COST-TO-GO ESTIMATION BY DECOMPOSED LATENT PATH INTEGRATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Estimating the effort required to reach a goal from a given state is a fundamental problem in planning and control. Many recent approaches estimate this cost by encoding the start and goal observations independently and computing a metric or quasimetric between their representations. This paper introduces *decomposed latent path integration* (DeLPI), an architectural primitive that estimates distance through accumulated motion in latent space. In our instantiation, the two observations are jointly encoded into a spatial grid of latent vectors, which is iteratively updated by a depth-recurrent module. The predicted distance is computed as the sum of the latent vector displacements produced over these updates. We compare our method against metric, quasimetric, and joint-encoding baselines on four environments with nontrivial combinatorial structure: *Switchyard*, a gridworld with a controllable degree of factor coupling; *Sokoban*; *Rubik's cube*; and *Craftax*. Across these environments, it either outperforms the baselines or achieves comparable performance with substantially fewer parameters. More broadly, our experiments show that, in these environments, architectures that jointly process the start and goal states consistently outperform approaches that embed the two states independently.

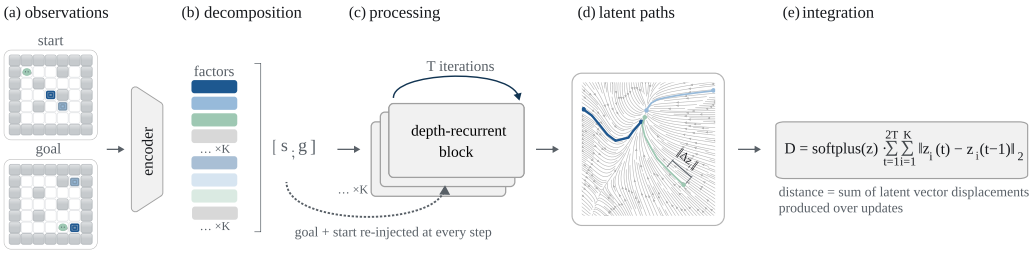


Figure 1: **Overview of the proposed method. (a) observations:** a factorized gridworld where planning distance depends on coupled factors. **(b) decomposition:** the pair (s, g) of a start and a goal state is encoded into an indexed collection of latent vectors. **(c) processing:** a depth-recurrent network updates latent vectors over T iterations (2 blocks per iteration) with start and goal re-injection. **(d) latent paths:** each latent vector traverses a path in latent space, one segment per block update. **(e) integration:** the predicted distance $\hat{d}(s, g)$ is the total length of those paths, summed over cells and iterations.

1 INTRODUCTION

Heuristic search and goal-conditioned policies both often rely on an estimate of how far a state is from a goal. In planning, this quantity is often called the *optimal cost-to-go*, $d(s, g)$, and it is the minimum cost of reaching goal g from state s. Learned approximations of d serve as heuristics that guide search (Groshev et al., 2018; Chrestien et al., 2021). In *goal-conditioned reinforcement learning* (GCRL), the analogous quantity is the *optimal goal-conditioned value*, $V^*(s, g)$ (Schaul et al., 2015; Andrychowicz et al., 2017), and under unit step costs, $V^*(s, g) = -d(s, g)$. Both fields thus face the same estimation problem, and we refer to its target as the planning distance.

Recent work in GCRL learns an embedding $\phi(s)$ and a distance function D such that $d(s, g) \approx D(\phi(s), \phi(g))$. Because some state transitions are irreversible, D is often a quasimetric (Wang et al., 2023; Wang & Isola, 2022; Liu et al., 2023). The appeal of this approach is that the embedding

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.

captures the transition structure of the environment during training, so that a distance query reduces to evaluating a geometric relation between two static embeddings.

This static readout is well suited to environments that decompose into independent factors, since the distance between two states is the sum of the distances between their factors. The embedding function can therefore embed each factor separately. Even though many natural environments can be decomposed into factors, these factors often interact. An unsatisfied precondition can change the shortest path abruptly and ignoring the interactions can lead to poor estimates.

Motivated by this observation, we propose *DEcomposed Latent Path Integration* (DeLPI), a primitive for planning distance estimation under factorized, coupled dynamics, and instantiate it in a model which we also call DeLPI (Decomposed Latent Path Integrator). In contrast to existing approaches, which decode a scalar distance from static representations, DeLPI encodes the start and goal observations jointly into a grid of latent vectors. A depth-recurrent convolutional network, with global self-attention over the cells, then updates these vectors over T iterations (2 blocks/updates per iteration). The predicted planning distance is computed directly as the total length of the paths the cells trace in latent space, $\hat{d}(s, g) \propto \sum_t \sum_i \|z_i^{(t)} - z_i^{(t-1)}\|_2$, where $z_i^{(t)}$ denotes the latent vector of cell i after update t.

To systematically test this primitive, we introduce *Switchyard*, a small Sokoban-like environment with several coupled mechanisms allowing us to control the degree of coupling between the factors. To ensure proper evaluation, we also evaluate the model on the *Sokoban*, *Rubik's cube* and *Craftax* environments which have nontrivial combinatorial structure and have been used in previous work in planning or reinforcement learning. In this work, we mostly focus on the supervised setting, where models are trained on a static dataset of start-goal pairs and the labels are obtained from optimal or non-optimal trajectories. In Section 4, we also evaluate a setting where models are trained online inside a search algorithm. We compare DeLPI with learned heuristics from the planning literature and with embedding-based metric and quasimetric models from goal-conditioned reinforcement learning, and show that it outperforms all baselines on Switchyard, Sokoban and Craftax, and that on the Rubik's cube it matches the strongest baseline with a fraction of its parameters.

The introduced primitive is general and can be easily instantiated in other models. At the same time it is distinct from other architectural ideas and can therefore be a viable candidate for increasingly common agentic architecture search (Liu et al., 2025; Jeong et al., 2026).

To summarize, this work makes the following contributions:

- We propose *decomposed latent path integration*, an architectural primitive that predicts the planning distance as the accumulated displacement of a decomposed latent representation over recurrent updates, instead of decoding a scalar from a static representation, and instantiate it in a model called DeLPI.
- We introduce *Switchyard*, a gridworld with controllable degree of coupling between involved mechanisms.
- We show that DeLPI outperforms other representative models used for distance estimation on Switchyard, Sokoban and Craftax, and that on the Rubik's cube it matches the strongest baseline with 3% of its parameters. When the estimates are used with a greedy policy, DeLPI solves almost twice as many boards on Sokoban at a quarter of the search cost.
- We show that architectures that process start and goal states jointly in most cases outperform embedding models that encode them separately.
- We show that the latent displacement is interpretable: cells on a shortest path accumulate about twice the motion of the rest, and the motion decomposes along the elements that can be acted on.

2 Decomposed Latent Path Integrator

DeLPI computes the distance $\hat{d}(s, g)$ directly from the pair of observations (Figure 1): a convolutional encoder maps the start and the goal jointly to one latent vector per grid cell, a depth-recurrent convolutional network updates these vectors with the start and the goal re-injected at every block, and the predicted distance is the total length of the paths the vectors trace in latent space.

Decomposition. An encoder maps the pair (s, g) jointly to an indexed collection of K latent vectors $z_1^{(0)}, \ldots, z_K^{(0)}$, and all later computation operates on this collection. DeLPI does not depend on how this array is produced. In our experiments, each state is a grid with N binary channels, one per cell type, and the start and goal are stacked into $2N$ channels. Two 3×3 convolutions then map the stacked pair to one latent vector per grid cell (Figure 1b), so the collection has the shape of the grid and each vector summarizes a small neighbourhood of its cell in both states. Appendix D replaces this encoder with learned slot queries that attend over the cells, so that each vector comes to represent an entity of the environment instead of a concrete position.

Recurrent processing. A depth-recurrent convolutional network then refines the latent vectors over T iterations, reusing the same weights at every iteration (Figure 1c). In the presented model, each iteration consists of two residual blocks with distinct weights. Each block normalizes the latent grid, concatenates the $2N$ channels of the input pair, and applies two 3×3 convolutions, to which it adds a global self-attention term over all cells so that distant cells can interact within a single iteration. The resulting update is added to the latent grid with a fixed step size.[1] Because the start and goal are re-injected at every block, the latent grid stays conditioned on the input pair throughout the iterations. The variant in Appendix D replaces this recurrent convolutional block with a weight-tied transformer over the slot vectors, and Table 5 compares the two.

Path integration. Over the $2T$ updates, each latent vector z_i traces a path $z_i^{(0)}, \ldots, z_i^{(2T)}$ in latent space, one segment per residual block (Figure 1d). The predicted distance is the total length of these K paths, $\hat{d}(s, g) = \alpha \sum_{t=1}^{2T} \sum_{i=1}^{K} \|z_i^{(t)} - z_i^{(t-1)}\|_2$, scaled by a learned calibration factor $\alpha > 0$ (Figure 1e). The path length is itself the prediction, so no decoder is needed. Because the sum decomposes over cells, each region of the grid contributes its own term to the distance. The *DeLPI-I* ablation in Section 5 replaces this readout with a linear head on the final latent vectors. On Switchyard, accuracy improves as iterations are added and falls without the integration, so in this goal-conditioned setting, recurrence and integration complement each other (Figure 6).

3 Experimental Settings

This section describes the evaluation of DeLPI, covering the four benchmarks, the baselines, the training protocol and the metrics. Across all four benchmarks an example is a pair of a start and goal state, and the label is the number of steps on a path between them, in most cases the shortest one, computed by breadth-first search over the state graph. Further details are presented in Appendix B.

3.1 Benchmarks

Switchyard is a factorized gridworld built to control how strongly the factors interact. Reaching the goal requires several mechanisms to be operated in the right order, and Appendix A describes the elements and the state representation, and Figure 7 shows one instance. Instances are generated at six grid sizes, from 9×9 to 19×19, with the same number of mechanisms at every size, so the input grows while the coupling stays constant. A second ladder, in which the number of mechanisms grows with the grid, is reported in Appendix C (Figure 9). For *Sokoban* we adopt the setup of CRTR (Ziarko et al., 2025), 12×12 boards with four boxes generated as in Czechowski et al. (2021). On this benchmark we also report a setup in which the labels come from the non-optimal solution trajectories of CRTR, which overshoot the shortest path by 6.4 steps on average. For *Craftax* (Matthews et al., 2024) we use the Classic variant on 16×16 worlds, made fully observable and deterministic. The *Rubik's cube* is the only benchmark that is not a gridworld. Because its moves form a group, any pair of states can be rewritten so that the goal is the solved cube, which is what cube-solving networks such as DeepCubeA (Agostinelli et al., 2019) exploit to take a single state as input. We keep the pair unnormalized, so the task stays a distance between two arbitrary states, and every pair is labelled with its exact shortest-path distance, which is feasible up to distance nine (for scale, any cube can be solved in at most 26 quarter turns).

[1] The normalization is GroupNorm with a single group, and the nonlinearity after each encoder layer and after the first convolution of a block is SiLU. Self-attention is computed from the activation after that first convolution, and the step size is $0.1/\sqrt{W}$ for width W.

3.2 BASELINES

The baselines cover the ways the distance is computed from the two states, and all of them are trained on the same labels as DeLPI. From goal-conditioned reinforcement learning we take the *embedding models*, which encode each state separately with a shared convolutional encoder and differ in the head that compares the two embeddings, from the simplest to the most structured: *Sym* takes the L_1 distance, which is symmetric, *MRN* (Liu et al., 2023) adds an asymmetric residual to a symmetric metric, and *IQE* (Wang & Isola, 2022) builds a quasimetric from unions of intervals over the latent components. From the learned heuristics for search we take *CoAt* (Chrestien et al., 2021), a convolution-attention extension of the joint-encoding network of Groshev et al. (2018), in our own re-implementation. Finally, CRTR (Ziarko et al., 2025) targets problems with nontrivial combinatorial structure, Sokoban and the Rubik's cube among them, and reports a supervised baseline that encodes the pair of states jointly. The *s-CRTR* $_{class}$, the *s-CRTR* $_{reg}$ and the *s-CRTR* $_{MLP}$ build on its networks, the first predicting a distribution over distance bins and the other two a single scalar. Appendix B.2 gives the architecture and configuration of every baseline, together with the few details in which our use deviates from the original works. Sym, MRN and IQE were not trained on our benchmarks by their authors, so we ran an encoder sweep for each head on Switchyard. The encoder which worked the best for all three was a variant of the CRTR encoder. We report CoAt on all four benchmarks, Sym, IQE, MRN and the two convolutional s-CRTR networks on the three gridworlds, and the s-CRTR $_{\text{MLP}}$ on the Rubik's cube, where CoAt and the s-CRTR $_{\text{MLP}}$ are trained at three parameter budgets down to the size of DeLPI, so that the models are also comparable at matched budgets.

3.3 TRAINING PROTOCOL

All models minimize a smooth L_1 loss on the label, except the s-CRTR $_{\text{class}}$, which minimizes cross-entropy over distance bins. All models are trained with the Adam optimizer using four random seeds unless stated otherwise. We report the checkpoint with the lowest validation MAE on Sokoban, the checkpoint with the highest validation correlation on the Rubik's cube and Craftax, and the final checkpoint on Switchyard. Appendix B.3 lists the training hyperparameters.

Ranking objective. Chrestien et al. (2023) argue that a heuristic for a search algorithm should instead be trained to rank the states of the open list rather than to regress the distance, and we therefore also train DeLPI and CoAt under their L* ranking loss on the Sokoban boards released by the authors (to have a matching evaluation setup). We evaluate the models in two regimes: under *static* L*, we form the state pairs from an optimal solution path and the states that the search algorithm discovers off the path. Under *in-loop* L*, the current model guides a budgeted search algorithm that collects the training states for the next update of the model. Appendix B.4 provides further details about the loss and the two training regimes.

3.4 METRICS

We report two direct metrics on held-out test examples: the Pearson correlation between predicted and true distances and the mean absolute error (MAE) in steps. We also report two planning metrics: the success rate of a greedy policy, which always steps to the successor with the smallest predicted distance, and the success rate of a best-first search. The budgets and the numbers of test instances differ between benchmarks, because we reuse the settings of the work each benchmark comes from (Appendix B.5). On the Rubik's cube, neither planning metric is comparable with published cube-solving rates, since our pairs stop at distance nine while a random cube is 16 to 22 turns from its target. Under the ranking objective, we report the number of test boards solved by A* within 1,000 and 10,000 expansions, and the mean number of expansions per board.

4 RESULTS

We report the direct metrics and the planning metrics on each of the four benchmarks, followed by the results under the ranking objective. DeLPI leads on Switchyard, Sokoban and Craftax, and on the Rubik's cube it matches the strongest baseline with a fraction of its parameters.

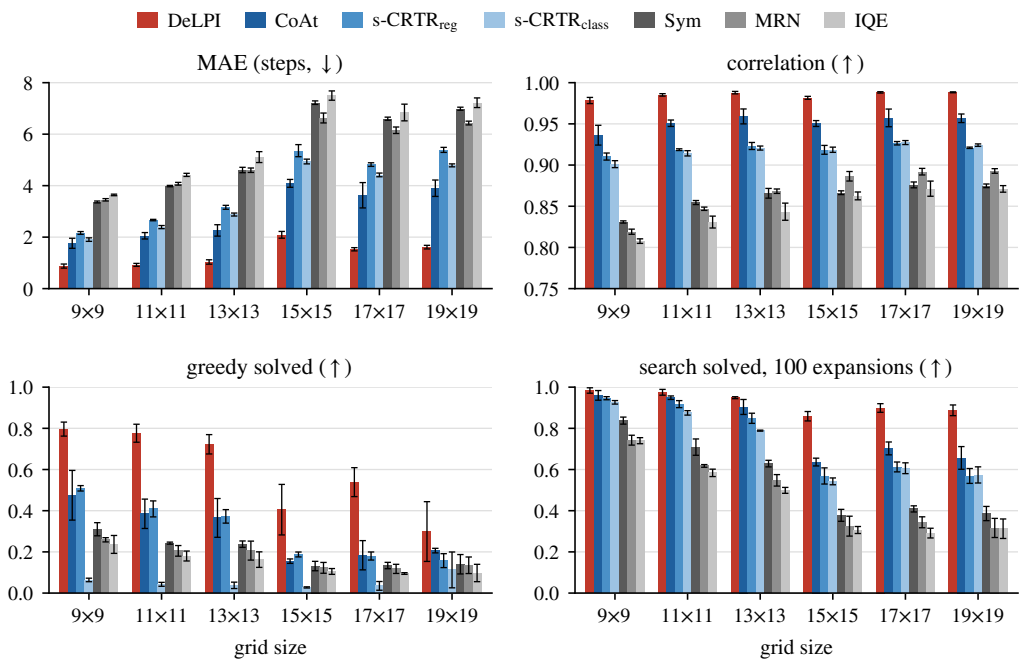


Figure 2: **Results on *Switchyard* at six grid sizes.** Accuracy over all test pairs of held-out instances. Greedy and search success on 200 test pairs per grid size. The correlation axis starts at 0.75.

Switchyard. Figure 2 reports the results on Switchyard for grids from 9×9 to 19×19. The ranking is the same at every grid size: DeLPI, then CoAt, then the two s-CRTR networks on the stacked pair, then the three embedding models. DeLPI has about half the error of CoAt at every size, and its worst seed has a higher correlation than the best CoAt seed. The errors of all models grow with the grid, since the distances grow with it, and the 15×15 grid, which has the longest distances relative to its size, is the hardest for every model. The greedy policy solves roughly twice as many pairs with DeLPI as with CoAt at every grid size, and in search the difference is widest on the three largest grids. Section 5 provides a qualitative analysis of the predicted distances over the environment together with interpretability results and Appendix C reports the second difficulty ladder, in which the number of mechanisms grows with the grid.

Sokoban. Table 1 reports all models on Sokoban. DeLPI has the lowest error and the highest correlation on the test set, it is the only model that solves more than half of the boards with the greedy policy, and it solves the most boards with search while generating the fewest nodes per board. Its 48% lower error than CoAt results in almost twice the greedy success rate and a quarter of the search cost.

The lower block of Table 1 reports the same models trained on the non-optimal labels mentioned in Section 3. Direct metrics are still measured against the shortest-path distance, so every model has a higher error and a lower correlation than with shortest-path labels. DeLPI stays first on every planning metric, with twice the greedy success of the best baseline and the fewest nodes in search.

Rubik's cube. Table 2 shows that DeLPI matches the strongest baseline with 3% of its parameters. At 57k parameters its correlation and MAE differ from those of CoAt at 1.83M by less than one standard deviation across seeds, and its greedy and search success rate is the highest. The baselines degrade as they shrink, and at the parameter budget of DeLPI both CoAt and the s-CRTR$_{\text{MLP}}$ fall behind by a clear margin.

The standard approach to the Rubik's cube exploits the group structure to reduce a pair to a single cube, which we deliberately do not do. The pair task is considerably harder: with the reduced input, DeLPI and three cube-solving networks all reached correlations between 0.975 and 0.982 in preliminary runs, whereas on the pair task the correlations in Table 2 range from 0.888 to 0.933.

Table 1: Results on *Sokoban* with labels from shortest paths (upper block) and from non-optimal paths (lower block). Accuracy on 20k test states against the shortest-path distance, planning on the 1,000 evaluation boards.

	model	MAE (↓)	corr (↑)	greedy (↑)	search (↑)	nodes (↓)
Labels from shortest paths						
	DeLPI *(ours)*	$\mathbf{0.85}_{\pm 0.03}$	$\mathbf{0.993}_{< 0.001}$	$\mathbf{0.83}_{\pm 0.02}$	$\mathbf{0.98}_{< 0.01}$	$\mathbf{262}_{\pm 9}$
joint-encoding	CoAt	$1.64_{\pm 0.06}$	$0.981_{\pm 0.001}$	$0.44_{\pm 0.02}$	$0.87_{\pm 0.02}$	$1127_{\pm 119}$
	s-CRTR $_{\text{reg}}$	$1.74_{\pm 0.01}$	$0.979_{\pm 0.001}$	$0.44_{\pm 0.02}$	$0.88_{\pm 0.01}$	$1067_{\pm 66}$
	s-CRTR $_{\text{class}}$	$1.76_{\pm 0.02}$	$0.975_{< 0.001}$	$0.25_{\pm 0.01}$	$0.88_{\pm 0.01}$	$1061_{\pm 39}$
embedding	Sym	$1.99_{\pm 0.05}$	$0.975_{\pm 0.001}$	$0.43_{\pm 0.01}$	$0.86_{\pm 0.01}$	$1170_{\pm 63}$
	MRN	$3.66_{\pm 0.58}$	$0.948_{\pm 0.008}$	$0.22_{\pm 0.08}$	$0.84_{\pm 0.02}$	$1383_{\pm 178}$
	IQE	$5.48_{\pm 0.52}$	$0.938_{\pm 0.007}$	$0.21_{\pm 0.02}$	$0.78_{\pm 0.02}$	$1788_{\pm 130}$
Labels from non-optimal paths						
	DeLPI *(ours)*	$4.83_{\pm 0.28}$	$\mathbf{0.949}_{\pm 0.001}$	$\mathbf{0.34}_{< 0.01}$	$\mathbf{0.84}_{< 0.01}$	$\mathbf{1308}_{\pm 9}$
joint-encoding	CoAt	$4.73_{\pm 0.18}$	$0.929_{\pm 0.005}$	$0.18_{\pm 0.01}$	$0.59_{< 0.01}$	$2875_{\pm 13}$
	s-CRTR $_{\text{reg}}$	$4.15_{\pm 0.15}$	$0.935_{\pm 0.010}$	$0.17_{\pm 0.02}$	$0.69_{\pm 0.02}$	$2305_{\pm 141}$
	s-CRTR $_{\text{class}}$	$\mathbf{3.54}_{\pm 0.13}$	$0.939_{\pm 0.005}$	$0.17_{< 0.01}$	$0.72_{\pm 0.02}$	$2122_{\pm 136}$
embedding	Sym	$6.00_{\pm 0.20}$	$0.926_{\pm 0.010}$	$0.09_{\pm 0.01}$	$0.65_{< 0.01}$	$2531_{\pm 10}$
	MRN	$9.15_{\pm 0.38}$	$0.920_{\pm 0.006}$	$0.13_{\pm 0.01}$	$0.67_{\pm 0.06}$	$2424_{\pm 364}$
	IQE	$11.54_{\pm 2.10}$	$0.932_{\pm 0.003}$	$0.14_{\pm 0.01}$	$0.70_{\pm 0.03}$	$2263_{\pm 219}$

Table 2: Results on the *Rubik's cube*: direct metrics on 45,674 test pairs, and the success rates of the greedy policy and of A* search within 1,000 evaluations on 180 test pairs, mean ± standard deviation over four training seeds. The 0.06M rows of CoAt and the s-CRTR $_{\text{MLP}}$ are the best of six configurations with 50k to 61k parameters, chosen on the validation split.

	model	params (M)	MAE (↓)	corr (↑)	greedy (↑)	search (↑)
	DeLPI *(ours)*	0.06	$\mathbf{0.65}_{\pm 0.01}$	$\mathbf{0.933}_{\pm 0.001}$	$\mathbf{0.588}_{\pm 0.027}$	$\mathbf{0.857}_{\pm 0.011}$
joint-encoding	CoAt	1.83	$\mathbf{0.64}_{\pm 0.02}$	$\mathbf{0.933}_{\pm 0.003}$	$\mathbf{0.576}_{\pm 0.020}$	$\mathbf{0.849}_{\pm 0.014}$
		0.52	$0.77_{\pm 0.01}$	$0.911_{\pm 0.002}$	$0.518_{\pm 0.011}$	$0.793_{\pm 0.012}$
		0.06	$0.82_{\pm 0.01}$	$0.903_{\pm 0.003}$	$0.482_{\pm 0.018}$	$0.768_{\pm 0.035}$
	s-CRTR $_{\text{MLP}}$	8.53	$0.81_{\pm 0.02}$	$0.902_{\pm 0.004}$	$0.481_{\pm 0.011}$	$0.725_{\pm 0.022}$
		0.51	$0.81_{\pm 0.01}$	$0.904_{\pm 0.001}$	$0.487_{\pm 0.011}$	$0.739_{\pm 0.013}$
		0.06	$0.91_{\pm 0.01}$	$0.888_{\pm 0.002}$	$0.463_{\pm 0.026}$	$0.682_{\pm 0.017}$

Craftax. On Craftax (Table 3) DeLPI outperforms CoAt in the direct metrics with a fraction of its parameters and in greedy success it is ahead on average, but the margin is not significant. The greedy rates are low in absolute terms because a pair counts as solved only if the policy reaches the target within the true distance, so a single detour results in a failed trajectory. With search DeLPI solves more pairs than CoAt, and no DeLPI seed falls below the best CoAt seed.

Ranking objective. Table 4 reports the results under the L* objective. Under static L*, DeLPI performs on par with CoAt, solving slightly more boards within both budgets with fewer expansions. Under in-loop L*, DeLPI solves 185 of the 200 boards within 1,000 expansions and all but one within 10,000, and its worst seed solves 181. CoAt collapses on two of its three seeds, each of

Table 3: Results on *Craftax*: direct metrics on 156k test pairs, greedy success on 400 test pairs and search success on 200 test pairs; mean ± standard deviation over four training seeds.

	model	params (M)	MAE (↓)	corr (↑)	greedy (↑)	search (↑)
	DeLPI *(ours)*	0.12	$\mathbf{0.23}_{\pm 0.01}$	$\mathbf{0.993}_{\pm 0.001}$	$\mathbf{0.52}_{\pm 0.05}$	$\mathbf{0.94}_{\pm 0.03}$
joint-encoding	CoAt	0.87	$0.37_{\pm 0.02}$	$0.983_{\pm 0.002}$	$\mathbf{0.47}_{\pm 0.04}$	$0.84_{\pm 0.06}$
	s-CRTR $_{\text{reg}}$	0.34	$0.40_{\pm 0.01}$	$0.982_{\pm 0.001}$	$\mathbf{0.49}_{\pm 0.05}$	$0.85_{\pm 0.03}$
	s-CRTR $_{\text{class}}$	0.34	$0.31_{\pm 0.01}$	$0.978_{\pm 0.001}$	$0.33_{\pm 0.03}$	$0.73_{\pm 0.03}$
embedding	Sym	1.29	$1.27_{\pm 0.06}$	$0.837_{\pm 0.005}$	$0.04_{< 0.01}$	$0.53_{\pm 0.03}$
	MRN	1.29	$1.24_{\pm 0.05}$	$0.838_{\pm 0.007}$	$0.04_{\pm 0.01}$	$0.38_{\pm 0.04}$
	IQE	1.29	$1.27_{\pm 0.08}$	$0.828_{\pm 0.015}$	$0.04_{\pm 0.01}$	$0.45_{\pm 0.04}$

Table 4: Results under the L* ranking objective on the 10×10 Sokoban of Chrestien et al. (2023): number of the 200 test boards solved by A* within 1,000 and 10,000 expansions, and mean expansions per board, mean $\pm$ standard deviation over three training seeds. Under in-loop L*, two of the three CoAt seeds collapse.

		solved (↑)		expansions (↓)	
	model	1,000	10,000	1,000	10,000
*static L**	**DeLPI** *(ours)*	$162_{\pm 3}$	$194_{\pm 1}$	$360_{\pm 7}$	$914_{\pm 64}$
	CoAt	$155_{\pm 13}$	$193_{\pm 3}$	$394_{\pm 71}$	$1053_{\pm 297}$
*in-loop L**	**DeLPI** *(ours)*	$\mathbf{185}_{\pm 4}$	$\mathbf{199}_{\pm 0}$	$\mathbf{201}_{\pm 22}$	$\mathbf{360}_{\pm 75}$
	CoAt	$73_{\pm 89}$	$121_{\pm 67}$	$724_{\pm 375}$	$5184_{\pm 4001}$

which solves only 22 boards within 1,000 expansions, and its remaining seed falls behind every DeLPI seed. DeLPI therefore does not depend on the regression objective and can be trained to rank. It performs on par with CoAt when the training states are given and is more robust when it has to be trained in a self-labeling loop.

Embedding versus joint-encoding models. The embedding models, which embed the two states separately and compute the distance between the embeddings, perform substantially worse than the joint-encoding models, which process both states together. We report them on the gridworlds only. On Sokoban, the best embedding model, Sym, has more than twice the error of DeLPI (1.99 vs. 0.85 steps) and solves 0.43 of the boards with the greedy policy, against 0.44 for CoAt and the s-CRTR$_{\text{reg}}$ and 0.83 for DeLPI (Table 1). On Switchyard, Sym, IQE and MRN stay below 0.90 in correlation at every grid size, against 0.98 or more for DeLPI, and in direct metrics and search they fall behind the s-CRTR$_{\text{class}}$, which shares their encoder but is applied to the stacked pair (Figure 2). These results indicate that the main limitation of the embedding models stems from processing the two states separately, rather than from any deficiency in the distance head itself.

5 Interpreting the Predicted Distances

In this section, we provide a qualitative analysis of the predictions of the models on the Switchyard environment, where we control the coupling between the mechanisms. We provide two analyses: (i) a comparison of the predicted distances with the ground truth on a concrete instance of the environment, and (ii) a visualisation of the displacements of the cells over the updates. For the first, we fix a start state, enumerate every reachable state and compare the predicted distances with the exact ones, which shows where a model does not approximate the distance well. For the second, we use the fact that DeLPI predicts the distance as a sum of displacements over the cells of the grid, so we can plot these displacements individually at every update, which shows which elements of the environment the model "focuses" on.

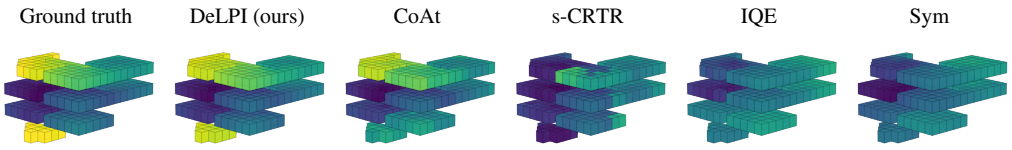


Figure 3: **Learned distance-to-goal volumes against ground truth.** Each panel shows a three-dimensional volume of the predicted distances for selected models. For fixed start state we compute the distances to all reachable target states. The two horizontal axes represent the two spatial dimensions of the grid, and the vertical axis represents the configuration of the gates, so each level of the volume is one configuration. States that are not reachable are omitted. Colour is the distance from the start to that state. Ours is the only one that covers the full range of distances and reproduces the gradient between neighbouring states (see level 0, bottom).

Distance-to-goal volumes. Figure 3 shows the distance-to-goal volume of one instance of Switchyard environment for a few selected models (a random selection of ten instances is presented in Figure 10). On this instance, our model captures the distances within every level of the z-axis (corresponding to different configurations of the gates) and does not show any short-range or long-range biases. Figure 11 in Appendix C shows the calibration curves, which report the whole evaluation

pool. In these curves our model is the only one whose bias stays below one move on average, three times smaller than the other joint-encoding models and six times smaller than the embedding heads.

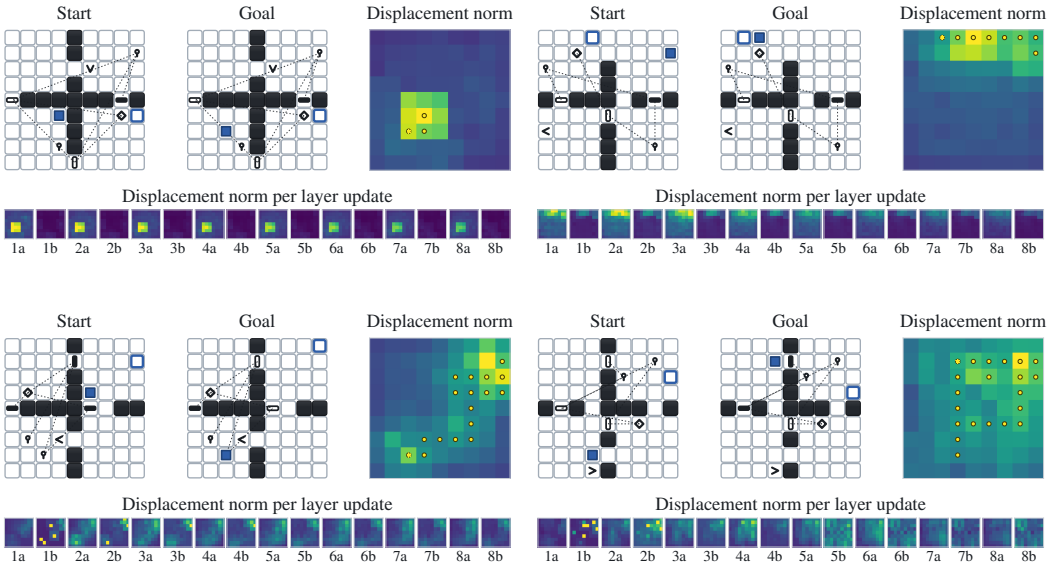


Figure 4: **Latent motion per cell over recurrent updates.** Accumulated displacement for four held-out start–goal pairs, ordered by increasing distance to the goal. For each pair, from left to right, start state, goal state, and the displacement accumulated over all the updates, with the yellow dots showing the cells on a shortest path. Those cells accumulate about twice the motion of the rest. The strip below each pair is the evolution of the recurrent state over the eight iterations and their two blocks. Scales are normalised within each pair.

Latent motion per cell. In Figure 4, we show the accumulated displacement of the cells over the updates for four held-out pairs (chosen so that we cover different ranges of the shortest path length), and Figure 12 shows six more. Cells on a shortest path (overlaid yellow dots) accumulate about twice the displacement of the off-path cells on the short distance pairs, and about 1.4 times on the longest distance pairs, where the displacement spreads over the board. The two blocks of an iteration carry different loads depending on the distance to the goal. On the short pairs block a captures around 75% of the total value, and on the longest ones the two cover it evenly. In the opening iteration block b appears to specialise in highlighting the mechanisms, moving levers on the agent's path about six times more than an average cell and closed gates about twice. From the next iteration on the motion settles onto the route the agent will take. Over the whole pool the motion decomposes along the elements that can be acted on, the agent, the box and the levers (Figure 5).

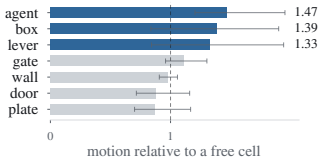


Figure 5: **Decomposed motion by cell type**, relative to a free cell of the same board, over 2,000 held-out pairs. Only the elements that can be acted on rise above the level of the free cell.

Integration ablation. To test how the path integration interacts with the recurrent processing, we train DeLPI-I, which keeps the encoder and the recurrent body and replaces the readout with a linear head on the mean of the final latent vectors, on the 11×11 Switchyard with T ranging from 1 to 8 iterations (Figure 6). With one iteration the two readouts are within 0.01 in correlation. With every doubling of T, DeLPI improves from 0.93 to 0.99 while DeLPI-I falls from 0.92 to 0.88, with a growing spread across seeds. Hence, in this goal-conditioned setting, the two mechanisms complement each other.

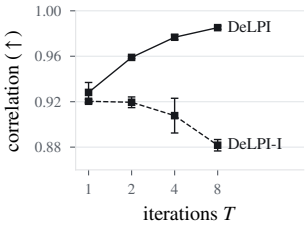


Figure 6: **Correlation against number of iterations,** with the same encoder and recurrent body in both models.

6 Related Work

As mentioned in Section 1, heuristic search and GCRL face the same estimation problem. Goal-conditioned value functions (Schaul et al., 2015), including the quasimetric heads such as IQE (Wang & Isola, 2022) and MRN (Liu et al., 2023), encode the start and the goal separately, so relational information must survive compression into independent vectors before the distance is evaluated. CRTR (Ziarko et al., 2025) also embeds the two states separately, trained with a contrastive loss, and, like us, targets problems with nontrivial combinatorial structure. Because our setting provides ground-truth distances, our s-CRTR baselines build on the supervised baseline reported in their paper, which encodes the pair of states jointly and classifies their distance into bins. Joint-encoding models avoid the embedding bottleneck by stacking the two states at the input of one network (Groshev et al., 2018), CoAt (Chrestien et al., 2021) extends this architecture with attention, and Chrestien et al. (2023) show that ranking states guides search better than estimating costs.

A second group of related methods makes planning an iterative computation inside the network. VINs (Tamar et al., 2016) unroll value iteration on a learned latent MDP, DRC (Guez et al., 2019) learns planning-like behaviour with stacked recurrent convolutions, and weight-tied recurrent networks extrapolate to harder instances by iterating longer (Dehghani et al., 2019; Bansal et al., 2022). VINs plan over an explicit action space in a fixed map, so they do not estimate the distance between two arbitrary states. We combine the two threads, processing a joint-encoding encoding recurrently and reading the distance from the accumulated displacement of the latent vectors instead of decoding a scalar from a static representation. Appendix E discusses further related work.

7 Conclusion

Our main contribution is a primitive for planning distance estimation in which the prediction is the accumulated displacement of a decomposed latent representation over recurrent updates, instead of a scalar decoded from static embeddings. We instantiated it in DeLPI and evaluated it on four environments whose factors interact: Switchyard, where the degree of coupling is under our control, Sokoban, the Rubik's cube and Craftax. On Switchyard, Sokoban and Craftax, DeLPI outperforms both the learned heuristics of the planning literature and the embedding-based metric and quasimetric models of goal-conditioned reinforcement learning, and on the Rubik's cube it matches the strongest baseline with 3% of its parameters. Its lead is wider when the estimates have to guide a planner than in the direct metrics: on Sokoban a greedy policy on DeLPI solves almost twice as many boards as the same policy on the strongest baseline, at a quarter of the search cost. The accumulated displacement is also readable: the cells on a shortest path accumulate about twice the motion of the rest, and the motion decomposes along the elements the agent can act on.

The primitive can be instantiated in other recurrent architectures, and because of its distinctiveness from other architectural ideas, we see it as a good candidate for the agentic architecture search that is becoming a common practice (Liu et al., 2025; Jeong et al., 2026).

Our experiments were conducted mostly in a supervised setting. We use this setting on purpose to demonstrate the effectiveness of the introduced primitive without other confounders. Exploring how this primitive can be used in an unsupervised or reinforcement learning setting is left for future work. Lastly, the architecture lends itself to adaptive computation. The prediction is a sum of the displacements produced by the individual updates, so it is defined after every update and the recurrence can be stopped after any number of iterations, which would let the model spend more computation on the pairs that need it. Our initial experiments in this direction were not successful, and we leave the question for future work.

AI USE STATEMENT

In this work, we used an LLM-based model (Claude Fable) to provide feedback on the text, to build the tables, to run scripts that aggregate results, and to launch pre-defined experiments and evaluations. First drafts of some result paragraphs were written by the assistant from the tables and rewritten by the authors. We have not used generative AI tools to generate data, to develop the model or its motivation, or to propose hypotheses. All LLM generated code was checked by the authors, and every reported number was verified independently. We take full responsibility for the final content of this work, including text, claims or produced artifacts.

REPRODUCIBILITY STATEMENT

The source code, the generators of Switchyard and of the Craftax pairs, and the result files behind every number are available in an anonymous repository at `https://anonymous.4open.science/r/delpi-code-D8F3/`. The Sokoban boards and the CRTR networks come from Ziarko et al. (2025) and the boards of the ranking experiments from Chrestien et al. (2023). Appendix A describes the Switchyard environment, its labels and its split, and Appendix B the other three benchmarks, every baseline, the training protocol, the ranking objective and the planning metrics.

REFERENCES

Forest Agostinelli, Stephen McAleer, Alexander Shmakov, and Pierre Baldi. Solving the Rubik's cube with deep reinforcement learning and search. *Nature Machine Intelligence*, 1(8):356–363, 2019. doi: 10.1038/s42256-019-0070-z.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/hash/453fadbd8a1a3af50a9df4df899537b5-Abstract.html`.

Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. End-to-end algorithm synthesis with recurrent networks: Extrapolation without overthinking. In *Advances in Neural Information Processing Systems*, volume 35, pp. 20232–20242. Curran Associates, Inc., 2022. doi: 10.52202/068431-1471. URL `https://proceedings.neurips.cc/paper_files/paper/2022/hash/7f70331dbe58ad59d83941dfa7d975aa-Abstract-Conference.html`.

Leah Chrestien, Tomáš Pevný, Antonín Komenda, and Stefan Edelkamp. Heuristic search planning with deep neural networks using imitation, attention and curriculum learning. arXiv preprint arXiv:2112.01918, 2021. Presented at the PRL Workshop at IJCAI 2022.

Leah Chrestien, Stefan Edelkamp, Antonín Komenda, and Tomáš Pevný. Optimize planning heuristics to rank, not to estimate cost-to-goal. In *Advances in Neural Information Processing Systems*, volume 36, pp. 25508–25527, 2023. doi: 10.52202/075280-1110. URL `https://proceedings.neurips.cc/paper_files/paper/2023/hash/50ea4dbd1cff6bd3daef939eff10c092-Abstract-Conference.html`.

Konrad Czechowski, Tomasz Odrzygóźdź, Marek Zbysiński, Michał Zawalski, Krzysztof Olejnik, Yuhuai Wu, Łukasz Kuciński, and Piotr Miłoś. Subgoal search for complex reasoning tasks. In *Advances in Neural Information Processing Systems*, volume 34, pp. 624–638, 2021. URL `https://proceedings.neurips.cc/paper/2021/hash/05d8cccb5f47e5072f0a05b5f514941a-Abstract.html`.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HyzdRiR9Y7`.

Edward Groshev, Maxwell Goldstein, Aviv Tamar, Siddharth Srivastava, and Pieter Abbeel. Learning generalized reactive policies using deep neural networks. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS 2018)*, pp. 408–416. AAAI Press, 2018. doi: 10.1609/icaps.v28i1.13872. URL https://ojs.aaai.org/index.php/ICAPS/article/view/13872.

Arthur Guez, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sébastien Racanière, Théophane Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, Greg Wayne, David Silver, and Timothy Lillicrap. An investigation of model-free planning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2464–2473. PMLR, 2019. URL https://proceedings.mlr.press/v97/guez19a.html.

Seokhoon Jeong, Mijung Kim, and Taehwan Kim. Agentic neural architecture search. *arXiv preprint arXiv:2607.07984*, 2026. URL https://arxiv.org/abs/2607.07984.

Bo Liu, Yihao Feng, Qiang Liu, and Peter Stone. Metric residual network for sample efficient goal-conditioned reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(7):8799–8806, 2023. doi: 10.1609/aaai.v37i7.26058. URL https://ojs.aaai.org/index.php/AAAI/article/view/26058.

Yixiu Liu, Yang Nan, Weixian Xu, Xiangkun Hu, Lyumanshan Ye, Zhen Qin, and Pengfei Liu. AlphaGo moment for model architecture discovery. *arXiv preprint arXiv:2507.18074*, 2025. URL https://arxiv.org/abs/2507.18074.

Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 11525–11538. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/hash/8511df98c02ab60aea1b2356c013bc0f-Abstract.html.

Michael Matthews, Michael Beukman, Benjamin Ellis, Mikayel Samvelyan, Matthew Thomas Jackson, Samuel Coward, and Jakob Nicolaus Foerster. Craftax: A lightning-fast benchmark for open-ended reinforcement learning. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 35104–35137. PMLR, 2024. URL https://proceedings.mlr.press/v235/matthews24a.html.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1312–1320, Lille, France, 2015. PMLR. URL https://proceedings.mlr.press/v37/schaul15.html.

Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks: Learning generalizable representations for visuomotor control. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4732–4741. PMLR, 2018. URL https://proceedings.mlr.press/v80/srinivas18b.html.

Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems*, volume 29, pp. 2154–2162. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/hash/c21002f464c5fc5bee3b98ced83963b8-Abstract.html.

Tongzhou Wang and Phillip Isola. Improved representation of asymmetrical distances with interval quasimetric embeddings. In *NeurIPS 2022 Workshop on Symmetry and Geometry in Neural Representations (NeurReps)*, 2022. doi: 10.48550/arXiv.2211.15120. URL https://arxiv.org/abs/2211.15120. Proceedings Track.

Tongzhou Wang, Antonio Torralba, Phillip Isola, and Amy Zhang. Optimal goal-reaching reinforcement learning via quasimetric learning. In Andreas Krause, Emma Brunskill, Kyunghyun

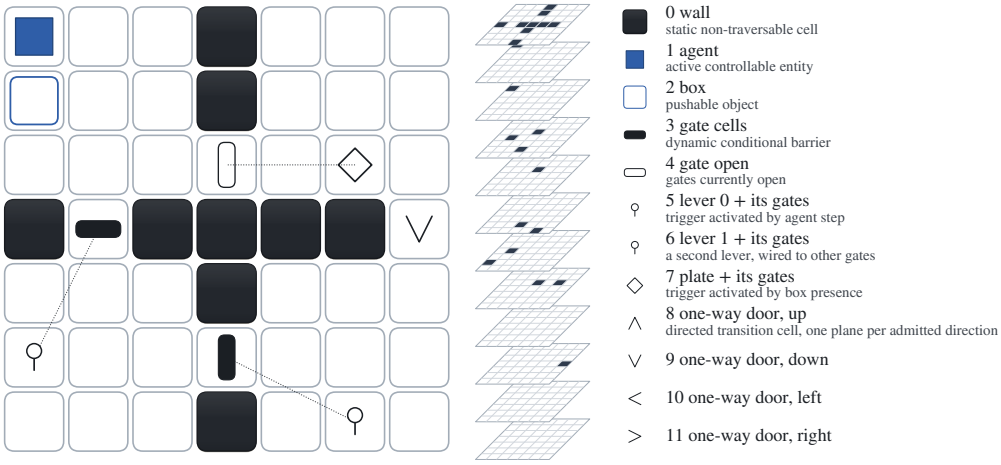
Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 36411–36430. PMLR, 2023. URL https://proceedings.mlr.press/v202/wang23al.html.

Yuhui Wang, Qingyuan Wu, Dylan R. Ashley, Francesco Faccio, Weida Li, Chao Huang, and Jürgen Schmidhuber. Scaling value iteration networks to 5000 layers for extreme long-term planning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 64914–64936. PMLR, 2025. URL https://proceedings.mlr.press/v267/wang25do.html.

Alicja Ziarko, Michał Bortkiewicz, Michał Zawalski, Benjamin Eysenbach, and Piotr Miłoś. Contrastive representations for temporal reasoning. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://openreview.net/forum?id=UlkcH5Ccrk.

A THE SWITCHYARD ENVIRONMENT

This appendix describes the Switchyard environment. Figure 7 shows one instance. The generator and the instances used in the experiments are in the repository linked in the reproducibility statement.



* Note: we draw wires on the board so the links between triggers and gates are visible.

Figure 7: **A Switchyard instance and its state representation.** On the left, the board: walls, gates, levers, plate and one-way doors, with visible wires linking triggers and gates. On the right, the twelve binary channels, one per cell type, along with a short description of each element and layer.

Layout. An instance is a square grid at one of six sizes, from 9×9 to 19×19. Walls along the middle row and the middle column divide it into four rooms, and each wall segment between two neighboring rooms contains one passage. Three of the four passages are gates and the fourth is an open doorway. Half of the wall segments, chosen at random, contain a second open doorway. The generator then places two levers, one plate and one one-way door on free cells, connects each lever to a random non-empty subset of the gates, and does the same for the plate. These counts describe the instances of the size ladder in Figure 2; the second ladder in Appendix C places more levers and one-way doors as the grid grows, with the counts given in Figure 9.

Objects and rules. The agent moves one cell at a time in four directions. Moving into the box pushes it one cell further if that cell is free, which excludes walls, closed gates, the one-way door and the grid boundary. A closed gate blocks the agent and the box alike. When the agent stands on a lever, a fifth action pulls it, which opens every closed gate the lever is connected to and closes every open one. The plate holds the gates it is connected to open for as long as the box stands on it. The one-way door admits the agent only when it enters in the door's direction and never admits the box. A state consists of the positions of the agent and the box and the open or closed status of each gate.

Observations. A state is represented as the grid with 12 binary channels: the walls, the agent, the box, the gate cells, the open gates, one channel per lever marking its cell and the gates it is connected to, one channel marking the plate and the gates it holds, and four channels for the one-way door, one per direction. Which gates a lever or the plate controls is therefore part of the observation, and a model has to read it from the grid.

Distance labels. For a start state sampled at random in an instance, breadth-first search over the state graph gives the exact distance of every reachable state. We sample goal states stratified over distance, a few at every distance up to a maximum that depends on the grid size (28, 36, 44, 70, 66 and 72 steps for the six sizes; the 15×15 grid has the longest distances relative to its size, as noted in Section 4), and label each start-goal state pair with the exact number of steps between them.

Split. We generate 683 instances at each grid size and hold out every fourth one for testing, so the layouts and the mechanism wiring of the test instances are unseen during training. The training pairs come from 6,800 breadth-first searches over the training instances, and the test pairs from the same number over the test instances.

B EXPERIMENTAL DETAILS

This appendix gives the data details of the Sokoban, Rubik's cube and Craftax benchmarks, the baseline architectures, the training hyperparameters, the ranking-objective protocol and the planning metrics, in the order of Section 3.

B.1 BENCHMARK DATA

Sokoban. A state has one binary channel per cell type: wall, floor, empty target, box, box on a target, agent, and agent on a target. We compute the labels by a backward breadth-first search from the goal state, which gives the exact number of actions still needed from every state of the level. The training set holds 3.78M states from 80k levels, and the validation set 20k states from 4.1k levels. Accuracy is measured on 20k states from the 1,000 levels of the test trajectories that CRTR holds out.

Rubik's cube. We enumerate every state within nine turns of the solved state by breadth-first search. Depth ten would need about 150 GB of memory, more than was available to us. Because the moves of the cube form a group, composing a uniformly random cube A with a state at distance d yields a cube B at distance exactly d. We generate pairs for $d = 1, \ldots, 9$ and add 5% identical pairs, which have $d = 0$. The training, validation and test splits contain disjoint sets of cubes and hold 557k, 11.3k and 45.7k pairs.

Craftax. A state is determined by the grid, the agent's position and facing direction, and the inventory. The randomness of the dynamics is fixed and no creatures spawn. The six initial states of a world are its generated start and five states reached from it by sequences of random actions. From each of them we enumerate every state within 14 actions, the deepest enumeration that remains feasible. The dataset contains 767k pairs from 40 worlds, split by world into 24 training, 8 validation and 8 test worlds with 461k, 151k and 156k pairs. The 35 channels of a state are a one-hot encoding of the block type at each cell (17), a channel marking the agent's cell (1), a one-hot encoding of the facing direction broadcast over the whole grid (5), and one channel per inventory item (12), constant over the grid and proportional to the number the agent holds.

B.2 BASELINES

Figure 8 shows a sketch of DeLPI, its two variants and the baselines side by side. The paragraphs below give the configuration of every baseline.

Embedding models. On Switchyard and Sokoban, the shared encoder is the convolutional network of CRTR: a 3×3 convolution with 64 channels, batch normalization and ReLU, four residual pairs of 3×3 convolutions, global average pooling and a linear layer to a 64-dimensional embedding. On Sokoban we load CRTR's code unmodified, and on Switchyard we use a re-implementation with the same parameters. Sym takes the L_1 distance between the two embeddings. IQE and MRN are the quasimetric heads of the torchqmet library, IQE with 64 dimensions in components of 16 and max-mean reduction, MRN with 64 dimensions. Both satisfy the triangle inequality. On Craftax, the three heads use the encoder selected in a sweep of 18 encoders under the same heads: two 3×3 convolutions with ReLU, of 64 and 256 channels, a fixed sine-cosine positional code added to every cell, and the mean over the cells as a 256-dimensional embedding, with IQE and MRN re-implemented on the 256 dimensions. The sweep raised the correlation of every head by 0.03 to 0.06 over CRTR's encoder and left the gap to the joint-encoding models intact.

s-CRTR networks on the stacked pair. The convolutional network of CRTR reads the $2N$ stacked channels on all three gridworlds. The s-CRTR$_{\text{class}}$ has one output bin per unit of distance, 150 on Switchyard and Sokoban and 16 on Craftax, and is trained with cross-entropy on the integer distance. We take the median of the predicted distribution, whereas CRTR's supervised baseline takes the most likely bin. The s-CRTR$_{\text{reg}}$ has one output with a softplus and is trained with the smooth L_1 loss.

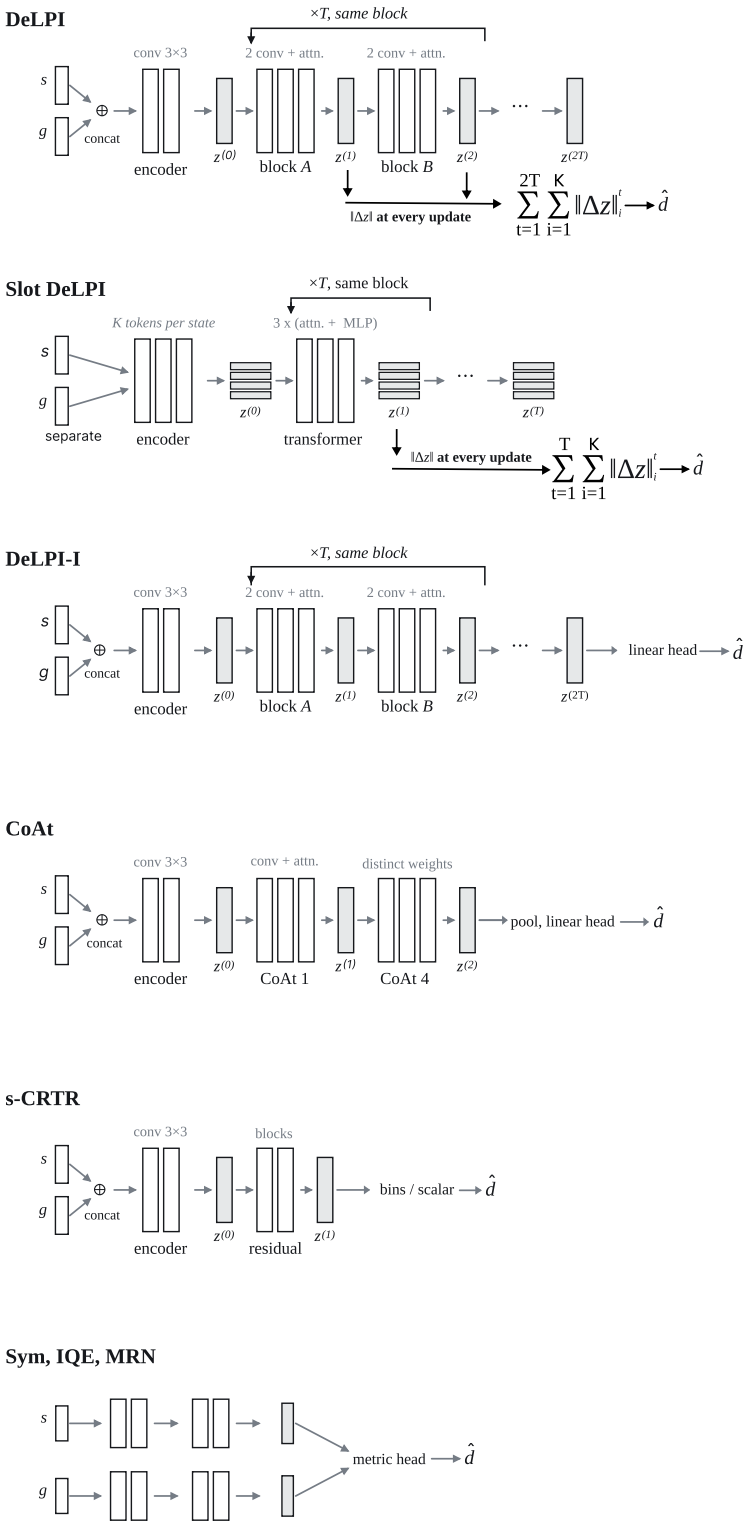


Figure 8: **The architectures compared in this paper.** DeLPI and its two variants, the slot encoder (Appendix D) and the linear readout DeLPI-I (Section 5), and the three baseline families: CoAt, the s-CRTR networks on the stacked pair, and the embedding models, which encode the two states separately and compare them in a metric head. For every model, $z^{(t)}$ is the latent collection after update t and $\hat{d}$ the predicted distance. *e-bins/scalar* are the two heads of s-CRTR (classification and regression).

CoAt. On Switchyard, on Sokoban and under the ranking objective, our implementation follows the code released with Chrestien et al. (2023). Seven 3×3 convolutions with 64 filters and ReLU, with the input pair concatenated to the output of each, are followed by four CoAt blocks. Each block is a 3×3 convolution with 180 filters whose output is split into queries, keys and values for two-head self-attention over all grid cells, followed by the concatenation of the attention output, the convolution output with a sine-cosine positional encoding, and the input pair. The network ends in the mean over all cells, a dense layer with 256 units and ReLU, and a linear output, with a softplus on Sokoban and under the ranking objective and without one on Switchyard. On the cube and Craftax, the implementation does not concatenate the input pair, concatenates a 16-channel positional encoding after the convolutions and after every block, adds a 1×1 convolution of the block input to the attention output, and ends in two dense layers with ReLU and a linear output with a softplus. The original paper reads the output at the cell of the agent, whereas both implementations average over all cells, as the released code does. On Craftax the widths are the published 64 and 180 filters with 256 dense units. On the cube, the 1.83M configuration doubles the 64 filters of the convolutions to 128 and the dense units to 512 and keeps 180 filters in the blocks. The 0.52M configuration has 32 filters, 96 filters in the blocks and 512 dense units, and the 0.06M configuration has 16, 24 and 128, the best of six configurations with 50k to 61k parameters, chosen on the validation split.

Chrestien CNN. We also evaluated the convolutional network of Groshev et al. (2018), in the re-implementation of Chrestien et al. (2021), on the Rubik's cube and Craftax, at four seeds each. CoAt dominated it on every metric on both benchmarks, for example 0.918 against 0.933 in test correlation on the cube and 0.980 against 0.983 on Craftax, so we omit it from the tables.

s-CRTR$_{\text{MLP}}$. The residual MLP of CRTR is loaded unmodified: a linear layer with layer normalization and ReLU, residual blocks of two linear layers with layer normalization, and a linear output with a softplus. Its input is the 54 color indices of each cube as a vector, start followed by goal, whereas CoAt zero-pads the 9×12 grid to 12×12. CRTR trains it as a classifier, whereas we train it with the smooth L_1 loss. The 8.5M configuration has hidden width 1024 and depth 8, the shape CRTR uses on the cube. The 0.5M configuration has width 244 and depth 8, and the 0.06M configuration width 105 and depth 4, the best of six configurations with 50k to 61k parameters, chosen on the validation split.

B.3 Training

Unless stated otherwise, models are trained with batch size 128 and learning rate 10^{-3}.

Switchyard. All models are trained for 160k steps, separately at each grid size.

Sokoban. All models are trained for 33 epochs. The learning rate (10^{-4}, $3 \cdot 10^{-4}$ or 10^{-3}) and the batch size (128 or 500) are selected on the validation set for every model.

Rubik's cube. All models are trained for 320k steps. As data augmentation, each training pair is transformed by one of the 24 rotations of the whole cube, which permutes the face colors and leaves the label unchanged.

Craftax. All models are trained for 160k steps.

B.4 Ranking objective

Under the L* objective, every state on an optimal path must receive a lower $f = g + h$ than any other state in the open list. The loss of Chrestien et al. (2023) is a pairwise ranking loss over such state pairs, $\mathrm{softplus}(f_{\mathrm{path}} - f_{\mathrm{other}})$, which penalizes an on-path state whose f is not below that of an off-path state. We train DeLPI and CoAt on the 10×10 Sokoban boards with three boxes released by the authors, with 20k training boards per seed, 200 held-out test boards and three random seeds. Under static L*, the state pairs come from an exact solution path and the states that A* discovers off the path. Under in-loop L*, the A* that collects the training states has a budget of 2,000 expansions and starts from states at most n steps from the goal, where n grows to 40 over training. Both models are trained for 20k updates with at most 512 state pairs per update. The input to DeLPI is the same seven-channel state as on Sokoban, and the input to CoAt is the five-channel state and goal images of Chrestien et al. (2023).

B.5 PLANNING METRICS

Switchyard. For success of the *greedy* policy, we report the fraction of 200 test pairs per grid size, the same pairs for every model, for which the greedy policy reaches the goal state within $\max(20, 4d)$ steps, where d is the true distance. The policy keeps no record of visited states, so it can also fail by cycling between them. For the success of the *search*, we report the fraction of the same pairs solved within a budget of 100 expansions.

Sokoban. On Sokoban, we use the CRTR planners unchanged on the 1,000 evaluation boards of CRTR(Ziarko et al., 2025). The *greedy* policy never returns to a visited state and gives up when every successor has been visited or after 6,000 steps; its success is the fraction of boards solved. For the success of the *search*, we report the fraction of boards solved within a budget of 6,000 generated nodes, and *search cost* is the mean number of nodes generated per board.

Rubik's cube. For the success of the *greedy* policy, we report the fraction of 180 held-out test pairs, 20 per exact distance from 1 to 9, for which the policy reaches the goal cube within $\max(20, 4d)$ steps, where d is the true distance, taking the first successor on ties. For the success of the *search*, we report the fraction of the same pairs solved by A* with the predicted distance as the heuristic within a budget of 1,000 heuristic evaluations, where every generated state is evaluated once. Unlike the best-first search of the other benchmarks, A* adds the cost already paid to the predicted distance.

Craftax. For the success of the *greedy* policy, we report the fraction of 400 test pairs, 50 per test world, for which the policy reaches the goal within d steps, where d is the true distance. The policy excludes actions that leave the state unchanged. For the success of the *search*, we report the fraction of 200 test pairs solved by the same search as on Sokoban, within the same budget of 6,000 generated nodes.

C EXTENDED RESULTS

This appendix extends the results of Sections 4 and 5.

Scaling with the number of mechanisms. Figure 9 reports DeLPI and CoAt on a second ladder of Switchyard instances, in which the number of mechanisms grows with the grid. Each model is trained with two seeds per grid size. On grid sizes 11×11 and larger, DeLPI outperforms CoAt in accuracy, with 62 to 72% of its error, and solves more pairs under both the greedy policy and search. At 9×9, the densest instances with twelve objects on 81 cells, the two models are within seed spread of each other: CoAt is ahead by 0.006 in correlation and by 0.007 in search, and DeLPI is ahead in error and under the greedy policy. DeLPI fits the training instances there but transfers less to the held-out instances: its training MAE is 0.6 steps against 2.9 on the held-out instances, whereas CoAt has 1.4 against 2.9. Both models fall below their values on the size ladder at the same grid size, by 0.03 to 0.09 in correlation, so scaling the number of mechanisms is harder for both models than scaling the grid size. On the size ladder at 13×13, DeLPI with widths 32, 64 and 96 reaches correlations of 0.979, 0.985 and 0.988 over two seeds. The smallest, with 76k parameters, already outperforms CoAt at 1.81M parameters.

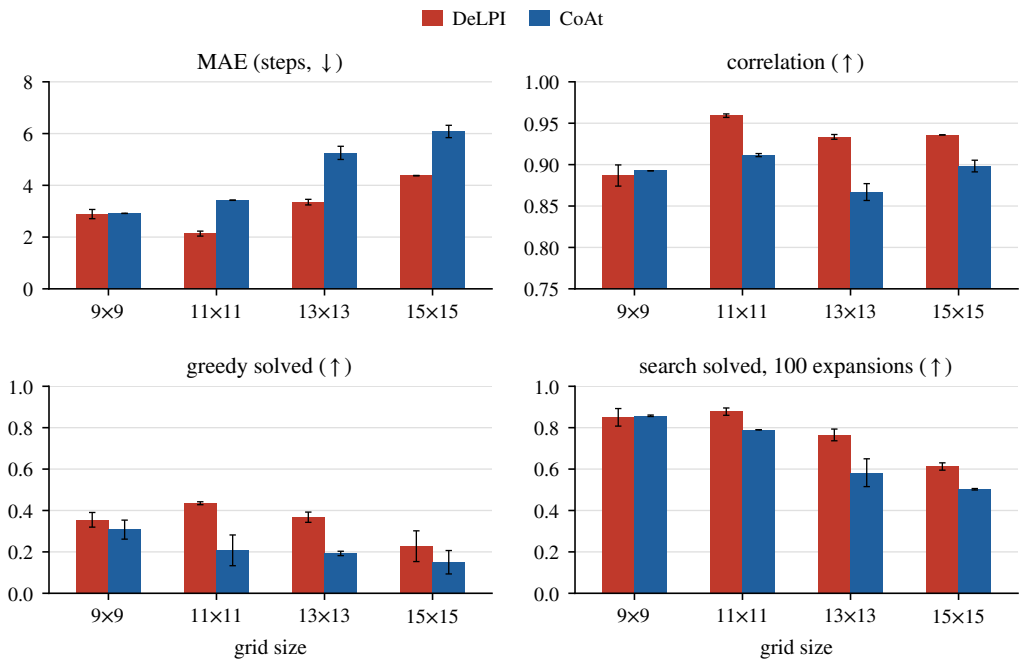


Figure 9: **Scaling with the number of mechanisms.** DeLPI and CoAt on Switchyard instances with four gates, three levers and two one-way doors at 9×9 and 11×11, and four gates, four levers and three one-way doors at 13×13 and 15×15. Accuracy over all test pairs of held-out instances. Greedy and search success on 200 test pairs per grid size.

Distance-to-goal volumes. Figure 10 reports the distance-to-goal volume of the Switchyard environment for ten random held-out instances. Our model recovers the gradient within every level and is the only one that consistently reaches the most distant states. CoAt captures the gradient between neighbouring states but in general does not reach the far ones, and on some instances it also shows abrupt changes between neighbours. The s-CRTR variants usually resolve one configuration. The classifier is the one with the most abrupt changes between neighbouring states, with whole levels that are artefactual, and even so it stays closer to the ground truth than the regressor, whose changes are smooth but entirely artefactual. The embedding models resolve a single level and spread a homogeneous gradient over the rest, over-predicting the states that are close and under-predicting the far ones, and in many cases they assign a whole level to a layer without any change in the environment that would justify it.

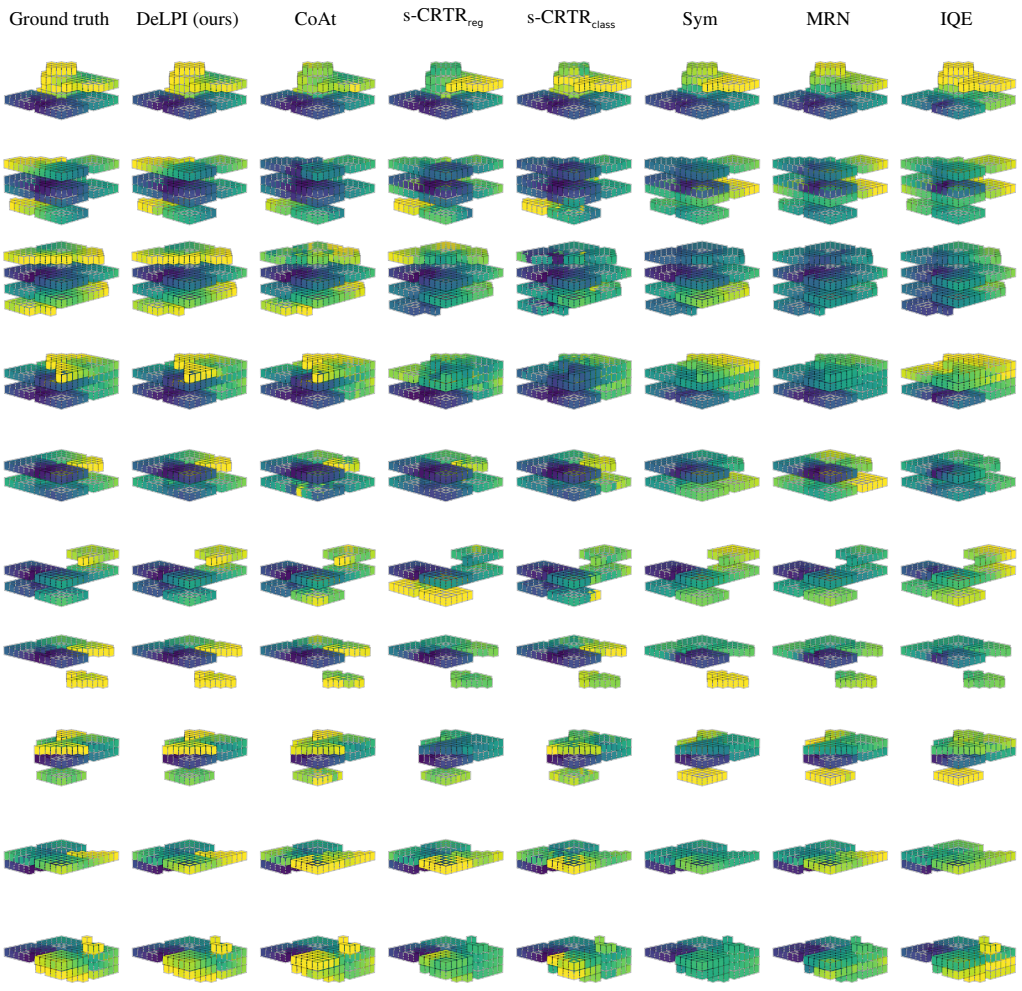


Figure 10: **Learned distance-to-goal volumes against ground truth.** Each panel shows a three-dimensional volume of the predicted distances for selected models. Each row is one of ten random held-out instances. For a fixed start state we compute the distances to all reachable target states. The two horizontal axes represent the two spatial dimensions of the grid, and the vertical axis represents the configuration of the gates, so each level of the volume is one configuration. States that are not reachable are omitted. Colour is the distance from the start to that state, by breadth-first search for the ground truth; all panels in a row share one scale. Ours is the only one that covers the full range of distances and reproduces the gradient between neighbouring states.

Calibration. Figure 11 reports the calibration of every model over the whole evaluation pool of 11,650 held-out pairs, the counterpart of the biases discussed in Section 5. Every baseline compresses the range, over-predicting the states that are close and under-predicting the far ones, and the effect grows with the distance. DeLPI is the only model whose bias stays below one move on average, three times smaller than the other joint-encoding models and six times smaller than the embedding models, and the spread of its predictions is two to six times narrower.

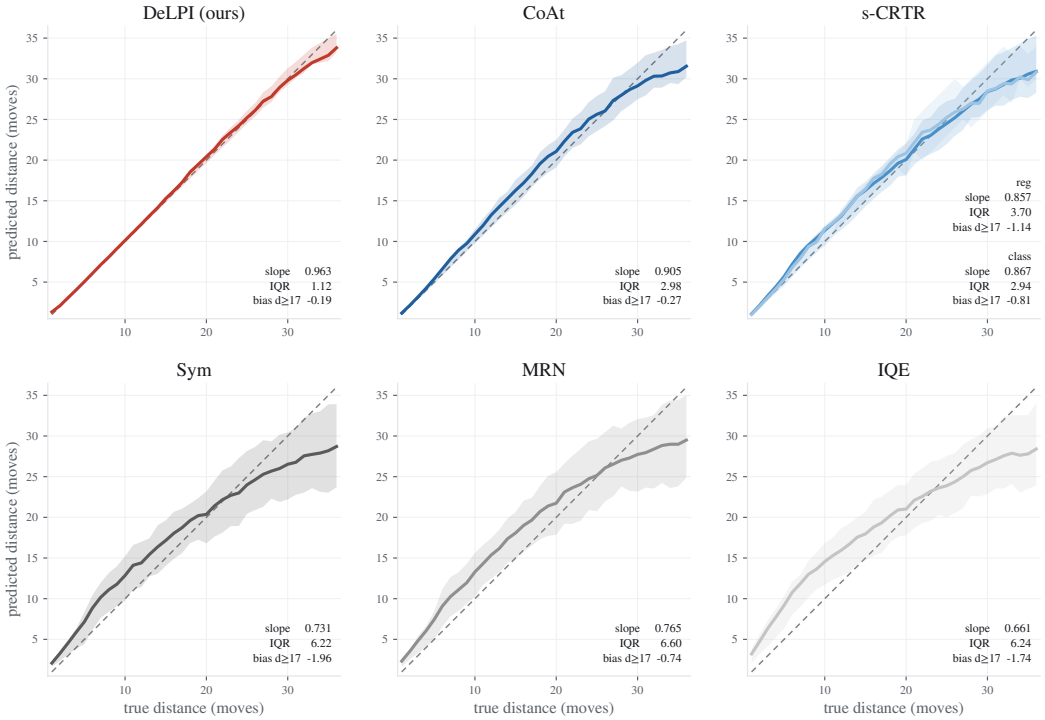


Figure 11: **Predicted against true distance.** Binned mean prediction at each true distance, over the same 11,650 held-out pairs for every model; the shaded band is the interquartile range of predictions in that bin, and IQR is its mean width. The dashed line is perfect calibration; slope is the least-squares fit of predicted on true distance over all pairs. Bias is the mean signed error, reported on long-range pairs ($d \geq 17$).

Latent motion. Figure 12 reports the latent motion of Figure 4 on six further held-out pairs. The cells on a shortest path accumulate about twice the motion of the rest, and the specialisation of the first iteration is easiest to see in the last row, where block 1b lights up the mechanisms on the route the agent will take.

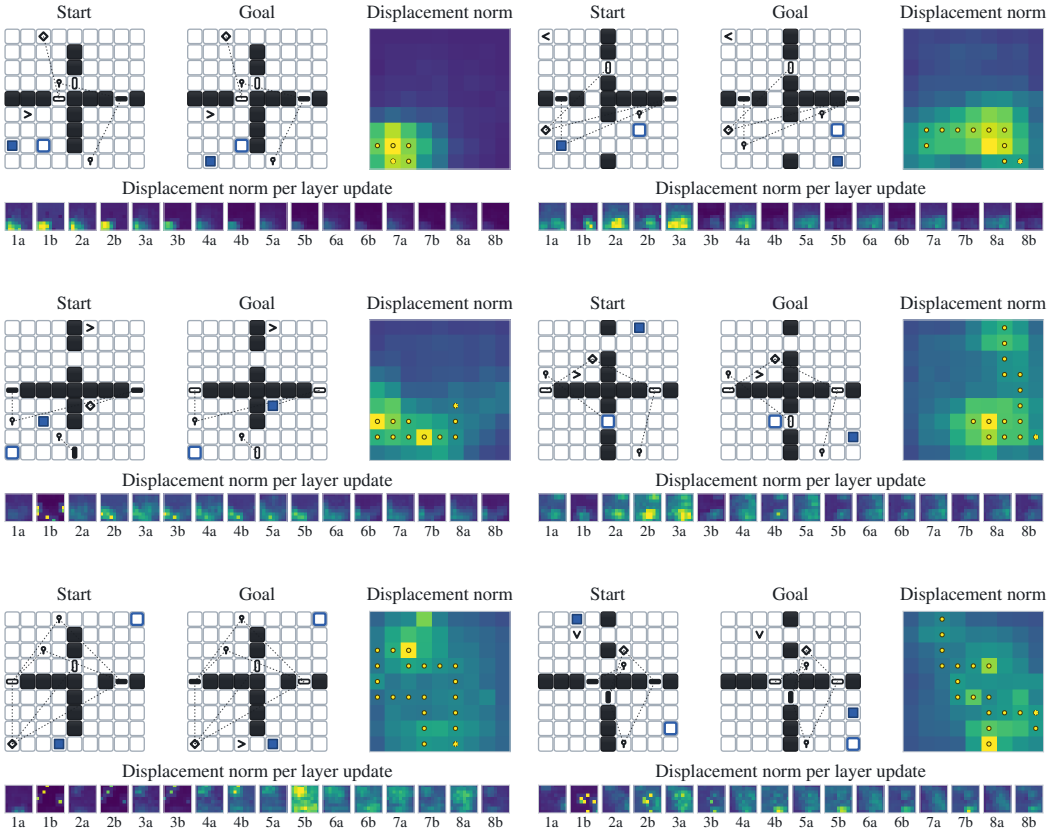


Figure 12: **Latent motion per cell over recurrent updates.** Accumulated displacement for six held-out start–goal pairs, ordered by increasing distance to the goal. For each pair, from left to right, the start state, goal state, and the displacement accumulated over all updates, with yellow dots showing the cells on a shortest path. Those cells accumulate about twice the motion of the rest. The strip below each pair is the evolution of the recurrent state over the eight iterations and their two blocks. Scales are normalised per pair.

D THE SLOT VARIANT OF DELPI

Section 2 presents DeLPI with a convolutional encoder that keeps one latent vector per grid cell. This appendix compares it with a variant that replaces this encoder with learned slot queries, and with CoAt, the strongest baseline.

Slot DeLPI. A shared convolutional encoder maps each state separately to per-cell features with a learned positional encoding, and K learned queries attend over these features (Locatello et al., 2020), so that each state is summarized by K slot vectors, which are free to represent entities of the environment rather than positions. The start slots and the goal slots, with role and identity embeddings, form one token sequence that a transformer updates over $T = 8$ iterations, with three layers per iteration and the same weights at every iteration. After every iteration the goal tokens are reset to their encoding, which plays the role of the re-injection in Section 2, and on Sokoban a frozen copy of the start slots is appended as a third group of tokens. The readout is the path integration of Section 2 over the K start slots. The latent width is 256 with four attention heads, and on Sokoban each state carries two extra channels with the box and target positions of the goal. The number of slots, the number of layers and, on Sokoban, the number of iterations was selected on the validation split of each benchmark from a screen of four to six configurations around this setting, which chose $K = 16$ on Switchyard and $K = 24$ on Sokoban.

Table 5: Slot DeLPI against DeLPI and CoAt: direct metrics on *Switchyard* at 11×11 under the protocol of Figure 2 and on *Sokoban* with labels from shortest paths under the protocol of Table 1, mean $\pm$ standard deviation over four training seeds.

	Switchyard 11×11			*Sokoban*		
model	`params` (M)	`MAE` (↓)	`corr` (↑)	`params` (M)	`MAE` (↓)	`corr` (↑)
DeLPI	0.55	$\mathbf{0.92}_{\pm 0.06}$	$\mathbf{0.985}_{\pm 0.002}$	0.53	$\mathbf{0.85}_{\pm 0.03}$	$\mathbf{0.993}_{< 0.001}$
CoAt	1.81	$2.05_{\pm 0.12}$	$0.951_{\pm 0.004}$	1.71	$1.64_{\pm 0.06}$	$0.981_{\pm 0.001}$
slot DeLPI	2.08	$2.95_{\pm 0.13}$	$0.901_{\pm 0.005}$	2.13	$2.02_{\pm 0.03}$	$0.972_{\pm 0.001}$

Comparison. Table 5 shows that slot DeLPI falls behind DeLPI and CoAt on both benchmarks, with four times the parameters of DeLPI: the correlation drops from 0.985 to 0.901 on Switchyard and from 0.993 to 0.972 on Sokoban, and the MAE more than doubles. No other number of slots or layers in the hyperparameter search improved it. The slot alternative is conceptually more desirable because the model could potentially learn to simulate the movements and interaction only of the relevant objects, but we did not manage to make it work as well as the grid encoder of Section 2.

E EXTENDED RELATED WORK

Goal-conditioned value estimation. The geometric family models the value function as a metric or quasimetric over learned embeddings. The lineage traces back to Universal Value Function Approximators (Schaul et al., 2015), which factor the state and the goal into entirely independent embedding vectors. Since many environments feature irreversible transitions, the field evolved toward quasimetric heads such as Interval Quasimetric Embeddings (Wang & Isola, 2022) and Metric Residual Networks (Liu et al., 2023). These structured heads can represent any quasimetric over the embeddings (Wang et al., 2023), but they come with a representational bottleneck: the two states are embedded separately, so relational information must survive compression into independent vectors before the distance is evaluated. Proposed to overcome the limitations of contrastive reinforcement learning, CRTR (Ziarko et al., 2025) uses contrastive losses over trajectories so that the similarity between the embeddings of two states reflects their temporal proximity, and this similarity itself serves as the planning heuristic. The two states are encoded separately by one shared encoder. The paper evaluates on problems with nontrivial combinatorial structure, Sokoban and the Rubik's cube among them, which is the regime we study. Their paper also reports a supervised baseline that takes the pair of states jointly and classifies the distance into discrete bins. Because our setting provides exact distance labels, our s-CRTR baselines build on this supervised baseline rather than on the contrastive representation.

Learned heuristics on the stacked pair. Early work such as Groshev et al. (2018) demonstrated the value of stacking the start and goal states together as input channels into a single convolutional network to learn search heuristics, and later extensions such as CoAt (Chrestien et al., 2021) build directly on this foundation. By feeding both states into a single unified architecture, these models avoid the embedding bottleneck and allow distant features of the board to interact within a single pass. The joint-encoding structure is also flexible with respect to the training signal: Chrestien et al. (2023) optimize the same architecture to rank states for search instead of estimating exact costs.

Recurrent planning. A second lineage embeds a differentiable planning computation directly within the network. Value Iteration Networks (Tamar et al., 2016) and Universal Planning Networks (Srinivas et al., 2018) established this paradigm, and recent extensions such as Dynamic Transition VINs (Wang et al., 2025) show that it scales to very long horizons. While these models are conceptual precursors of our approach, they plan over an explicit action space with a learned or given transition model, whereas we estimate the distance between two observations without either. DRC (Guez et al., 2019) shows that stacked recurrent convolutions learn planning-like behaviour on Sokoban without an explicit planning operator. In the broader domain of sequence modeling, Universal Transformers (Dehghani et al., 2019) and Deep Thinking models (Bansal et al., 2022) demonstrate that weight-tied recurrent blocks let networks extrapolate to harder problems by iterating for more steps, a property our depth-recurrent trunk inherits.