other optimal plans	alternative-plan successors
Unseen problems	Extrapolation	∼80% → 20% of problems	the query's problem
	Multi-Domain	the same, all nine domains at once	the query's problem
Unseen domains	Cross-Domain	one domain → another	the target domain
	Leave-One-Out	eight domains → the ninth	the target domain

Architecture The frozen encoder E is MPNet (all-mpnet-base-v2), BGE-M3, Qwen2.5-7B, or Llama-3.3-70B (768 to 8,192 dimensions). For the two decoder-only models we mean-pool the final hidden states (Appendix B.1.1). Results use Llama-3.3-70B unless another encoder is named. Each state is encoded as a prompt that contains the problem and goal description followed by a `CURRENT STATE` block listing the facts (PDDL literals) that hold (Appendix C.4). For T_θ we evaluate a residual MLP that concatenates its inputs and adds a skip connection, and a hypernetwork that generates action-specific parameters, both with fewer than 500K parameters. The two perform on par (Appendix G.1), so we report the MLP throughout (details in Appendix B.1).

Training objective We optimize $\mathcal{L} = \mathcal{L}_{\text{state}} + \lambda \mathcal{L}_{\text{action}}$ with $\lambda = 2$. The state loss is InfoNCE (Oord et al., 2018) between predicted and true next-state embeddings with in-batch negatives. The action loss requires the true action to yield a prediction closer to s' than up to $K{=}50$ alternative actions applied to the same state (Appendix B.2). Hyperparameters and checkpoints are selected on a held-out development split, and results average 3 seeds.

Intended use Action selection and search remain external to EMBEDPLAN. It targets settings where states and actions are available as text but no symbolic model is, so that the alternative is querying an LLM at every expansion. Our classical domains have exact simulators, which makes them a controlled testbed: every prediction can be scored against ground truth. Retrieval needs a set of candidate successors that contains the true one, so we measure how accuracy degrades as that set grows to every state observed in a domain, and how the transition behaves when iterated on its own predictions. Training is per domain: a one-time embedding pass plus ∼15 minutes of training, which breaks even against autoregressive generation after ∼4,300 transitions with Llama-3.3-70B (Appendix B.5), so it pays off for a domain that is planned in repeatedly.

Data We use nine classical PDDL domains from ACPBENCH (Kokel et al., 2025) (Appendix C.3): Blocksworld (a robot arm stacks blocks), Depot (hoists and trucks move crates), Ferry (a ferry carries one car at a time), Floortile (robots paint a tiled floor), Goldminer (a robot clears rock with a laser or bombs to reach gold), Grid (a robot carries keys through a grid of locked cells), Logistics (trucks and airplanes deliver packages within and across cities), Rovers (rovers sample, image, and transmit data), and Satellite (satellites calibrate instruments and take images). A *domain* fixes the action schemas, and a *problem* fixes the objects, the initial state, and the goal. Each action is a *grounded* instance (e.g., `pick-up(C)`) of a *lifted* action schema (`pick-up(?x)`). For each problem we extract state-action-next-state triplets (s_t, a_t, s_{t+1}), 2.97M transitions over 67 problems (259K unique states), from 13,256 (Depot) to 1,248,696 (Satellite) per domain, together with optimal plans for the plan-level protocols (Appendix C).

Metrics Hit@k is the fraction of queries whose true next state ranks in the top k of a candidate pool. Unless stated otherwise the pool holds 128 states, the true successor and 127 distractors drawn as in Table 1, so chance is 3.9% for Hit@5 and 0.8% for Hit@1. We emphasize Hit@5 because it is robust to the pool: when the pool grows to every observed state (Interpolation), Hit@1 falls by about 3× while Hit@5 falls by at most 1.33× (Appendix B.4), so 128-way Hit@1 largely reflects the pool size. A top-5 list is also useful where a verifier can filter candidates. Where we claim usability without one, we report Hit@1. Hit@1 and Hit@10 appear in most appendix tables.

Table 2: Top-5 retrieval degrades gracefully as the pool grows to every observed state, while Hit@1 falls about 3×. Interpolation, Llama-3.3-70B, % (mean over 3 seeds, SE in Table 10). "Full": every observed state of the domain.

	Ferry		Logistics	
Pool	**Hit@1**	**Hit@5**	**Hit@1**	**Hit@5**
128	99.0	100.0	96.3	99.9
512	97.2	99.9	88.2	99.4
2,048	90.9	99.4	70.1	96.7
8,192	76.6	98.0	47.2	83.1
Full	35.2	86.9	32.6	74.9
Full size	46,205		13,373	

Table 3: Trained on other transitions of the same problems, EMBEDPLAN matches or beats every LLM tested at a fraction of the compute. *Generate*: Hit@5 of the true successor among the LLM's generated candidates (EMBEDPLAN's retrieval Hit@5: first row of Table 2). *Rank*: top-1 accuracy over the same 128 candidates EMBEDPLAN ranks. Tokens and GFLOPs are per query, for EMBEDPLAN with cached state embeddings (Appendix B.5). n/a: not available.

Method	**Ferry**	**Logistics**	**Tokens**	**GFLOPs**
Generate s', Hit@5 (%)				
Llama-3.1-8B	2	95	3,840	∼4.8k
Qwen3-30B	41	86	874	∼18.7k
Llama-3.1-70B	45	100	781	∼35k
GPT-oss-120B	56	100	1,776	∼281k
GPT-5.4	56	100	829	n/a
Rank 128 candidates, top-1 (%)				
GPT-5.4	44	92	n/a	n/a
EMBEDPLAN	**99.0**	**96.3**	n/a	**0.3**

Reference methods To locate what limits transfer, we keep the head and pool and vary only the state representation (Section 4.3). Two non-learned floors need no training: Identity predicts the current state's embedding, and Offset adds the mean training displacement of the ground action or of its schema. The learned references feed the same head sparse lexical features of the same text (TF-IDF bigrams, bag-of-words, character 3–5 grams), a tabular one-hot code with one lookup row per state, or a bag of the state's symbolic facts (bag-of-literals). Lifted STRIPS induction instead infers each action schema's add and delete effects from those facts and serves as an oracle upper bound. Two null controls bound the protocol from below: the prompt without its `CURRENT STATE` block (context-only) and a fixed random Gaussian vector per state (Appendix L).

4 RESULTS

A learned transition over frozen text embeddings is near-perfect within observed problems, partial on unseen problems of an observed domain, and near chance on unseen domains. Section 4.1 shows that within observed problems the component has the properties search needs, Section 4.2 maps where generalization stops, and Section 4.3 traces the limit to the frozen state representation.

4.1 WITHIN OBSERVED PROBLEMS

Accuracy For every encoder but the smallest, MPNet (Table 6), Interpolation Hit@5 is at least 99.5% (99.7% with Llama-3.3-70B, mean Hit@1 92.1% in Table 14), and predictions land next to their targets (Figure 2). MPNet scores near zero on Rovers and Satellite even along plans of observed problems (Table 21), likely because MPNet reads at most 384 tokens and these domains' long prompts end with the state block. Interpolation is the easy regime: test transitions come from training problems, a transition that recurs across plans can fall in both splits, and distractors come from the whole domain, so even predicting no change (Identity) reaches 73.7% Hit@5 on the three domains of Table 7. Only learned transitions reach the ceiling, and Hit@1 separates them from the floor (95.4% against 46.9%, Table 34).

Candidate-pool scaling Top-5 retrieval holds up when the 128-state pool grows to every observed state of the domain (Table 2). Hit@1 falls about 3×, to 35.2% on Ferry (46,205 states) and 32.6% on Logistics (13,373), whereas Hit@5 falls from 100.0% to 86.9% and from 99.9% to 74.9%, and Hit@10 stays at 95.0% and 88.0% (Appendix B.4). The true successor thus stays in a short list against the full pool, though not reliably at rank one.

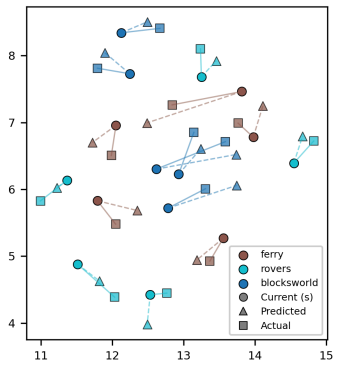


Figure 2: Predictions land next to their targets (illustrative). PCA of the latent space for sampled transitions from three domains (Interpolation, BGE-M3): states s (○), predicted embeddings $\hat{\mathbf{h}}_{s'}$ (△), true next states s' (□). Each state is linked to its true successor (solid) and to its prediction (dashed).

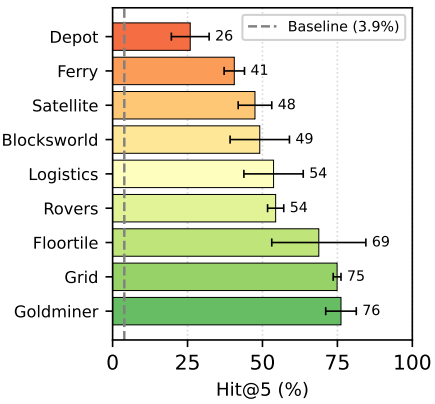


Figure 3: Extrapolation varies widely by domain (Llama-3.3-70B). Hit@5 (%) on unseen problems. Dashed line: chance (5 of 128 = 3.9%). Error bars: ± SE.

Table 4: Replacing each state prediction with the nearest real state before the next step (*snapped*) keeps multi-step rollout within 92–99% of the accuracy obtained with the true state at each step. Step-wise Hit@1 (%), Interpolation, 1,000 candidate states, mean ± SE over seeds (Blocksworld: single run). Retention: snapped over true-state accuracy.

Regime	**Ferry**	**Logistics**	**Blocksworld**
True state at each step	95.3 ± 0.5	74.9 ± 2.2	80.3
Own prediction, snapped	94.2 ± 0.7	69.0 ± 2.1	78.4
Own prediction, not snapped	68.3 ± 2.0	40.0 ± 1.9	59.6
Retention (snapped)	99%	92%	98%

Comparison to LLMs Within observed problems, EMBEDPLAN matches or beats every LLM we test (Table 3). We compare on Ferry and Logistics, two domains with different state-tracking structure, in two tasks. In *generation*, the LLM receives s and a and writes s': all five do well on Logistics, but none matches the true Ferry state more than 56% of the time. Because generation is harder than retrieval, we also give GPT-5.4 *the identical ranking task* over the same 128 candidates: it selects the true successor for 44% of Ferry and 92% of Logistics queries, against 99.0% and 96.3% for EMBEDPLAN (on Logistics the margin is small next to the spread across our runs, 91.2–96.3%). The comparison is matched in task, not in information: EMBEDPLAN has observed other transitions of the same problems, whereas the LLM is zero-shot, and on unseen problems EMBEDPLAN's Ferry Hit@1 falls to 12.0% (Table 15). What one-time training buys within observed problems is accurate successor ranking 101× faster end to end than generation through an LLM API (1.9 s per transition), and 11,215× faster with cached state embeddings (Appendix B.5).

Closed-loop rollout Supplying the true state at every step (teacher forcing) hides error accumulation, so we also feed the model's own output back as the next input (Table 4), in two regimes. *With snapping*, each state prediction is replaced by the nearest real state the model retrieves, right or wrong, and that state becomes the next input, so retrieval errors propagate. *Without snapping*, the raw predicted embedding is fed back and no candidate states are consulted at intermediate steps. With snapping, rollout retains 92–99% of teacher-forced step accuracy on all three domains. Because the regimes coincide at depth 1 by construction, the aggregate dilutes the contrast: from depth 2 on, snapped rollout on Ferry retains 98% of teacher-forced Hit@1 (94.1% vs. 96.0%), against 49% (46.7%) without snapping, whose latent drifts off the manifold of real state embeddings (cosine to

Table 5: Hit@5 stays near chance until the target domain has been observed. Llama-3.3-70B, mean ± SE across domains. Rows are ordered by what the test data share with training, and Δ is the gain over chance in percentage points (pp). Plan-Variant faces the hardest distractors (Table 1).

Protocol	Observed	Hit@5	Δ
Interpolation	Problems	99.7 ± 0.1	$+96$
Plan-Variant	Problems	51.2 ± 5.5	$+47$
Extrapolation	Domain	54.6 ± 5.5	$+51$
Multi-Domain	Domain	37.2 ± 3.8	$+33$
Leave-One-Out	8 others	9.2 ± 1.2	$+5.3$
Cross-Domain	1 other	6.6 ± 0.5	$+2.7$
Chance		3.9	

Table 6: Larger encoders extrapolate better, but none closes the gap. Hit@5 (%), mean ± SE across the nine domains. Gap in pp, with a paired t-test across domains: $^{**}p < 0.01$, $^{***}p < 0.001$.

Encoder	Params	Interp.	Extrap.	Gap
MPNet	110M	70.0 ± 15	26.8 ± 6.3	43^{**}
BGE-M3	568M	99.6 ± 0.2	36.3 ± 5.0	63^{***}
Qwen2.5-7B	7B	99.5 ± 0.2	47.7 ± 4.9	52^{***}
Llama-3.3-70B	70B	99.7 ± 0.1	54.6 ± 5.5	45^{***}

the nearest real state 0.83 after one step and ~ 0.55 thereafter, Appendix D.8). Per-step snapping is therefore what keeps multi-step rollout accurate. The test trajectories are short (mean 2.2 steps on Ferry), so this establishes stability over short horizons only.

Action disambiguation To test whether the model separates the effects of different actions, we apply every action applicable in s and rank the actions by how close their predictions come to the true s'. Acc@k is the fraction of queries whose true action ranks in the top k. Mean Acc@5 is 85.3% within observed problems (Acc@1 32.0%, Table 14) and 16.2% on unseen problems, where training without the action loss reaches only 4.8% (Appendix G.2).

4.2 Where generalization stops

Accuracy falls with each step down the exposure ladder (Table 5). Observing a domain lifts Hit@5 from 6.6% (Cross-Domain) to 54.6% on its unseen problems, and observing the problems themselves lifts it to 99.7%. The second gap also includes the harder, same-problem distractors of Extrapolation, so it upper-bounds the cost of unseen problems alone.

Unseen problems On unseen problems of an observed domain, Hit@5 is 54.6% ($14\times$ chance) and well above the no-change floor at Hit@1 (21.4% against 3.3% for Identity on the three reference domains of Table 7). Scaling the encoder helps with diminishing returns (from 26.8% for MPNet to 54.6% for Llama-3.3-70B, Table 6), and no encoder closes the gap. Difficulty depends more on the domain than on the encoder (26–76%, Figure 3 and Table 13): grid-based domains with local, predictable effects (Goldminer, Grid) extrapolate best, and Depot, which requires compositional multi-object reasoning, extrapolates worst. When the true successor misses the top five, the top-ranked state typically differs from it in one to three facts (median 2, Appendix J).

Along whole plans We also score every step of held-out optimal plans with teacher forcing. Under Plan-Variant the plans are unseen but their problems are observed, and each step is ranked against successors along alternative optimal plans, the hardest distractors we use: they are reachable and often differ from the target in a single fact. Mean per-step Hit@5 is 51.2%, and 22.6% of plans are correct at every step. Plans of unseen problems fare far worse, at 10.1% per step and 3.3% of plans (Table 20), with the same collapse for every encoder (Table 21).

Unseen domains Transfer across domains fails almost completely. Cross-Domain reaches 6.6%, only 2.7 pp above chance (the untrained model scores 4.2%, Appendix D.6). The full 9×9 matrix (Table 16) shows one notable success, Ferry→Logistics (22.3%), two domains that both transport objects between locations. Leave-One-Out fares little better at 9.2% despite eight training domains (Table 17). All nine domains share discrete states, typed objects, and precondition–effect schemas, yet the learned transition captures essentially none of this commonality. We see two plausible causes, which our experiments do not separate. First, frozen embeddings organize states by surface form,

Table 7: With the head, data, and pool fixed, the state representation decides transfer to unseen problems. Extrapolation, Hit@5 (%) per domain (mean ± SE over 3 seeds) and three-domain means. The learned rows train the identical head on a different state representation (lifted STRIPS induction instead induces symbolic effects), and the first group requires the symbolic facts of each state. On these templated states an action edits only a few facts, which favors sparse and symbolic representations (Section 4.3). Bold: best without symbolic facts.

	Hit@5				Hit@1
Method	**Ferry**	**Logistics**	**Goldminer**	**Mean**	**Mean**
Given the symbolic facts of each state					
Lifted STRIPS induction (oracle)	100.0 ± 0.0	100.0 ± 0.0	99.3 ± 0.3	99.8	99.8
Bag-of-literals	51.9 ± 5.7	81.4 ± 0.7	95.7 ± 2.0	76.3	37.1
Sparse lexical features of the same text					
Char 3–5 grams	71.1 ± 1.7	69.8 ± 7.4	97.3 ± 0.7	**79.4**	41.9
Bag-of-words	70.8 ± 2.7	67.3 ± 4.8	99.3 ± 0.2	79.1	**45.0**
TF-IDF bigrams	59.5 ± 2.1	50.0 ± 4.3	97.2 ± 1.2	68.9	37.2
Frozen LLM embeddings					
EMBEDPLAN (Llama-3.3-70B)	40.6 ± 3.5	53.7 ± 9.8	76.2 ± 5.2	56.8	21.4
EMBEDPLAN (BGE-M3)	42.8 ± 2.1	51.7 ± 3.9	70.2 ± 5.2	54.9	19.9
Offset, lifted	34.3 ± 0.8	56.4 ± 5.4	50.7 ± 1.7	47.2	5.5
Offset, grounded	28.6 ± 2.1	53.1 ± 2.3	57.3 ± 4.1	46.3	7.2
Identity	18.5 ± 1.0	49.8 ± 3.4	41.2 ± 1.6	36.5	3.3
Random vectors [null]	4.4 ± 0.5	4.8 ± 0.6	5.2 ± 0.5	4.8	0.7
Context-only [null]	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0	0.0

which differs across domains, rather than by structural role (Appendix F.1): nothing tells the model that `pick-up(BlockA)` and `board(Car1)` follow the same precondition–effect pattern. Second, the learned action semantics are domain-specific: the action disambiguation loss improves generalization within a domain (Appendix G.2) by learning fine distinctions among one domain's actions, which transfer poorly to another's. Training one model on all nine domains is a partial remedy: it reaches 37.2%, which is 17 pp below the single-domain models (Table 18), keeping about two-thirds of their accuracy with one model instead of nine.

4.3 WHAT LIMITS TRANSFER

Because the head is independent of the encoder, the framework can attribute the transfer gap to its source. We train the same head on the same data, rank against the same pool, and vary only the state representation (Table 7), on three domains that span the observed extrapolation difficulty: Ferry, Logistics, and Goldminer.

Learning beats no change and lookup. On unseen problems the learned transition improves on Identity by 20 pp Hit@5 (18 pp Hit@1) and on the best Offset by 10 pp (14 pp). A tabular one-hot model trained on identical data, which has no representation for states it has not seen, reaches 8.3% Hit@1 on Logistics even within observed problems (against 96.3% for EMBEDPLAN) and 0.2% on unseen problems, below the 0.8% chance level (Appendix L). *The head is not the main limit.* Fed character 3–5 grams of the same text, the identical head reaches 79.4% Hit@5 (41.9% Hit@1), against 56.8% (21.4%) with Llama-3.3-70B embeddings, and given each state's literal decomposition, which our text interface does not require, lifted STRIPS induction recovers the dynamics from the same data exactly (99.8%). Both advantages come from how the benchmark renders states. Every state is generated from a fixed template, so an action edits only a few facts of an otherwise identical text: `pick-up(C)` replaces "arm empty", "C clear", and "C on table" with "arm holding C" and leaves every other fact unchanged. Sparse features encode such an edit directly as a few changed dimensions, and a symbolic learner reads it off the literals, whereas a pooled LLM embedding of the whole prompt, most of which is shared problem and goal text, barely moves (cosine 0.997–0.999 between consecutive states with BGE-M3). In free-form descriptions, where a fact can be phrased in many ways and no literal decomposition is given, neither shortcut is available, and that is the regime the text interface targets (Section 5.1). Adapting the encoder already helps: low-rank adaptation (LoRA) of BGE-M3 raises full-pool Hit@1 on unseen problems from 3.8% to 9.3% when the head

is first trained with the encoder frozen (6.2% with a cold joint start, Appendix M), and larger frozen encoders extrapolate better (Table 6).

Two observations show what a better encoder must capture. First, transfer to a new problem means handling new objects: most held-out transitions use ground actions never seen in training (57.6% on Ferry, 65.5% on Logistics), although no action schema is new (Appendix L.4), and the lifted STRIPS learner handles them by abstracting objects into schema parameters. Second, reordering a state's facts moves a BGE-M3 embedding 1.13× further than applying an action (Appendix N), whereas bags of words, character n-grams, and literals are largely insensitive to fact order. Because the encoder is a plug-in, the results name a concrete target, an object-aware, order-invariant state encoder, which the framework can evaluate directly.

5 Conclusion

We introduced EmbedPlan, a transition model built on frozen text embeddings, and mapped its boundaries across 9 domains, 4 encoders, 6 protocols of increasing exposure, and reference methods from no-change floors to symbolic induction. Within observed problems, the component has the properties search needs: transition prediction is near-perfect (99.7% Hit@5), the true next state stays in the top five for most queries even when every observed state is a candidate, chaining predictions over several steps keeps 92–99% of single-step accuracy, and EmbedPlan beats GPT-5.4 at picking the true next state from the same candidates while taking about 0.17 ms per transition with cached state embeddings. On unseen problems it remains well above chance and the no-change floor (54.6% Hit@5), and on unseen domains it stays near chance. Because the same head can be trained over any representation, the framework also locates this boundary in the state representation. On our templated states, an action edits only a few facts, which sparse lexical and symbolic representations capture directly and pooled LLM embeddings barely register. State encoders that bind objects to roles and ignore fact order are therefore a concrete target for transfer to new problems. More broadly, separating a frozen encoder from a cheap learned transition makes text-based planning modular: any new text encoder can be dropped in and measured with the same head and protocols.

5.1 Limitations and future work

EmbedPlan is a transition component for discrete, domain-specific settings, not a domain-general planner, and must be trained per domain because cross-domain transfer fails. Retrieval assumes a candidate pool that contains the true successor. Our domains are templated, so we have not yet tested the regime the text interface is meant for: states without a clean literal decomposition, where symbolic induction and lexical features would break. Several results also rest on narrower evidence: pool scaling, multi-step rollout, and LLM ranking are measured within observed problems, rollouts cover short test trajectories (mean 2.2 steps), Extrapolation rests on one or two held-out problems per domain, and the reference comparison covers three domains, with two of three seeds sharing their held-out problems (Appendix L.5) and without ablating the problem and goal text that every prompt shares. Integrating EmbedPlan into beam or tree search over long horizons on held-out problems is the direct next step.

AI use statement

We used AI assistants to edit author-written text and figures for clarity and structure, to help explore the relevant literature, and for coding assistance in implementing the experimental pipeline. The research questions, method, evaluation protocols, experiments, and results are the authors' own, and every reported number was produced by the experimental code and runs described in Section 3 and the appendix. The authors reviewed all AI-assisted text, figures, and code and take full responsibility for the content of this paper.

Ethics statement

This work studies transition prediction on classical planning benchmarks rendered as natural language. It involves no human subjects, no personally identifiable information, and no sensitive data. All domains and problem instances are drawn from publicly available planning benchmarks, and

the pre-trained encoders we use are publicly released models. We see no direct pathway from this study to harmful applications beyond those already associated with general-purpose language models. The reported failure modes under cross-domain transfer are stated explicitly so that the method is not over-claimed for safety-critical planning.

Reproducibility statement

The experimental protocols, dataset construction, and splits are described in Section 3, with full hyperparameters, encoder configurations, and per-domain results given in the appendix. All evaluation protocols are defined precisely enough to be re-implemented, and the complete numerical results underlying every figure and table in the main paper are reported in the appendix. Source code and the scripts used to generate the transition datasets are available in an anonymized repository: `https://anonymous.4open.science/r/AnonymizedTextualPlanningLatentTransitions-DC6D/README.md`.

References

Ashutosh Adhikari, Xingdi Yuan, Marc-Alexandre Côté, Mikulá Zelinka, Marc-Antoine Rondeau, Romain Laroche, Pascal Poupart, Jian Tang, Adam Trischler, and Will Hamilton. Learning Dynamic Belief Graphs to Generalize on Text-Based Games. In *Advances in Neural Information Processing Systems*, volume 33, pp. 3045–3057. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/hash/1fc30b9d4319760b04fab735fbfed9a9-Abstract.html.

Diego Aineto, Sergio Jiménez Celorrio, and Eva Onaindia. Learning action models with minimal observability. *Artificial Intelligence*, 275:104–137, October 2019. ISSN 0004-3702. doi: 10.1016/j.artint.2019.05.003. URL https://www.sciencedirect.com/science/article/pii/S0004370218304259.

Masataro Asai and Alex Fukunaga. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *Proceedings of the aaai conference on artificial intelligence*, volume 32, 2018.

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. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15619–15629, 2023.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Stephen N. Cresswell, Thomas L. McCluskey, and Margaret M. West. Acquiring planning domain models using LOCM. *The Knowledge Engineering Review*, 28(2):195–213, June 2013. ISSN 0269-8889, 1469-8005. doi: 10.1017/S0269888912000422. URL https://www.cambridge.org/core/journals/knowledge-engineering-review/article/abs/acquiring-planning-domain-models-using-locm/139F7A0986CD481F88A778B08891DC49.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. Simcse: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 6894–6910, 2021.

Robert Gieselmann and Florian T Pokorny. Latent planning via expansive tree search. *Advances in Neural Information Processing Systems*, 35:16821–16835, 2022.

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging pretrained large language models to construct and utilize world models for model-based task planning. *Advances in Neural Information Processing Systems*, 36:79081–79094, 2023.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. *Advances in neural information processing systems*, 31, 2018.

David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rkpACe1lx.

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, 2019.

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 Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 8154–8173, 2023.

Youling Huang, Guanqiao Chen, Junchi Yao, Lu Wang, Fangkai Yang, Chao Du, ChenZhuo Zhao, Pu Zhao, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. Beyond state consistency: Behavior consistency in text-based world models, 2026. URL https://arxiv.org/abs/2604.13824.

Peter Humphreys, Arthur Guez, Olivier Tieleman, Laurent Sifre, Théophane Weber, and Timothy Lillicrap. Large-scale retrieval for reinforcement learning. *Advances in Neural Information Processing Systems*, 35:20092–20104, 2022.

Brendan Juba, Hai S. Le, and Roni Stern. Safe Learning of Lifted Action Models, July 2021. URL http://arxiv.org/abs/2107.04169. arXiv:2107.04169 [cs.AI].

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Paul Saldyt, and Anil B Murthy. Position: Llms cant plan, but can help planning in llm-modulo frameworks. In *Forty-first International Conference on Machine Learning*, 2024.

Michael Katz, Harsha Kokel, Kavitha Srinivas, and Shirin Sohrabi Araghi. Thought of search: Planning with language models through the lens of efficiency. *Advances in Neural Information Processing Systems*, 37:138491–138568, 2024.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. In *International Conference on Learning Representations*, 2020.

Najoung Kim and Tal Linzen. Cogs: A compositional generalization challenge based on semantic interpretation. In *Empirical Methods in Natural Language Processing*, 2020.

Thomas Kipf, Elise van der Pol, and Max Welling. Contrastive learning of structured world models. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1gax6VtDB.

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

Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International conference on machine learning*, pp. 2873–2882. PMLR, 2018.

Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. In *International conference on machine learning*, pp. 5639–5650. PMLR, 2020.

Yixia Li, Hongru Wang, Peng Lai, Zhiwen Ruan, He Zhu, Youxin Zhu, Ganlong Zhao, Minda Hu, Yun Chen, Sibei Yang, Peng Li, Jeff Z. Pan, Jia Pan, Guanhua Chen, Yang Liu, and Guanbin Li. Bridging the agent-world gap: Text world models for llm-based agents, 2026. URL https://arxiv.org/abs/2606.09032.

Jessy Lin, Yuqing Du, Olivia Watkins, Danijar Hafner, Pieter Abbeel, Dan Klein, and Anca Dragan. Learning to Model the World with Language, May 2024. URL http://arxiv.org/abs/2308.01399. arXiv:2308.01399 [cs.CL].

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023.

Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL – The Planning Domain Definition Language – Version 1.2. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

Vincent Micheli, Eloi Alonso, and François Fleuret. Transformers are sample-efficient world models. In *International Conference on Learning Representations*, 2023.

Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. Mteb: Massive text embedding benchmark. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 2014–2037, 2023.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

James Oswald, Kavitha Srinivas, Harsha Kokel, Junkyu Lee, Michael Katz, and Shirin Sohrabi. Large language models as planning domain generators. In *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*, pp. 423–431. AAAI Press, 2024.

Alon Oved, Segev Shlomov, Sergey Zeltyn, Nir Mashkif, and Avi Yaeli. Snap: semantic stories for next activity prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 28871–28877, 2025.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PmLR, 2021.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992, 2019.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

Max Schwarzer, Ankesh Anand, Rishab Goel, R Devon Hjelm, Aaron Courville, and Philip Bachman. Data-efficient reinforcement learning with self-predictive representations. In *International Conference on Learning Representations*, 2021.

William Shen, Felipe Trevizan, and Sylvie Thiébaux. Learning Domain-Independent Planning Heuristics with Hypergraph Networks. *Proceedings of the International Conference on Automated Planning and Scheduling*, 30:574–584, June 2020. ISSN 2334-0843. doi: 10.1609/icaps.v30i1.6754. URL https://ojs.aaai.org/index.php/ICAPS/article/view/6754.

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.

Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning General Optimal Policies with Graph Neural Networks: Expressive Power, Transparency, and Limits. *Proceedings of the International Conference on Automated Planning and Scheduling*, 32:629–637, June 2022. ISSN 2334-0843. doi: 10.1609/icaps.v32i1.19851. URL https://ojs.aaai.org/index.php/ICAPS/article/view/19851.

Marcus Tantakoun, Christian Muise, and Xiaodan Zhu. LLMs as planning formalizers: A survey for leveraging large language models to construct automated planning models. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 25167–25188, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.1291. URL https://aclanthology.org/2025.findings-acl.1291/.

Sam Toyer, Felipe Trevizan, Sylvie Thiébaux, and Lexing Xie. Action Schema Networks: Generalised Policies With Deep Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), April 2018. ISSN 2374-3468. doi: 10.1609/aaai.v32i1.12089. URL https://ojs.aaai.org/index.php/AAAI/article/view/12089.

Ruoyao Wang, Graham Todd, Ziang Xiao, Xingdi Yuan, Marc-Alexandre Côté, Peter Clark, and Peter Jansen. Can language models serve as text-based world simulators? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1–17, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Qiang Yang, Kangheng Wu, and Yunfei Jiang. Learning action models from plan examples using weighted MAX-SAT. *Artificial Intelligence*, 171(2):107–143, February 2007. ISSN 0004-3702. doi: 10.1016/j.artint.2006.11.005. URL https://www.sciencedirect.com/science/article/pii/S0004370206001408.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023a.

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

Gaoyue Zhou, Hengkai Pan, Yann Lecun, and Lerrel Pinto. DINO-WM: World models on pre-trained visual features enable zero-shot planning. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 79115–79135. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/zhou25t.html.

Max Zuo, Francisco Piedrahita Velez, Xiaochen Li, Michael Littman, and Stephen Bach. Planetarium: A rigorous benchmark for translating text to structured planning languages. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 11223–11240, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.560. URL https://aclanthology.org/2025.naacl-long.560/.

A EVALUATION PROTOCOLS

These protocols form a diagnostic hierarchy for capability characterization. We deliberately isolate transition accuracy from search algorithms, candidate generation, and latency measurement to provide a clean assessment of what frozen embeddings support. End-to-end planning evaluation, while important for deployed systems, would conflate these factors.

We evaluate EMBEDPLAN under six protocols that measure progressively more demanding generalization capabilities. Let $\mathcal{D} = \{d_1, \ldots, d_9\}$ denote the set of nine planning domains. For each domain $d \in \mathcal{D}$, let $\mathcal{P}_d = \{p_1^d, \ldots, p_{n_d}^d\}$ denote the set of problem instances, where $n_d = |\mathcal{P}_d|$. Each problem $p \in \mathcal{P}_d$ yields a set of transitions $\mathcal{T}_p = \{(s_i, a_i, s_i')\}_{i=1}^{m_p}$ extracted from optimal plan trajectories, where $m_p = |\mathcal{T}_p|$.

The complete transition set for domain d is $\mathcal{T}_d = \bigcup_{p \in \mathcal{P}_d} \mathcal{T}_p$, and the global transition set across all domains is $\mathcal{T} = \bigcup_{d \in \mathcal{D}} \mathcal{T}_d$.

A.1 PROTOCOL 1: INTERPOLATION SPLIT

Definition Transitions are randomly partitioned regardless of problem or domain membership:

$$\mathcal{T}_d = \mathcal{T}_d^{\text{train}} \cup \mathcal{T}_d^{\text{test}}, \quad \mathcal{T}_d^{\text{train}} \cap \mathcal{T}_d^{\text{test}} = \emptyset, \tag{3}$$

$$|\mathcal{T}_d^{\text{train}}| \approx 0.8\,|\mathcal{T}_d|, \quad |\mathcal{T}_d^{\text{test}}| \approx 0.2\,|\mathcal{T}_d|, \tag{4}$$

where each transition $(s, a, s') \in \mathcal{T}_d$ is assigned to train or test uniformly at random.

Key property For at least one problem p, transitions appear in both splits:

$$\exists\, p \in \mathcal{P}_d : \ \mathcal{T}_p \cap \mathcal{T}_d^{\text{train}} \neq \emptyset \ \wedge \ \mathcal{T}_p \cap \mathcal{T}_d^{\text{test}} \neq \emptyset. \tag{5}$$

Measures *Interpolation* within known problem manifolds. Models can succeed by recognizing which problem instance a state belongs to and applying memorized problem-specific transition patterns.

A.2 PROTOCOL 2: PLAN-VARIANT SPLIT

Definition We evaluate full-plan execution on alternative optimal plans for problems seen during training. Let $\Pi_p = \{\pi_1, \pi_2, \ldots, \pi_k\}$ denote the set of optimal plans for problem p, where each plan π_i is a sequence of actions leading from the initial state to the goal. Plans are partitioned:

$$\Pi_p = \Pi_p^{\text{train}} \cup \Pi_p^{\text{test}}, \quad \Pi_p^{\text{train}} \cap \Pi_p^{\text{test}} = \emptyset, \tag{6}$$

$$\mathcal{T}_p^{\text{train}} = \bigcup_{\pi \in \Pi_p^{\text{train}}} \mathcal{T}_\pi, \quad \mathcal{T}_p^{\text{test}} = \bigcup_{\pi \in \Pi_p^{\text{test}}} \mathcal{T}_\pi, \tag{7}$$

where $\mathcal{T}_\pi$ denotes the transitions extracted from plan π. Optimal plans are obtained directly from the dataset. For problems admitting multiple optimal solutions, we partition plans such that train and test contain non-overlapping action sequences. Problems with unique optimal plans are excluded from this protocol.

Key property Train and test share the same underlying problem instances but differ in the action sequences used to reach the goal:

$$\mathcal{P}^{\text{train}} = \mathcal{P}^{\text{test}} = \mathcal{P}_d, \quad \text{but} \quad \Pi_p^{\text{train}} \cap \Pi_p^{\text{test}} = \emptyset \ \ \forall\, p \in \mathcal{P}_d. \tag{8}$$

Measures *Generalization to unseen solution paths under fixed problem structure.* Models must learn transition dynamics that transfer across different valid action sequences within the same problem, rather than memorizing specific plan trajectories. This tests whether the model captures the underlying state-action mechanics or merely overfits to observed plans.

A.3 Protocol 3: Extrapolation split

Definition Problems are partitioned, and all transitions from each problem go exclusively to train or test:

$$\mathcal{P}_d = \mathcal{P}_d^{\text{train}} \cup \mathcal{P}_d^{\text{test}}, \quad \mathcal{P}_d^{\text{train}} \cap \mathcal{P}_d^{\text{test}} = \emptyset, \tag{9}$$

$$\mathcal{T}_d^{\text{train}} = \bigcup_{p \in \mathcal{P}_d^{\text{train}}} \mathcal{T}_p, \quad \mathcal{T}_d^{\text{test}} = \bigcup_{p \in \mathcal{P}_d^{\text{test}}} \mathcal{T}_p, \tag{10}$$

with $|\mathcal{P}_d^{\text{train}}| \approx 0.8\,|\mathcal{P}_d|$.

Key property Train and test transitions come from disjoint problem sets:

$$\forall\, p \in \mathcal{P}_d^{\text{test}} :\ \mathcal{T}_p \cap \mathcal{T}_d^{\text{train}} = \emptyset. \tag{11}$$

Measures *Extrapolation* to new problem configurations within a known domain. Models must learn transferable domain dynamics (e.g., "pick-up removes a block from a surface") rather than problem-specific patterns.

A.4 Protocol 4: Multi-Domain learning (unified model)

Definition We train a single model on all domains simultaneously, with Extrapolation splits within each domain:

$$\mathcal{T}^{\text{train}} = \bigcup_{d \in \mathcal{D}} \mathcal{T}_d^{\text{train}}, \quad \text{where } \mathcal{T}_d^{\text{train}} = \bigcup_{p \in \mathcal{P}_d^{\text{train}}} \mathcal{T}_p, \tag{12}$$

$$\mathcal{T}^{\text{test}} = \bigcup_{d \in \mathcal{D}} \mathcal{T}_d^{\text{test}}, \quad \text{where } \mathcal{T}_d^{\text{test}} = \bigcup_{p \in \mathcal{P}_d^{\text{test}}} \mathcal{T}_p. \tag{13}$$

A single transition network T_θ^{multi} is trained on $\mathcal{T}^{\text{train}}$ and evaluated per domain:

$$\text{Multi-Domain}(d) = \text{Hit@}k\big(T_\theta^{\text{multi}}, \mathcal{T}_d^{\text{test}}\big). \tag{14}$$

Key property A single network $T_\theta^{\text{multi}} : \mathbb{R}^{128} \times \mathbb{R}^{128} \to \mathbb{R}^{128}$ must represent the transition functions of all nine domains in one shared parameter space.

Measures *Capacity sharing across domains.* Tests whether a unified 128-dimensional latent space can encode transition dynamics for multiple domains simultaneously.

A.5 Protocol 5: Cross-Domain transfer (zero-shot)

Definition We train on one source domain and evaluate on a different target domain:

$$\mathcal{T}^{\text{train}} = \mathcal{T}_{d_{\text{src}}}, \quad d_{\text{src}} \in \mathcal{D}, \tag{15}$$

$$\mathcal{T}^{\text{test}} = \mathcal{T}_{d_{\text{tgt}}}, \quad d_{\text{tgt}} \in \mathcal{D} \setminus \{d_{\text{src}}\}. \tag{16}$$

For comprehensive evaluation, we compute performance for all source-target pairs:

$$\text{Cross-Domain}(d_{\text{tgt}}) = \frac{1}{|\mathcal{D}| - 1} \sum_{d_{\text{src}} \neq d_{\text{tgt}}} \text{Hit@}k\Big(T_\theta^{d_{\text{src}}},\, \mathcal{T}_{d_{\text{tgt}}}\Big), \tag{17}$$

where $T_\theta^{d_{\text{src}}}$ denotes the transition network trained on domain d_{src}.

Key property Source and target share no domain, problem, or transition: $\mathcal{T}^{\text{train}} \cap \mathcal{T}^{\text{test}} = \emptyset$, and the object vocabularies and action schemas differ entirely.

Measures *Zero-shot domain transfer.* Models must learn transition dynamics that generalize across fundamentally different planning structures (e.g., from block manipulation to logistics).

A.6 Protocol 6: Leave-One-Out generalization

Definition We train on $|\mathcal{D}| - 1$ domains and evaluate on the held-out domain:

$$\mathcal{T}^{\text{train}} = \bigcup_{d \in \mathcal{D} \setminus \{d_{\text{held}}\}} \mathcal{T}_d, \tag{18}$$

$$\mathcal{T}^{\text{test}} = \mathcal{T}_{d_{\text{held}}}. \tag{19}$$

For comprehensive evaluation, we iterate over all held-out domains:

$$\text{LOO}(d_{\text{held}}) = \text{Hit@}k\big(T_\theta^{\mathcal{D} \setminus \{d_{\text{held}}\}}, \mathcal{T}_{d_{\text{held}}}\big). \tag{20}$$

Key property Unlike Cross-Domain, which uses a single source, LOO provides maximum training diversity:

$$|\mathcal{T}^{\text{train}}| = \sum_{d \neq d_{\text{held}}} |\mathcal{T}_d| \;\gg\; |\mathcal{T}_{d_{\text{src}}}|. \tag{21}$$

Measures *Transfer from diverse training to an unseen domain.* Tests whether exposure to eight diverse domains enables generalization to an entirely novel ninth domain.

A.7 Protocol hierarchy

The six protocols form a hierarchy of increasing generalization difficulty.

Table 8: Evaluation protocol hierarchy by generalization difficulty.

Protocol	Measures	Shared domain	Shared problems	Shared plans	Difficulty
Interpolation	Interpolation	Yes	Yes	Yes	Lowest
Plan-Variant	Plan generalization	Yes	Yes	No	Low
Extrapolation	Extrapolation	Yes	No	No	Medium
Multi-Domain	Capacity sharing	Yes	No	No	Medium
Cross-Domain	Zero-shot transfer	No	No	No	High
Leave-One-Out	Diverse transfer	No	No	No	Highest

Design rationale This hierarchy isolates where frozen embeddings succeed versus fail:

- **Interpolation** → **Plan-Variant**: quantifies overfitting to specific action sequences.
- **Plan-Variant** → **Extrapolation**: quantifies the interpolation-extrapolation gap.
- **Extrapolation** → **Multi-Domain**: tests capacity sharing.
- **Multi-Domain** → **Cross-Domain/LOO**: tests zero-shot transfer.

B Experimental setup

B.1 Architecture

Figure 4 provides a complete illustration of the EmbedPlan architecture.

B.1.1 Frozen encoders

States and actions are encoded using frozen pre-trained language models. We experiment with four encoders spanning different architectures and scales.

MPNet (all-mpnet-base-v2) A 110-million parameter sentence transformer (Reimers & Gurevych, 2019) trained on over 1 billion sentence pairs. This model produces 768-dimensional embeddings optimized for semantic similarity tasks. We use the `sentence-transformers` library with default mean pooling.

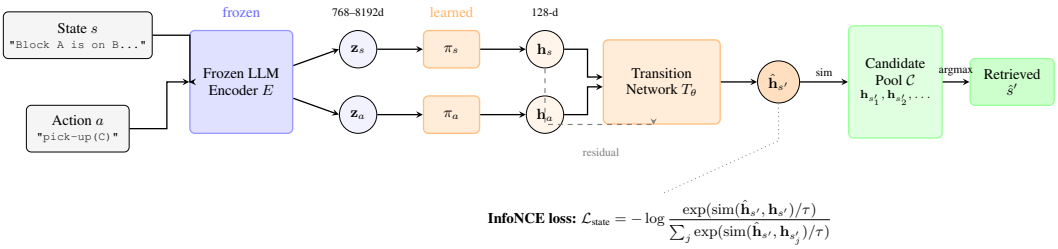


Figure 4: Complete EMBEDPLAN architecture. State and action descriptions are encoded by a frozen LLM encoder E into high-dimensional embeddings $\mathbf{z}_s, \mathbf{z}_a$. Learned projection heads π_s, π_a reduce dimensionality to a shared 128-d space. The transition network T_θ (with a residual connection from $\mathbf{h}_s$) predicts the next-state embedding $\hat{\mathbf{h}}_{s'}$, trained via InfoNCE to maximize similarity to the ground-truth embedding. At inference, the model retrieves the most similar state from a candidate pool.

BGE-M3 (BAAI/bge-m3) A multilingual embedding model (568M parameters) designed for multi-granularity retrieval. We use the dense retrieval component, which extracts 1,024-dimensional embeddings from the `[CLS]` token of the final layer.

Qwen2.5-7B-Instruct A 7-billion parameter instruction-tuned autoregressive model. We extract sentence embeddings by applying mean pooling over the final layer's hidden states, producing 3,584-dimensional embeddings.

Llama-3.3-70B-Instruct A 70-billion parameter autoregressive language model. Since decoder-only models do not have a natural sentence embedding, we apply mean pooling over the final layer's hidden states across all input tokens:

$$\mathbf{z} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{h}_t^{(L)}, \tag{22}$$

where $\mathbf{h}_t^{(L)} \in \mathbb{R}^{8192}$ is the hidden state at position t in the final layer L, and T is the sequence length. Mean pooling over decoder-only hidden states is a common approximation when dedicated sentence embeddings are unavailable (Muennighoff et al., 2023).

Rationale for freezing We freeze encoders for two reasons. First, it isolates our research question: we test whether *existing* pre-trained representations support transition learning, without confounding this with task-specific fine-tuning that might introduce planning structure. Second, it enables efficient experimentation: embeddings are computed once and cached, reducing each training run from hours of LLM inference to minutes of lightweight optimization.

Pooling strategy rationale We adopt mean pooling uniformly across decoder-only models for consistency, acknowledging that this is one of several reasonable choices. Instruction-tuned sentence encoders or last-token pooling with explicit EOS prompting may yield embeddings better suited for semantic similarity, so our results represent a lower bound on what optimized embedding extraction could achieve. We prioritize architectural consistency across encoders over per-encoder optimization, and leave pooling sensitivity analysis to future work.

B.1.2 PROJECTION HEADS

High-dimensional encoder outputs present computational and statistical challenges. We introduce learnable projection heads that map to a lower-dimensional space where transition learning occurs.

Each projection head is a multi-layer perceptron:

$$\pi(\mathbf{z}) = W_L\, \sigma\big(\text{LN}(W_{L-1}\, \sigma(\text{LN}(W_1 \mathbf{z})))\big), \tag{23}$$

where $\sigma(\cdot)$ denotes the GELU activation and $\mathrm{LN}(\cdot)$ denotes layer normalization. We use separate projection heads for states (π_s) and actions (π_a), allowing each to learn modality-specific transformations.

Default configuration

- Input dimension: encoder-dependent (768 for MPNet, 1,024 for BGE-M3, 3,584 for Qwen2.5-7B, 8,192 for Llama-3.3-70B)
- Output dimension: 128
- Hidden layers: 4
- Activation: GELU
- Normalization: LayerNorm after each hidden layer

B.1.3 TRANSITION NETWORK

We investigate two architectures embodying different hypotheses about how actions transform states.

Residual MLP (primary) Our primary architecture processes the concatenated state-action representation through a feedforward network with a residual connection:

$$\hat{\mathbf{h}}_{s'} = \mathrm{LN}\Big(f_\theta\big([\mathbf{h}_s, \mathbf{h}_a]\big) + W_{\text{res}}\,\mathbf{h}_s\Big). \tag{24}$$

The feedforward network f_θ has architecture

$$f_\theta(\mathbf{x}) = W_3\,\sigma\big(W_2\,\sigma(W_1\mathbf{x})\big), \tag{25}$$

with dimensions $W_1 : \mathbb{R}^{256} \to \mathbb{R}^{128}$, $W_2 : \mathbb{R}^{128} \to \mathbb{R}^{128}$, and $W_3 : \mathbb{R}^{128} \to \mathbb{R}^{128}$.

The residual term $W_{\text{res}}\mathbf{h}_s$ encodes an inductive bias: actions produce *incremental modifications* to states rather than wholesale replacements. Most predicates remain unchanged after a single action, and only a few flip.

Hypernetwork (alternative) An alternative hypothesis is that different actions require fundamentally different transformations. We implement this via a hypernetwork (Ha et al., 2017) that generates action-conditioned modulation parameters:

$$g_\phi(\mathbf{h}_a) = W_g^{(2)}\,\sigma\big(W_g^{(1)}\mathbf{h}_a\big) \in \mathbb{R}^{2L\cdot d_{\text{adapt}}}. \tag{26}$$

This output is split into L pairs of scale and shift vectors $(\mathbf{A}_i, \mathbf{b}_i)$ for FiLM-style conditioning (Perez et al., 2018).

B.2 TRAINING DETAILS

B.2.1 LOSS FORMULATIONS

We optimize the composite objective $\mathcal{L} = \mathcal{L}_{\text{state}} + \lambda\mathcal{L}_{\text{action}}$ with $\lambda = 2$.

The state prediction loss is InfoNCE (Oord et al., 2018) over batch elements:

$$\mathcal{L}_{\text{state}} = -\frac{1}{B}\sum_{i=1}^{B} \log \frac{\exp\big(\mathrm{sim}(\hat{\mathbf{h}}_{s'_i}, \mathbf{h}_{s'_i})/\tau\big)}{\sum_{j=1}^{B}\exp\big(\mathrm{sim}(\hat{\mathbf{h}}_{s'_i}, \mathbf{h}_{s'_j})/\tau\big)}, \tag{27}$$

where $\mathrm{sim}(\mathbf{u}, \mathbf{v}) = \mathbf{u}^\top\mathbf{v}/(\|\mathbf{u}\|\|\mathbf{v}\|)$ is cosine similarity, $\tau = 0.07$ is the temperature, and $B = 128$ is the batch size.

The action disambiguation loss contrasts predictions from different actions applied to the same state:

$$\begin{aligned}\mathcal{L}_{\text{action}} &= -\frac{1}{B}\sum_{i=1}^{B}\log\frac{\exp(z_{i,i}/\tau)}{\sum_{k=1}^{K}\exp(z_{i,k}/\tau)},\\ z_{i,k} &:= \mathrm{sim}\big(\hat{\mathbf{h}}_{s'_i}^{(a_k)},\, \mathbf{h}_{s'_i}\big),\end{aligned} \tag{28}$$

where $\{\hat{\mathbf{h}}_{s'}^{(a_k)}\}_{k=1}^{K}$ are predictions from applying each of up to $K = 50$ grounded actions to state s.

In-batch negatives The denominator of $\mathcal{L}_{\text{state}}$ treats all B states in the batch as candidates, with the $B-1$ non-matching states serving as negatives. This provides 127 negatives per positive without additional computation.

Why contrastive? Contrastive learning offers advantages over regression losses: (1) scale invariance via cosine similarity, (2) direct optimization of the retrieval objective, and (3) natural hard negative mining from same-domain states in the batch.

B.2.2 HYPERPARAMETER CONFIGURATION

Table 9 lists the hyperparameter search space and final configuration.

Table 9: Hyperparameter search space. Bold values indicate the final configuration.

Hyperparameter	**Values explored**
Architecture	
Model type	**MLP**, hypernetwork
Hidden size	**128**, 256
Number of layers	**2**, 4
Dropout	**0.0**, 0.5
Layer normalization	**Yes**
Loss function	
Action loss weight (λ)	0, 0.5, 1, 1.5, **2**, 4
Projection head	
Use projection	**Yes**, no
Projection dimension	**128**
Projection layers	2, **4**
Optimization	
Learning rate	2×10^{-3}, 1×10^{-3}, $\mathbf{4\times10^{-5}}$
Batch size	64, **128**
InfoNCE temperature (τ)	**0.07**
Weight decay	$\mathbf{1\times10^{-2}}$
Training	
Max epochs	**400**
Warmup epochs	**10**
Early stopping patience	**100**

B.3 EVALUATION PROCEDURE

Hit@k computation For each test transition (s, a, s'):

1. Compute the predicted embedding $\hat{\mathbf{h}}_{s'} = T_\theta(\pi_s(\mathbf{z}_s), \pi_a(\mathbf{z}_a))$.
2. Compute the similarity to all candidates, $\text{sim}_j = \text{sim}(\hat{\mathbf{h}}_{s'}, \mathbf{h}_{s'_j})$ for $s'_j \in \mathcal{C}$.
3. Rank candidates by similarity in descending order.
4. Record a hit if the correct s' appears in the top k.

The candidate pool $\mathcal{C}$ contains 128 states: the ground-truth next state plus 127 distractors sampled according to the evaluation protocol (uniformly from the domain for Interpolation, and from the same problem instance for Extrapolation).

Tie-breaking When multiple candidates have identical similarity scores, we select one of them uniformly at random. The reference-method comparison of Appendix L instead uses worst-case tie-breaking, which is required for its null controls to be meaningful (Appendix L.2). Learned representations produce essentially no exact ties, so the convention does not affect their scores.

Action disambiguation Given a state s and ground-truth next state s', we apply every grounded action applicable in s, $\mathcal{A}(s)$, and compare each prediction with s':

$$\hat{a} = \arg\max_{a \in \mathcal{A}(s)} \operatorname{sim}\big(T_\theta(\mathbf{h}_s, \mathbf{h}_a), \mathbf{h}_{s'}\big). \tag{29}$$

Acc@1 measures how often $\hat{a}$ matches the true action. More generally, Acc@k ranks all $a \in \mathcal{A}(s)$ by this similarity and measures how often the true action is among the top k.

B.4 Candidate pool construction and size

Every Hit@k in this paper is measured against a candidate pool, so the pool is part of the metric. Two aspects matter independently: how distractors are *drawn*, and how *many* there are.

Distractor difficulty The protocols of Appendix A already vary the first aspect. Under Interpolation distractors are drawn uniformly from all domain states, so most share neither objects nor goal with the query. Under Extrapolation they are drawn exclusively from the query's own problem instance, so every candidate shares the object vocabulary and differs from the target by a handful of predicates. Under the Plan-Variant protocol the competing candidates are the successor states of alternative optimal plans, which are reachable and often differ by a single literal. These three settings span easy, hard, and adversarial distractors at a fixed pool size of 128.

Pool size The second aspect is held fixed at 128 throughout the main results, which is a controlled measurement rather than a deployment setting. We therefore sweep the pool from 128 states to every state observed in the domain on Ferry and Logistics, two domains whose observed state sets differ by more than $3\times$ (46,205 and 13,373 states). The split is Interpolation and the encoder is Llama-3.3-70B. The transition model is unchanged and only the retrieval pool grows.

Table 10: Effect of candidate pool size under Interpolation, Llama-3.3-70B, mean $\pm$ SE over 3 seeds. "Full" retrieves against every observed state in the domain. The inflation column is the 128-way value divided by the full-pool value.

Domain	Metric	128	512	2,048	8,192	Full	Infl.
Ferry 46,205 states	Hit@1	98.96 ± 0.09	97.21 ± 0.27	90.90 ± 0.55	76.58 ± 0.92	35.23 ± 1.14	$2.81\times$
	Hit@5	99.97 ± 0.03	99.85 ± 0.08	99.38 ± 0.09	97.97 ± 0.25	86.93 ± 1.36	$1.15\times$
	Hit@10	99.99 ± 0.01	99.94 ± 0.03	99.73 ± 0.08	99.03 ± 0.14	94.96 ± 0.63	$1.05\times$
Logistics 13,373 states	Hit@1	96.29 ± 0.23	88.21 ± 0.65	70.12 ± 1.83	47.20 ± 1.60	32.55 ± 1.69	$2.96\times$
	Hit@5	99.85 ± 0.08	99.37 ± 0.12	96.73 ± 0.31	83.13 ± 0.93	74.89 ± 1.71	$1.33\times$
	Hit@10	99.90 ± 0.05	99.72 ± 0.09	98.84 ± 0.11	92.99 ± 0.50	88.01 ± 0.92	$1.14\times$

The premise holds: the 128-way pool inflates Hit@1 by $2.8\times$ on Ferry and $3.0\times$ on Logistics. Hit@k for larger k degrades far more gracefully: $1.15\times$ and $1.33\times$ at k=5, and $1.05\times$ and $1.14\times$ at k=10. Against *every* observed state the model still places the correct successor in the top 5 for 86.9% of Ferry transitions and 74.9% of Logistics transitions, and in the top 10 for 95.0% and 88.0%. This is the empirical basis for emphasizing Hit@5 in Section 3: top-5 retrieval largely survives removing the 128-way pool, whereas strict top-1 retrieval is substantially an artifact of it.

Logistics degrades faster in relative terms at intermediate sizes because $8{,}192$ candidates already cover 61% of its state space, against 18% of Ferry's. At the full pool the two domains converge to within three points of Hit@1 of each other.

When no pool is available The retrieval formulation presupposes an enumerable candidate set, which classical planning benchmarks and discrete-observation model-based RL provide but open-ended settings do not. The transition function itself does not depend on this: it maps $(\mathbf{z}_s, \mathbf{z}_a)$ to a predicted next-state embedding and can be applied recursively to its own output, yielding a latent rollout with no pool at any step. Retrieval is then needed only where the trajectory must be read back out as text. We report closed-loop rollouts with and without per-step snapping to a real state in Appendix D.8. With snapping, rollout retains 92–99% of teacher-forced accuracy. Without it, the latent drifts off the state manifold and error compounds, which is the cost of dropping the pool entirely.

B.5 COMPUTE, LATENCY, AND COST ANALYSIS

All experiments were conducted on the following infrastructure:

- **GPU**: NVIDIA A100 80 GB
- **CPU**: AMD EPYC 7763 64-core processor
- **Memory**: 512 GB RAM
- **Framework**: PyTorch 2.1, CUDA 12.1

Training time Training EMBEDPLAN is efficient because the heavy semantic processing is offloaded to the frozen encoder. Generating and caching embeddings represents the bulk of the preparation time, after which optimizing the lightweight transition network takes only minutes:

- Embedding extraction (per domain, Llama-3.3-70B): $\sim$2 hours
- Embedding extraction (per domain, MPNet): $\sim$5 minutes
- Transition network training (per domain): $\sim$15 minutes
- Full experimental suite (all encoders, all protocols): $\sim$72 hours

Inference latency and efficiency A core motivation for EMBEDPLAN is bypassing the token-by-token generation bottleneck of standard LLM planning. To quantify this, we benchmarked the wall-clock latency of EMBEDPLAN against standard LLM API generation. As detailed in Table 11, even when accounting for the text-to-vector encoding overhead (full pipeline), our method is roughly $100\times$ faster than standard LLM generation. When embeddings for the state-action space are precomputed or batched, as is common in classical planning search algorithms, retrieval-based prediction drops to a fraction of a millisecond.

Table 11: Wall-clock latency and computational cost comparison. The full pipeline includes text-to-vector encoding time with BGE-M3. EMBEDPLAN eliminates the sequential token bottleneck of autoregressive generation. †Projection heads, transition network, and scoring of 128 candidates with cached encoder outputs (Llama-3.3-70B dimensions). The full-pipeline latency additionally includes encoding the query texts.

	EMBEDPLAN		
Metric	**Full pipeline**	**Batched/cached**	**LLM API**
Latency	18.6 ms	0.168 ms	1,888.2 ms
Speedup vs. LLM	$101\times$	$11{,}215\times$	$1\times$
Compute per transition	$\sim$286M FLOPs†	$\sim$286M FLOPs†	198 tokens

Matched candidate-ranking comparison The generation baselines of Table 3 solve a harder task than retrieval. To compare like with like, we give GPT-5.4 the identical ranking problem: the Interpolation query (s, a) together with the same 128 candidates EMBEDPLAN ranks, from which it must select the successor. GPT-5.4 selects the correct successor for 44% of Ferry and 92% of Logistics queries, against 99.0% and 96.3% Hit@1 for EMBEDPLAN (Table 10, with Logistics Hit@1 ranging over 91.2–96.3% across our runs), at roughly four orders of magnitude lower per-transition cost.

Training cost and break-even Training is a real tradeoff and is not amortized across domains, since cross-domain transfer fails (Section 4). Per domain, EMBEDPLAN needs (s, a, s') transitions, a one-time embedding pass, and $\sim$15 minutes of training. Against the measured $1{,}888$ ms autoregressive baseline it costs 0.168 ms per transition, so it breaks even after roughly $4{,}300$ transitions with Llama-3.3-70B as the encoder, or 640 with MPNet. The difference is the one-time embedding pass, which is far costlier with a 70B encoder. A single search expands 10^3 to 10^4 successors, so the method pays off for a domain that will be planned in repeatedly, not for one seen once.

Carbon footprint Estimated total compute for training and evaluation is $\sim$200 GPU-hours on A100. Using a carbon intensity of 0.4 kg CO_2/kWh and an A100 TDP of 400W, the estimated emissions are $\sim$32 kg CO_2.

C Dataset

C.1 Dataset statistics

We use nine classical PDDL domains from planning benchmarks (Kokel et al., 2025). Table 12 summarizes the dataset. Our transition datasets are derived from ACPBench domains but involve additional processing not included in the publicly released dataset. We will release our processed data upon acceptance.

Table 12: Dataset statistics by domain.

Domain	Problems	States	Transitions	Actions
Blocksworld	5	43,551	43,065	4
Depot	7	5,795	13,256	5
Ferry	10	46,205	225,300	3
Floortile	6	33,608	166,565	6
Goldminer	7	12,237	52,023	7
Grid	5	8,671	664,346	5
Logistics	7	13,373	46,866	6
Rovers	10	46,783	509,457	9
Satellite	10	49,204	1,248,696	4
Total	**67**	**259,427**	**2,969,574**	

C.2 State representation

States are rendered as natural language descriptions containing the current predicate values. An example from Blocksworld:

"Block A is on the table. Block B is on Block A. Block C is clear. The robotic arm is empty."

Actions are parameterized strings (e.g., `pick-up(BlockC)`, `stack(BlockA, BlockB)`). Note that the state count in Table 12 counts unique state *descriptions*: the terminal state of each plan contributes a state but no transition, which is why some domains list more states than transitions.

C.3 Domain descriptions

Blocksworld A robotic arm must rearrange colored blocks into a specified goal configuration. Only clear blocks (with nothing on top) can be moved. This domain is known for simple rules yet rich combinatorial complexity.

Depot Combines logistics and block stacking. Crates must be moved between depots using trucks for transportation and hoists for stacking and unstacking.

Ferry A ferry boat transports cars between locations. The ferry can carry only one car at a time, requiring optimization of loading and unloading sequences.

Floortile Robots paint the tiles of a grid-shaped floor according to a target pattern. A robot paints the tile above or below it, can switch between two colors, and cannot move onto a painted tile, so the painting order matters.

Goldminer A robot in a grid-shaped mine must reach and pick up the gold. A laser clears both hard and soft rock but destroys the gold if fired at it, whereas bombs clear only soft rock.

Grid An agent moves on a 2D grid to reach target locations, potentially with obstacles restricting movement.

Logistics Packages must be delivered within and across cities. Trucks handle intra-city transport, and airplanes handle inter-city deliveries.

Rovers Planetary exploration with multiple rovers collecting samples, taking images, and transmitting data. Rovers have specialized equipment and must communicate with a base station.

Satellite Satellites turn to point at targets and take images in given modes with onboard instruments, which must be switched on (one per satellite at a time) and calibrated first.

C.4 TRANSITION EXAMPLES

Figures 5–8 show representative transitions from four domains.

Blocksworld

State s: Block block_2 is on the table, Block block_3 is located on the table, No blocks are placed on top of block_1, The block block_2 is currently situated under the block block_1, The robotic arm is not holding anything, and Block block_3 is clear.

Action a: `(pick-up block_3)`

Result s': Block block_1 is clear, Block block_2 is located on the table, The robotic arm is holding block_3, and The block block_2 is currently situated under the block block_1.

Figure 5: Representative transition from Blocksworld.

Ferry

State s: Car c0 is at location l0, Car c2 is at location l1, The ferry is at l0, and Car c1 is on the ferry.

Action a: `(debark c1 l0)`

Result s': Car c0 is at location l0, Car c1 is at location l0, The ferry is at l0, The ferry is empty, and Car c2 is at location l1.

Figure 6: Representative transition from Ferry.

Logistics

State s: p3 is in t1, a0 is at l0-0, t1 is at l1-0, p0 is at l1-0, p1 is in t1, t0 is at l0-0, and p2 is in a0.

Action a: `(fly-airplane a0 l0-0 l1-0)`

Result s': t1 is at l1-0, p0 is at l1-0, p1 is in t1, p3 is in t1, t0 is at l0-0, a0 is at l1-0, and p2 is in a0.

Figure 7: Representative transition from Logistics.

Rovers

State s: Store(s) store0 is empty, Channel general is free, Image objective1 was communicated in mode colour, Rover rover0 has image objective1 in mode colour, Rover rover1 has soil analyzed in waypoint waypoint0, Rover rover1 is available, Rover rover1 is at waypoint0, Rocks can be sampled at the following location(s): waypoint0, Store(s) store1 is empty, Rover rover0 is available, and Rover rover0 is at waypoint2.

Action a: `(communicate_soil_data rover1 general waypoint0 waypoint0 waypoint1)`

Result s': Store(s) store0 is empty, Channel general is free, Rover rover0 has image objective1 in mode colour, Image objective1 was communicated in mode colour, Rover rover1 has soil analyzed in waypoint waypoint0, Rover rover1 is available, Rover rover1 is at waypoint0, Rocks can be sampled at the following location(s): waypoint0, Soil data was communicated from waypoint waypoint0, Store(s) store1 is empty, Rover rover0 is available, and Rover rover0 is at waypoint2.

Figure 8: Representative transition from Rovers.

D Extended results

D.1 Per-domain results

Table 13 provides a complete per-domain breakdown across encoders and evaluation protocols.

Table 13: Per-domain Hit@5 (%) for the Interpolation and Extrapolation splits.

	Qwen2.5-7B		**Llama-3.3-70B**	
Domain	**Interp.**	**Extrap.**	**Interp.**	**Extrap.**
Blocksworld	100.0	41.6±8.7	100.0	49.1±10
Depot	98.2	24.8±5.2	98.8	25.9±6
Ferry	99.9	36.7±0.6	100.0	40.6±3
Floortile	99.4	55.2±11	99.6	68.8±16
Goldminer	99.9	74.4±5.8	100.0	76.2±5
Grid	98.6	62.7±5.2	99.8	74.9±1
Logistics	99.6	44.7±12	99.9	53.7±10
Rovers	99.7	49.2±1.7	99.0	54.4±3
Satellite	99.9	40.0±0.6	99.9	47.5±6
Mean	99.5	47.7±4.9	99.7	54.6±5.5

D.2 Full performance metrics

Table 15 reports Hit@1/5/10 and action accuracy under the Extrapolation split.

The gap between Hit@5 (54.6%) and Acc@5 (16.2%) reveals that models predict correct next states without fully capturing causal action structure.

D.3 Cross-Domain transfer matrix

Table 16 shows the complete 9×9 transfer matrix.

The only notable transfer is Ferry→Logistics (22.3%), which we attribute to shared transportation semantics in state descriptions.

D.4 Leave-One-Out results

Despite training on eight diverse domains, LOO performance (9.2%) barely exceeds the untrained model (4.2%).

Table 14: Single-step transition prediction (Interpolation split, Llama-3.3-70B). *Next state prediction*: whether the ground-truth next state is in the top-k retrieved candidates. *Action disambiguation*: whether, among all actions applied to s, the correct action a produces the prediction closest to ground-truth s'. Mean Acc@5 is 85.3%.

	Next state prediction (%)			Action disambiguation (%)		
Domain	**Hit@1**	**Hit@5**	**Hit@10**	**Acc@1**	**Acc@5**	**Acc@10**
Blocksworld	94.6 ± 0.4	100.0 ± 0.0	100.0 ± 0.0	25.9 ± 0.9	88.6 ± 0.2	99.5 ± 0.1
Depot	76.9 ± 1.2	98.8 ± 0.1	99.4 ± 0.1	29.8 ± 2.3	82.2 ± 2.1	93.3 ± 1.4
Ferry	98.5 ± 0.2	100.0 ± 0.0	100.0 ± 0.0	43.1 ± 0.8	96.8 ± 0.1	99.8 ± 0.0
Floortile	97.9 ± 0.4	99.6 ± 0.0	99.8 ± 0.0	37.1 ± 2.8	86.5 ± 1.9	97.3 ± 0.8
Goldminer	93.8 ± 0.8	100.0 ± 0.0	100.0 ± 0.0	17.0 ± 0.5	73.3 ± 0.6	97.4 ± 0.2
Grid	90.1 ± 0.2	99.8 ± 0.1	99.9 ± 0.0	28.6 ± 0.2	86.7 ± 0.1	96.4 ± 0.3
Logistics	93.4 ± 0.6	99.9 ± 0.1	99.9 ± 0.0	25.1 ± 2.0	81.3 ± 3.8	94.0 ± 1.9
Rovers	86.0 ± 0.8	99.0 ± 0.1	99.6 ± 0.1	26.1 ± 1.5	73.9 ± 1.4	86.3 ± 0.7
Satellite	97.6 ± 0.2	99.9 ± 0.0	100.0 ± 0.0	55.0 ± 1.7	98.5 ± 0.3	99.9 ± 0.0
Mean	92.1 ± 2.3	99.7 ± 0.1	99.8 ± 0.1	32.0 ± 3.8	85.3 ± 2.9	96.0 ± 1.4

Table 15: Full metrics (Llama-3.3-70B, Extrapolation split). Action accuracy measures whether the correct action is identified given (s, s').

	State prediction			Action accuracy		
Domain	**Hit@1**	**Hit@5**	**Hit@10**	**Acc@1**	**Acc@5**	**Acc@10**
Blocksworld	17.6±5.2	49.1±9.8	64.6±7.5	0.7±0.1	7.9±1.2	24.0±2.3
Depot	4.7±1.2	25.9±6.4	41.2±8.1	0.7±0.2	7.8±1.7	21.8±2.3
Ferry	12.0±1.7	40.6±3.5	58.1±4.0	1.1±0.4	10.3±1.2	23.7±2.3
Floortile	37.6±13	68.8±16	78.9±13	3.2±0.6	28.3±8.7	52.3±14
Goldminer	35.8±6.4	76.2±5.2	88.0±1.7	3.0±0.2	16.5±1.2	45.7±3.5
Grid	25.7±1.2	74.9±1.2	88.0±1.2	8.0±2.3	50.0±4.6	73.9±5.8
Logistics	16.5±3.5	53.7±9.8	70.8±9.8	0.6±0.3	8.4±1.7	25.3±1.7
Rovers	16.9±1.2	54.4±2.9	72.2±2.3	0.9±0.1	7.1±0.2	16.5±1.7
Satellite	14.4±2.3	47.5±5.8	66.9±5.8	0.9±0.2	9.1±1.2	23.5±4.0
Mean	20.1±3.6	54.6±5.5	69.9±4.9	2.1±0.8	16.2±4.8	34.1±6.4

D.5 Multi-Domain results

D.6 Untrained baseline

To establish a performance floor, we evaluated the transition function with randomly initialized weights.

The untrained model (4.2% Hit@5, close to the 3.9% chance level of 5 in 128) confirms the retrieval task's inherent difficulty and that learned performance results from actual dynamics learning.

D.7 Plan-level evaluation across encoders

We extend our evaluation to the trajectory level to assess whether high transition accuracy translates to reliable multi-step planning. We compare two splitting strategies:

- **Plan-Variant**: test plans come from problem instances seen during training, but the specific plans are held out.
- **Extrapolation**: test plans come from entirely new problem instances never seen during training.

We report two metrics:

- **Mean trajectory Hit@5**: the average Hit@5 across all steps in a trajectory.

Table 16: Cross-Domain transfer (Llama-3.3-70B). Hit@5 (%) training on the row domain and testing on the column domain. Chance: 3.9% (5 of 128). Untrained model: 4.2%.

	Block	**Depot**	**Ferry**	**Floor**	**Gold**	**Grid**	**Logis**	**Rover**	**Satel**	**Mean**
Blocksworld		5.7	5.5	6.7	6.1	6.5	9.2	5.1	5.3	6.3
Depot	4.3		5.3	4.8	5.0	4.9	6.0	7.5	5.4	5.4
Ferry	5.3	8.7		9.3	5.2	9.0	**22.3**	6.6	5.6	9.0
Floortile	5.0	7.3	7.0		6.1	6.5	7.5	10.0	7.9	7.2
Goldminer	4.2	5.4	5.7	5.0		7.5	5.4	5.9	4.9	5.5
Grid	5.4	6.6	8.6	9.0	10.4		14.3	5.9	5.4	8.2
Logistics	4.3	5.5	7.7	4.9	7.0	6.3		5.1	4.5	5.7
Rovers	4.3	5.7	5.0	8.4	4.6	5.9	4.8		8.2	5.9
Satellite	4.5	7.5	6.3	7.8	6.4	5.2	10.1	5.9		6.7
Mean	4.7	6.5	6.4	7.0	6.4	6.5	9.9	6.5	5.9	**6.6**

Table 17: Leave-One-Out results (Llama-3.3-70B). Train on eight domains, test on the held-out domain.

Held-out domain	**Hit@1**	**Hit@5**	**Hit@10**
Logistics	3.3±0.4	15.8±2.2	27.6±3.3
Grid	2.3±0.2	12.3±0.8	22.9±1.4
Rovers	2.6±0.3	12.6±1.1	22.6±1.6
Ferry	1.8±0.1	9.0±0.5	17.2±0.8
Satellite	1.6±0.1	8.4±0.8	16.1±1.5
Floortile	1.7±0.3	7.9±1.1	14.9±2.0
Goldminer	1.3±0.0	6.4±0.2	12.5±0.3
Depot	1.1±0.1	5.6±0.5	10.9±0.9
Blocksworld	1.0±0.0	5.2±0.1	10.3±0.3
Mean	1.9±0.3	9.2±1.2	17.2±2.0

- **Exact trajectory Hit@5**: the percentage of trajectories in which *every* step is correctly retrieved.

Table 20 reports the full results for Llama-3.3-70B, and Table 21 compares the three encoders. We observe a consistent trajectory generalization gap across all models. Under the Plan-Variant split, models achieve moderate reliability, though exact-match rates remain low due to error accumulation. Under Extrapolation, performance collapses. For Blocksworld, the exact trajectory rate falls to 0.0% across all encoders. This indicates that the transition model's planning capability is largely confined to seen problem manifolds. When forced to extrapolate to new problems, the probability of executing a valid multi-step plan drops to near zero.

Key observations

- **Plan-Variant performance is comparable for the two larger encoders**: Llama-3.3-70B and BGE-M3 achieve similar mean Hit@5 (∼51–53%), while MPNet lags substantially (24.3%).
- **Extrapolation collapse is universal**: all encoders degrade sharply under Extrapolation, with exact Hit@5 at most 3.4% on average.
- **Domain-specific patterns persist**: Goldminer and Grid extrapolate best (Table 20), while Blocksworld fails for every encoder.
- **MPNet struggles even under Plan-Variant**: Rovers and Satellite show near-zero performance, suggesting these domains require higher-capacity embeddings.

Table 18: Multi-Domain unified model (Llama-3.3-70B, Extrapolation split).

Domain	Hit@5	Domain	Hit@5
Floortile	52.0±10	Blocksworld	36.9±13
Rovers	51.9±5	Satellite	34.3±7
Goldminer	46.0±8	Grid	32.6±1
Ferry	37.7±14	Logistics	24.8±10
		Depot	18.9±12
Mean: 37.2±3.8 (vs. 54.6 single-domain)			

Table 19: Untrained baseline (random weights, Llama-3.3-70B).

Domain	Hit@1	Hit@5	Hit@10
Blocksworld	0.8±0.1	4.0±0.2	8.0±0.4
Depot	0.9±0.2	3.9±0.0	8.0±0.4
Ferry	0.7±0.1	4.3±0.6	8.4±0.9
Floortile	0.8±0.0	4.0±0.1	8.0±0.1
Goldminer	1.0±0.4	4.8±1.4	8.9±2.0
Grid	0.9±0.1	4.4±0.3	8.0±0.2
Logistics	0.9±0.1	4.1±0.2	8.6±0.6
Rovers	0.7±0.0	4.1±0.2	8.4±0.1
Satellite	0.7±0.0	3.9±0.1	7.8±0.1
Mean	0.8±0.0	4.2±0.1	8.2±0.1

D.8 CLOSED-LOOP ROLLOUT

The plan-level results above are teacher-forced: the true state is supplied at every step, so errors cannot accumulate. Here we remove that correction. All three regimes predict the next state from the current state and action and differ only in what is fed to the next step.

- **Teacher-forced**: the true state s_t (the setting of Appendix D.7).
- **Closed loop, with snapping**: the model's own prediction, snapped to the candidate pool (its top-1 retrieved state) and fed back as the next input. If retrieval picks a wrong state, that wrong state is propagated, so errors compound.
- **Closed loop, without snapping**: the model's own prediction as a raw latent, never snapped. No candidate pool is consulted at any intermediate step.

The split is Interpolation and the retrieval pool has 1,000 candidates. Aggregate step-wise Hit@1 appears in Table 4: with snapping, the rollout retains 99% (Ferry), 92% (Logistics), and 98% (Blocksworld) of teacher-forced accuracy.

Depth isolates the mechanism Table 22 breaks the Ferry rollout down by depth. The three regimes coincide at depth 1 by construction, before any feedback exists. From depth 2 they separate: snapped rollout stays flat, while unsnapped rollout collapses (52.4%, 20.9%, 10.3%). The collapse tracks a single measurable quantity, the drift of the predicted latent off the manifold of real state embeddings: its cosine similarity to the nearest real state is 1.00 at the input, 0.83 after one step, and $\sim$0.55 once unrolled further. Weighted over depths 2–4, snapped rollout retains 98% of teacher-forced Hit@1 (94.1% vs. 96.0%) and unsnapped rollout 49% (46.7%). Per-step snapping to that manifold is therefore what keeps multi-step rollout accurate. Rollout without snapping is limited by transition-function fidelity, which is where further gains should be sought.

Horizon Test plans in this benchmark are short: of the 300 Ferry trajectories, 243 have two steps and 7 reach four (mean 2.2). Depth 3 ($n = 57$) is the deepest row on which we rest a claim, and there snapped rollout holds at 92.4% against 20.9% without snapping. What is demonstrated is that

Table 20: Plan-level evaluation (Llama-3.3-70B). Trajectory metrics under the Plan-Variant and Extrapolation splits.

	Plan-Variant		Extrapolation	
Domain	**Mean Hit@5**	**Exact Hit@5**	**Mean Hit@5**	**Exact Hit@5**
Blocksworld	38.7 ± 0.8	1.6 ± 0.2	3.7 ± 1.3	0.0 ± 0.0
Depot	30.1 ± 2.6	11.2 ± 1.5	2.4 ± 1.7	1.0 ± 1.0
Ferry	71.1 ± 1.1	39.1 ± 1.6	2.0 ± 0.7	0.5 ± 0.2
Floortile	74.1 ± 2.7	33.6 ± 3.3	11.6 ± 4.4	2.8 ± 1.2
Goldminer	55.4 ± 2.2	23.8 ± 1.6	18.5 ± 6.2	5.5 ± 1.8
Grid	55.0 ± 2.4	37.4 ± 2.2	16.6 ± 3.7	8.2 ± 2.1
Logistics	52.8 ± 0.6	14.1 ± 2.0	22.7 ± 8.6	9.6 ± 4.4
Rovers	26.5 ± 0.8	10.0 ± 0.7	12.2 ± 3.5	1.9 ± 0.8
Satellite	56.8 ± 1.2	32.2 ± 1.0	1.2 ± 0.4	0.5 ± 0.1
Mean	51.2 ± 5.5	22.6 ± 4.6	10.1 ± 2.7	3.3 ± 1.2

Table 21: Plan-level evaluation across encoders. *Top*: Plan-Variant trajectory metrics for BGE-M3 and MPNet (Llama-3.3-70B in Table 20). *Bottom*: Extrapolation, mean $\pm$ SE over the six domains with an independent run for every encoder (Blocksworld, Depot, Ferry, Floortile, Goldminer, Grid).

	BGE-M3		MPNet	
Plan-Variant	**Mean Hit@5**	**Exact Hit@5**	**Mean Hit@5**	**Exact Hit@5**
Blocksworld	23.7 ± 1.5	0.2 ± 0.1	16.7 ± 0.3	0.3 ± 0.1
Depot	45.5 ± 1.0	20.9 ± 0.1	31.1 ± 1.6	10.8 ± 1.1
Ferry	63.3 ± 1.4	27.8 ± 2.1	60.9 ± 2.0	29.1 ± 2.3
Floortile	49.2 ± 1.0	13.1 ± 1.1	9.2 ± 0.5	3.0 ± 0.4
Goldminer	67.9 ± 2.6	35.0 ± 3.2	26.8 ± 0.8	10.5 ± 0.5
Grid	51.9 ± 1.5	34.0 ± 1.2	32.7 ± 2.7	17.5 ± 2.6
Logistics	67.2 ± 0.3	25.6 ± 1.6	41.3 ± 2.2	9.9 ± 1.9
Rovers	60.5 ± 0.2	24.7 ± 0.2	0.1 ± 0.0	0.0 ± 0.0
Satellite	51.2 ± 0.4	27.5 ± 0.6	0.2 ± 0.1	0.1 ± 0.0
Mean	53.4 ± 4.6	23.2 ± 3.6	24.3 ± 6.7	9.0 ± 3.2

Extrapolation (six domains)	**Mean Hit@5**	**Exact Hit@5**
Llama-3.3-70B	9.1 ± 3.0	3.0 ± 1.3
BGE-M3	8.2 ± 3.4	3.4 ± 1.6
MPNet	4.6 ± 2.7	2.0 ± 1.2

snapping sustains rollout over these short test trajectories, not over long horizons. Combining the snapped transition function with beam or tree search over longer horizons is the natural next step.

Table 22: Closed-loop rollout by depth on Ferry. Step-wise Hit@1 in % (mean ± SE), where n is the number of test trajectories reaching that depth. Same models, trajectories, and pool as Table 4, and the depth rows weighted by n reproduce the aggregate to rounding.

Depth	n	True state at each step	Snapped	Not snapped
1	300	94.3 ± 0.7	94.3 ± 0.7	94.3 ± 0.7
2	300	96.3 ± 0.5	94.4 ± 0.7	52.4 ± 3.4
3	57	94.2 ± 1.9	92.4 ± 0.5	20.9 ± 2.4
4	7	96.7 ± 4.7	93.3 ± 9.4	10.3 ± 7.4

E STATISTICAL ANALYSIS

E.1 MAIN GENERALIZATION GAP

Table 23 reports the interpolation-extrapolation gap.

Table 23: Statistical analysis of the generalization gap (Qwen2.5-7B, paired across the nine domains).

Statistic	Value
Interpolation (mean ± SE)	99.5% ± 0.2%
Extrapolation (mean ± SE)	47.7% ± 4.9%
Gap	51.8 pp
Paired t-test	$t(8) = 10.58$
p-value	5.57×10^{-6}
95% CI	[40.7, 62.9] pp

E.2 EFFECT SIZES FOR KEY COMPARISONS

Table 24: Effect sizes for major findings. The encoder column is given explicitly because the comparisons are drawn from different encoder runs.

Comparison	Encoder	Δ	Cohen's d	p
Interpolation vs. Extrapolation	Qwen2.5-7B	−51.8 pp	5.25	$< 10^{-5}$
Extrapolation vs. Cross-Domain	Llama-3.3-70B	−48.0 pp	4.32	$< 10^{-5}$
Llama vs. MPNet (Extrap.)	both	+27.8 pp	2.53	< 0.001
Single vs. Multi (Extrap.)	Llama-3.3-70B	+17.4 pp	1.42	0.028

E.3 PER-DOMAIN STATISTICAL TESTS

Table 25 reports per-domain tests.

All comparisons remain significant after Bonferroni correction ($\alpha = 0.05/9 = 0.0056$) except Logistics, Floortile, and Goldminer, which are significant only at the uncorrected $\alpha = 0.05$.

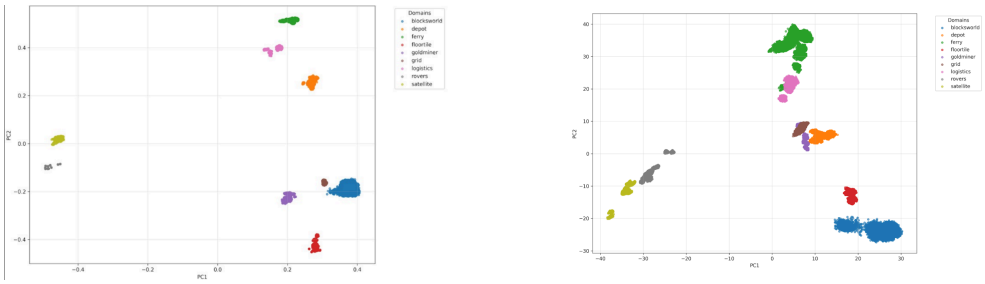
Table 25: Independent t-tests comparing interpolation and extrapolation per domain (Qwen2.5-7B).

Domain	**Gap (pp)**	**t-statistic**	**p-value**
Depot	73.4	12.87	1.17×10^{-4}
Ferry	63.3	65.49	3.24×10^{-6}
Satellite	59.9	153.39	4.11×10^{-8}
Blocksworld	58.4	5.95	2.70×10^{-3}
Logistics	55.0	4.28	8.28×10^{-3}
Rovers	50.5	20.22	8.90×10^{-6}
Floortile	44.2	3.68	1.57×10^{-2}
Grid	35.9	5.38	2.40×10^{-3}
Goldminer	25.5	3.54	1.13×10^{-2}

F Embedding Analysis

F.1 PCA visualization

Figure 9 compares embedding geometry across model scales.



(A) MPNet (110M) (B) Llama-3.3-70B

Figure 9: Embedding space fragmentation across scales. PCA of frozen state embeddings, colored by domain. **(A)** MPNet places each domain in its own tight, isolated cluster. **(B)** Llama-3.3-70B, despite being roughly 640× larger, shows the same fragmentation. Pre-trained embeddings thus group states by domain-specific surface features rather than by abstract planning roles, regardless of model scale.

Both MPNet and Llama place each domain in an isolated cluster, often split further into sub-clusters. While Llama shows slightly more spread within clusters, the critical structural limitation remains: the manifolds of different domains are disjoint. Scaling up model parameters does not automatically induce abstract, problem-invariant representations.

F.2 Domain complexity analysis

We find no clean relationship between the surface complexity of a domain and its generalization gap. Rovers has by far the largest action and predicate counts yet a mid-range gap (50.5 pp), while Ferry is among the simplest domains and shows one of the largest gaps (63.3 pp). This is consistent with our main claim: what limits extrapolation is how the embedding space organizes states, not how large or intricate the domain is.

Table 26: Domain complexity metrics and generalization gaps (Qwen2.5-7B). Action counts are action schemas as defined in the PDDL domain file, matching Table 12.

Domain	Actions	Predicates	Gap
Depot	5	8	73.4
Logistics	6	6	55.0
Satellite	4	8	59.9
Blocksworld	4	5	58.4
Ferry	3	5	63.3
Rovers	9	26	50.5
Floortile	6	10	44.2
Grid	5	9	35.9
Goldminer	7	7	25.5

G Ablations

G.1 Architecture and encoder comparison

Table 27: Architecture comparison for the two largest encoders (Extrapolation split, Hit@5 %).

Encoder	Parameters	MLP	Hypernetwork
Llama-3.3-70B	70B	54.6±5.5	53.8±5.5
Qwen2.5-7B	7B	47.7±4.9	46.2±5.0

Architecture choice has minimal impact on performance: the residual MLP matches or slightly exceeds the hypernetwork for both encoders, by at most 1.5 pp. The generalization gap is consistent across architectures, which points to the embedding rather than the transition network design as the limitation.

Larger encoders yield better extrapolation performance, but the improvement is sublinear: a roughly 640× increase in parameters (MPNet to Llama) yields only a 2.0× improvement in Hit@5 (Table 6). This suggests that scale alone does not resolve the underlying structural limitation.

G.2 Effect of the action disambiguation loss

We ablate the contribution of the action disambiguation loss $\mathcal{L}_{\text{action}}$ by comparing models trained with the full composite objective ($\lambda = 2$) against models trained with the state prediction loss only ($\lambda = 0$).

The action disambiguation loss yields improvements across both metrics and all nine domains. The direct target, Action Acc@5, shows the largest gain, improving 3.4× from 4.8% to 16.2%. Without explicit supervision on action effects, models fail to distinguish between actions with similar but distinct consequences: Action Acc@5 under $\lambda = 0$ barely exceeds the untrained baseline in most domains, indicating that the state prediction loss alone provides little signal for learning action semantics.

Hit@5 also improves by 19.3 pp (+55% relative), despite $\mathcal{L}_{\text{action}}$ not directly optimizing this metric. We suggest two mechanisms for this indirect benefit: (1) without action-contrastive training, the model may exploit spurious correlations between surface-level state-action features and outcomes, which fail to transfer to unseen problems, and (2) the action loss pushes the network to encode transformation patterns that separate an action's effect from its object binding, for example distinguishing `pick-up(A)` from `pick-up(B)`.

The improvement is consistent across domains but varies in magnitude. Grid shows the largest absolute gain in Action Acc@5 (+31.3 pp), likely because its spatial action semantics (movement in cardinal directions) are highly distinctive when explicitly supervised. Depot and Logistics show

Table 28: Ablation of the action disambiguation loss. Comparison of models trained without (λ=0) and with (λ=2) action disambiguation under Extrapolation evaluation (Llama-3.3-70B). The action loss yields gains in both state prediction (+19.3 pp) and action accuracy (3.4×).

	Hit@5 (%)		**Action Acc@5 (%)**	
Domain	λ=0	λ=2	λ=0	λ=2
Blocksworld	30.2±8	49.1±10	1.8±0.4	7.9±2
Depot	13.4±4	25.9±6	1.6±0.3	7.8±3
Ferry	24.8±3	40.6±3	2.5±0.5	10.3±2
Floortile	45.3±14	68.8±16	8.2±3	28.3±15
Goldminer	55.7±6	76.2±5	4.9±1.2	16.5±2
Grid	52.1±2	74.9±1	18.7±5	50.0±8
Logistics	31.5±9	53.7±10	1.9±0.6	8.4±3
Rovers	35.2±4	54.4±3	1.7±0.3	7.1±0.4
Satellite	29.8±5	47.5±6	2.2±0.5	9.1±2
Mean	35.3±4.5	54.6±5.5	4.8±1.9	16.2±4.8
Δ	+19.3 pp (+55%)		+11.4 pp (3.4×)	

the largest relative Hit@5 gains (+93% and +70%), suggesting that domains with complex multi-object interactions benefit most from learning precise action effects.

We set $\lambda = 2$ to emphasize action disambiguation, reflecting that distinguishing among K domain actions applied to the same state requires finer-grained representations than distinguishing among B random states in a batch. Without it, EMBEDPLAN's extrapolation performance drops by roughly one-third and action accuracy is nearly absent. The full ablation across $\lambda \in \{0, 0.5, 1, 1.5, 2, 4\}$ confirms $\lambda = 2$ as the best setting, and performance degrades slightly at $\lambda = 4$ due to over-emphasis on action discrimination at the expense of state prediction.

H PRELIMINARY STUDIES

H.1 LATENT DISTANCE ALIGNMENT

Before learning transition functions, we tested whether pre-trained embeddings already encode planning-relevant structure. If embedding geometry reflects plan costs, one could use simple distance-based heuristics for search without any additional learning.

Hypothesis We define *latent distance alignment* (LDA) as the property that embedding distance between a state and a goal correlates with the number of actions required to reach the goal:

$$\mathrm{corr}\big(d(\mathbf{e}_s, \mathbf{e}_g),\ \mathrm{cost}(s, g)\big) > 0, \tag{30}$$

where $\mathbf{e}_s$ and $\mathbf{e}_g$ are the embeddings of state s and goal g, $d(\cdot, \cdot)$ is cosine distance, and $\mathrm{cost}(s, g)$ is the optimal plan length from s to g.

Motivation This hypothesis draws from successes in other domains. CLIP embeddings align images and text such that semantic similarity corresponds to embedding proximity (Radford et al., 2021). Sentence embeddings place entailed sentences closer than contradictions (Reimers & Gurevych, 2019). We test whether similar alignment emerges for planning cost.

Method For 21,003 state-goal pairs across the nine domains, we computed embedding distances using all four encoders and correlated them with ground-truth plan costs from A* search.

Result After controlling for prompt length as a confound, correlations collapsed to near zero across all encoders. Pre-trained embeddings do not encode planning cost through geometric distance.

Implication This negative result motivated our transition learning approach: rather than relying on inherent geometry, we explicitly learn how actions transform states in embedding space.

I Reproducibility

I.1 Code and data availability

Code is available in an anonymized repository (`https://anonymous.4open.science/r/AnonymizedTextualPlanningLatentTransitions-DC6D/README.md`), and the processed transition datasets will be released upon acceptance. All base encoders (Llama-3.3-70B, Qwen2.5-7B, MPNet, BGE-M3) are publicly available.

I.2 Experimental reproducibility

- **Random seeds**: all experiments run with seeds $\{42, 123, 456\}$. Throughout the paper, $\pm$ denotes the standard error of the mean, over seeds for per-domain entries and over the nine domains for cross-domain means.
- **Hardware**: NVIDIA A100 80 GB GPU.
- **Software**: PyTorch 2.1, CUDA 12.1, Python 3.10.

J Error analysis

We qualitatively analyzed Hit@5 errors under Extrapolation evaluation to characterize failure modes. We sampled 50 incorrect predictions per domain, where the ground-truth next state ranked outside the top 5, and manually inspected the retrieved candidates.

Methodology For each error, we compared the top-1 retrieved state against the ground-truth next state, counting the number of differing predicates.

Findings Across domains, the median predicate difference between the top-1 retrieved state and the ground truth was 2 (IQR: 1–3). Table 29 shows representative examples, with the differing predicates set in italics.

Table 29: Representative Hit@5 errors. Retrieved states differ from the ground truth by one or two predicates (in italics), typically involving object locations or holdings within the same problem instance.

Domain	**Ground truth s'**	**Retrieved $\hat{s}'$**
Blocksworld	Block A is clear, *arm holds B*, *C is on table*	Block A is clear, *arm holds C*, *B is on table*
Ferry	Car c1 at l0, ferry empty, *c2 at l1*	Car c1 at l0, ferry empty, *c2 at l0*
Logistics	*Package p1 in truck t0*, t0 at l1-0	*Package p1 at l1-0*, t0 at l1-0

These errors suggest the model captures coarse transition structure (correct problem context, approximate state region) but struggles to resolve fine-grained predicate changes, particularly when multiple objects undergo similar transformations.

K Complete results tables

Tables 30 and 31 report complete metrics for the two largest encoders.

Table 30: Complete metrics for Qwen2.5-7B across all domains and splits.

Domain	Split	Hit@1	Hit@5	Hit@10
Blocksworld	Interpolation	94.3±0.2	100.0±0.0	100.0±0.0
	Extrapolation	14.2±4.0	41.6±8.8	56.6±8.3
Depot	Interpolation	76.7±1.0	98.2±0.1	99.0±0.0
	Extrapolation	4.9±1.2	24.8±4.9	42.2±7.8
Ferry	Interpolation	98.7±0.1	99.9±0.0	100.0±0.0
	Extrapolation	10.7±0.8	36.7±0.7	52.8±1.1
Floortile	Interpolation	96.2±0.3	99.4±0.0	99.6±0.0
	Extrapolation	24.0±5.7	55.2±11.0	69.0±11.5
Goldminer	Interpolation	94.0±0.6	99.9±0.0	100.0±0.0
	Extrapolation	34.5±6.8	74.4±5.7	86.7±2.2
Grid	Interpolation	80.9±1.2	98.6±0.1	99.6±0.1
	Extrapolation	19.4±3.2	62.7±5.2	76.9±5.7
Logistics	Interpolation	88.9±1.1	99.6±0.1	99.8±0.0
	Extrapolation	14.9±4.6	44.7±11.3	60.7±12.3
Rovers	Interpolation	97.4±0.1	99.7±0.0	99.8±0.0
	Extrapolation	18.5±1.0	49.2±1.8	65.0±1.8
Satellite	Interpolation	98.1±0.1	99.9±0.0	100.0±0.0
	Extrapolation	14.3±0.4	40.0±0.6	54.8±1.0

Table 31: Complete metrics for Llama-3.3-70B across all domains and splits.

Domain	Split	Hit@1	Hit@5	Hit@10
Blocksworld	Interpolation	96.2±0.2	100.0±0.0	100.0±0.0
	Extrapolation	17.6±5.2	49.1±9.8	64.6±7.5
Depot	Interpolation	79.5±0.7	98.8±0.1	99.2±0.1
	Extrapolation	4.7±1.2	25.9±6.4	41.2±8.1
Ferry	Interpolation	99.1±0.1	100.0±0.0	100.0±0.0
	Extrapolation	12.0±1.7	40.6±3.5	58.1±4.0
Floortile	Interpolation	97.4±0.2	99.6±0.0	99.8±0.0
	Extrapolation	37.6±12.7	68.8±15.6	78.9±12.7
Goldminer	Interpolation	95.8±0.3	100.0±0.0	100.0±0.0
	Extrapolation	35.8±6.4	76.2±5.2	88.0±1.7
Grid	Interpolation	84.3±0.8	99.8±0.0	99.9±0.0
	Extrapolation	25.7±1.2	74.9±1.2	88.0±1.2
Logistics	Interpolation	91.2±0.6	99.9±0.0	100.0±0.0
	Extrapolation	16.5±3.5	53.7±9.8	70.8±9.8
Rovers	Interpolation	97.8±0.1	99.0±0.1	99.5±0.1
	Extrapolation	16.9±1.2	54.4±2.9	72.2±2.3
Satellite	Interpolation	98.5±0.1	99.9±0.0	100.0±0.0
	Extrapolation	14.4±2.3	47.5±5.8	66.9±5.8

L Reference methods: specifications and full results

All reference methods use the protocol of Section 3: identical splits, identically budgeted heads, a 128-candidate pool, and best-epoch selection. Three domains spanning the observed range of extrapolation difficulty, three seeds.

L.1 SPECIFICATIONS

Non-learned floors. *Identity* predicts $\hat{s}' = E(s)$. *Offset* adds a mean training displacement keyed by ground action (*grounded*) or action schema (*lifted*), with unseen keys falling back to the global mean. Both operate in the encoder's own space.

Non-LLM encoders. Each replaces E with a sparse featurization of the same text through an identical head: *TF-IDF* (word 1–2 grams, sublinear tf, smooth idf), *bag-of-words* (raw counts), *char* (character 3–5 grams), and *bag-of-literals* (binary bag over the symbolic literal vocabulary). Vocabularies are fitted on training states only. Their widths range from 234 to 2,498 dimensions, and no test state received an all-zero representation.

Lifted STRIPS induction (oracle). For each training transition (s, a, s') with $a = (\textit{schema}\ o_1 \dots o_n)$ we take $\textit{add} = s' \setminus s$, $\textit{del} = s \setminus s'$, and replace each o_i by a positional marker $?p_i$, and application substitutes back. One transition per schema suffices under STRIPS, and every further transition is a consistency check. Disagreements with the first pair seen for a schema number zero on Ferry and Logistics and 52 on Goldminer, whose `fire-laser` and `detonate-bomb` have cell-dependent effects. An unseen schema yields no prediction and is scored as a miss, which is why the method scores zero under Cross-Domain and Leave-One-Out.

Tabular one-hot. Represents each state by a one-hot lookup row instead of its frozen text embedding and is trained on identical data. It isolates what the frozen embedding contributes beyond lookup: every test state that never occurred in training has an untrained row. On Logistics under Interpolation it reaches 8.3% Hit@1 against 96.3% for EMBEDPLAN, and on held-out problems 0.2%, below the 0.8% chance level, since every held-out state maps to an untrained row. The near-total failure of lookup on identical data indicates that the generalization reported in the main text comes from the geometry of the frozen embedding space.

Null controls. *Context-only* deletes the `CURRENT STATE` block, retaining problem and goal text, and therefore carries no state information. *Random* assigns a fixed Gaussian vector per state.

L.2 TIE-BREAKING AND THE NULL CONTROLS

Ranks use worst-case tie-breaking: a candidate scoring exactly equal to the ground truth counts against it. Any representation that collapses same-problem states produces ties among all candidates, and deleting the state block does exactly this. On Ferry, 22,075 distinct states within one held-out problem map to a single vector. Under best-case tie-breaking such a representation would attain 100.0% Hit@5 while carrying no state information, whereas under the worst-case convention it attains 0.0%. The random control, equally uninformative but with distinct vectors, scores near chance under both. The floor of the Extrapolation protocol is therefore 0.0, and the convention is load-bearing.

L.3 FULL RESULTS

Under Interpolation every text representation exceeds 99.8% Hit@5, so the protocol cannot separate them, and Hit@1 separates them only slightly (95.2–97.4). The main-text comparison is therefore reported under Extrapolation.

L.4 THE EXTRAPOLATION GAP IS PARTLY A GROUNDING EFFECT

A held-out problem introduces new objects, so many test transitions invoke a ground action absent from training (57.6% on Ferry, 65.5% on Logistics, 11.7% on Goldminer), yet *no* test action schema is unseen. Keying the offset reference on schemas rather than ground actions gains 5.7 points on Ferry and 3.3 on Logistics, while costing 6.6 on Goldminer where most groundings are already observed. The gap reflects a representation keyed on object identity rather than an inability to acquire domain structure.

Table 32: Extrapolation, Hit@5 (%), mean ± SE over 3 seeds. The Llama-3.3-70B row is reproduced from Table 13, and all other rows were computed for this comparison under the identical protocol.

Method	**Ferry**	**Logistics**	**Goldminer**	**Mean**
Lifted STRIPS induction (oracle)	100.0 ± 0.0	100.0 ± 0.0	99.3 ± 0.3	99.8
Char 3–5 grams	71.1 ± 1.7	69.8 ± 7.4	97.3 ± 0.7	79.4
Bag-of-words	70.8 ± 2.7	67.3 ± 4.8	99.3 ± 0.2	79.1
Bag-of-literals	51.9 ± 5.7	81.4 ± 0.7	95.7 ± 2.0	76.3
TF-IDF bigrams	59.5 ± 2.1	50.0 ± 4.3	97.2 ± 1.2	68.9
EMBEDPLAN (Llama-3.3-70B)	40.6 ± 3.5	53.7 ± 9.8	76.2 ± 5.2	56.8
EMBEDPLAN (BGE-M3)	42.8 ± 2.1	51.7 ± 3.9	70.2 ± 5.2	54.9
Offset, lifted	34.3 ± 0.8	56.4 ± 5.4	50.7 ± 1.7	47.2
Offset, grounded	28.6 ± 2.1	53.1 ± 2.3	57.3 ± 4.1	46.3
Identity	18.5 ± 1.0	49.8 ± 3.4	41.2 ± 1.6	36.5
Random vectors [null]	4.4 ± 0.5	4.8 ± 0.6	5.2 ± 0.5	4.8
Context-only [null]	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0

Table 33: Extrapolation, Hit@1 (%), mean ± SE over 3 seeds. The separation between EMBEDPLAN and the non-learned floors is wider here than at Hit@5.

Method	**Ferry**	**Logistics**	**Goldminer**	**Mean**
Lifted STRIPS induction (oracle)	100.0 ± 0.0	100.0 ± 0.0	99.3 ± 0.3	99.8
Bag-of-words	31.5 ± 3.3	25.4 ± 4.2	78.0 ± 1.9	45.0
Char 3–5 grams	31.0 ± 2.4	24.2 ± 2.4	70.6 ± 3.5	41.9
TF-IDF bigrams	24.7 ± 1.5	14.2 ± 2.9	72.8 ± 2.5	37.2
Bag-of-literals	14.2 ± 0.9	30.7 ± 0.8	66.5 ± 8.7	37.1
EMBEDPLAN (Llama-3.3-70B)	12.0 ± 1.7	16.5 ± 3.5	35.8 ± 6.4	21.4
EMBEDPLAN (BGE-M3)	12.9 ± 0.6	15.5 ± 1.7	31.3 ± 5.3	19.9
Offset, grounded	6.9 ± 2.1	5.0 ± 0.4	9.7 ± 2.5	7.2
Offset, lifted	5.9 ± 0.9	4.9 ± 0.9	5.8 ± 2.3	5.5
Identity	2.2 ± 0.1	4.4 ± 0.8	3.4 ± 0.1	3.3
Random vectors [null]	0.9 ± 0.1	0.6 ± 0.2	0.6 ± 0.1	0.7
Context-only [null]	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0

L.5 SEED VARIANCE

The Extrapolation split shuffles the problem list and holds out the last 20%. With 5–10 problems per domain, distinct seeds need not yield distinct partitions: seeds 42 and 456 select identical held-out problems in all three domains, and every non-learned reference differs by exactly 0.00 between them. The seed 42–456 difference therefore isolates training stochasticity (1.5 points for EMBEDPLAN), while the difference against seed 123 reflects the choice of held-out problems (11.0 points). Split choice dominates training noise by 4–$7\times$, so a three-seed spread over two distinct partitions understates the true variance.

Table 34: Interpolation, Hit@5 and Hit@1 (%), mean ± SE over 3 seeds. Six methods lie within 0.2 points of one another on Hit@5.

Method	Ferry	Logistics	Goldminer	Mean
Hit@5				
Bag-of-literals	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.1	100.0
Bag-of-words	99.9 ± 0.1	100.0 ± 0.0	100.0 ± 0.0	100.0
EMBEDPLAN (Llama-3.3-70B)	100.0 ± 0.0	99.9 ± 0.0	100.0 ± 0.0	100.0
TF-IDF bigrams	99.9 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0
Char 3–5 grams	99.9 ± 0.0	99.9 ± 0.1	100.0 ± 0.0	100.0
EMBEDPLAN (BGE-M3)	100.0 ± 0.0	99.8 ± 0.1	99.9 ± 0.0	99.9
Lifted STRIPS induction (oracle)	100.0 ± 0.0	100.0 ± 0.0	99.5 ± 0.1	99.8
Offset, grounded	90.1 ± 0.0	99.1 ± 0.1	96.1 ± 0.3	95.1
Offset, lifted	79.1 ± 0.2	98.3 ± 0.1	89.6 ± 0.3	89.0
Identity	49.6 ± 0.2	98.2 ± 0.2	73.2 ± 0.2	73.7
Random vectors [null]	8.2 ± 0.3	6.2 ± 0.2	7.8 ± 0.5	7.4
Context-only [null]	5.7 ± 0.3	6.2 ± 0.5	4.3 ± 0.1	5.4
Hit@1				
Lifted STRIPS induction (oracle)	100.0 ± 0.0	100.0 ± 0.0	99.5 ± 0.1	99.8
Bag-of-literals	98.7 ± 0.1	97.9 ± 0.2	95.7 ± 0.1	97.4
Char 3–5 grams	95.4 ± 0.2	97.1 ± 0.0	97.6 ± 0.1	96.7
Bag-of-words	95.5 ± 0.1	96.4 ± 1.0	97.8 ± 0.2	96.6
TF-IDF bigrams	94.3 ± 0.2	96.5 ± 0.5	98.0 ± 0.1	96.2
EMBEDPLAN (Llama-3.3-70B)	99.1 ± 0.1	91.2 ± 0.6	95.8 ± 0.3	95.4
EMBEDPLAN (BGE-M3)	97.8 ± 0.2	93.6 ± 0.9	94.3 ± 0.8	95.2
Offset, grounded	72.6 ± 0.1	86.1 ± 0.4	69.5 ± 0.6	76.1
Offset, lifted	51.7 ± 0.6	78.8 ± 0.1	54.6 ± 0.4	61.7
Identity	24.6 ± 0.2	76.6 ± 0.1	39.5 ± 0.3	46.9
Random vectors [null]	1.9 ± 0.1	1.2 ± 0.1	1.4 ± 0.1	1.5
Context-only [null]	1.0 ± 0.1	0.0 ± 0.0	0.1 ± 0.1	0.4

M PRELIMINARY STUDY: UNFREEZING THE ENCODER

To bound what the frozen constraint costs, we adapt BGE-M3 with LoRA (rank 16, α=32, on `query`, `key`, `value`, `dense`) jointly with the heads and transition network, for two epochs at batch size 16 with encoder learning rate 10^{-5}. Both arms then re-encode the pool with their own encoder and fit a fresh, identically budgeted head, so only the embedding space differs. A **cold start** unfreezes the encoder immediately, whereas a **warm start** first trains the head to convergence in the unadapted space. Since LoRA is initialized to a zero delta, the adapted encoder is bit-identical to the frozen one at that point and the warmed head transfers exactly.

Table 35: Full-pool Hit@1 (%), mean over three domains and three seeds. The full pool contains every state in the domain (12,237–46,205) rather than 128 candidates.

Encoder treatment	Extrapolation	Interpolation
Frozen	3.8	25.6
LoRA, cold start	6.2	39.6
LoRA, warm start	**9.3**	**41.7**

Table 36: Effect of the warm start, in points of full-pool Hit@1 relative to the cold start. The sign is consistent per domain across two independent splits.

Domain	Extrapolation	Interpolation
Goldminer	+13.3	+10.5
Ferry	−0.1	+1.6
Logistics	−3.9	−5.9

Adaptation is worth $1.6\times$ (Interpolation) to $2.4\times$ (Extrapolation) over the frozen encoder, and the schedule matters in a domain-dependent but reproducible way. On Goldminer, cold-start joint training is *worse than not fine-tuning at all*: the adapter is optimized against a transition function whose loss has not converged (≈ 2.05 during the joint stage, against 0.59 for the same head once the space stops moving) and which is then discarded. Warming the head removes this. Across all nine domain–seed cells the warm-started model is never worse than frozen, whereas the cold-started model is worse in two. We report per-domain effects rather than a pooled test, since the effect changes sign across domains.

Encoder capacity helps as well: scaling the frozen encoder from 568M to 70B parameters raises Extrapolation Hit@5 from 36.3% to 54.6% averaged over the nine domains (Table 6). On our templated states the sparse lexical references of Appendix L remain ahead (79.4% against 56.8% for the largest encoder on the three domains compared there), because an action edits only a few facts of an otherwise identical text, which sparse features encode directly while a pooled embedding of the whole prompt barely moves (Section 4.3). The gap therefore reflects the form of the representation rather than its size.

N SENSITIVITY TO FACT ORDERING

State descriptions render a *set* of literals as a comma-separated clause list, so the ordering is arbitrary and semantically null. We shuffle the clauses within the `CURRENT STATE` block, leaving problem and goal text untouched, and compare the resulting displacement against that of a real transition, over 256 states per domain and four reorderings each.

Table 37: Embedding displacement from a semantically null reordering ($d_{\text{order}} = 1 - \cos(s, \sigma(s))$, with σ a random clause order) against that from a real transition ($d_{\text{action}} = 1 - \cos(s, s')$), BGE-M3.

Domain	$\cos(s, s')$	$\cos(s, \sigma(s))$	$d_{\text{order}}/d_{\text{action}}$
Ferry	0.99672	0.99654	1.06
Goldminer	0.99881	0.99862	1.16
Logistics	0.99928	0.99915	1.18
Mean			**1.13**

Reordering displaces the embedding slightly *further* than applying an action does, in every domain. This is consistent with the ordering in Appendix L: the representations performing best under Extrapolation are bags of words, character n-grams, or literals and are largely insensitive to clause order. Order augmentation during encoding is a concrete direction for future work.