METRO-WM: LONG-HORIZON LATENT PLANNING WITH REALISABLE SUB-GOALS

Anonymous authors
Paper under double-blind review

ABSTRACT

Model-predictive control with Joint-Embedding Predictive Architectures (JEPAs) provides a strong zero-shot goal-reaching planner, but it is only effective over short planning horizons. Hierarchical extensions attempt to bridge this gap by learning a macro planner to predict intermediate latent sub-goals to guide the micro planner. In this work, we demonstrate that unconstrained latent sub-goal prediction is fundamentally flawed. A rigorous evaluation reveals that a leading state-of-the-art macro planner routinely emits physically unrealisable sub-goals. To resolve this, we introduce METRO-WM, a hierarchical framework that issues sub-goals by retrieving genuine states from prior experience rather than generating ungrounded latent vectors. Specifically, METRO-WM constructs a graph whose vertices are observed frames from offline expert demonstrations or random-action trajectories, allowing frames from different episodes to be connected and stitched into routes to the goal. Planning over the full graph also makes the system highly robust to execution errors: if the micro planner drifts off course, METRO-WM instantly finds a new optimal path from the current state. Our experiments show that METRO-WM achieves superior long-horizon success rates of up to 37.33 percentage points over the next best hierarchical approach while being up to $10.9\times$ faster, requiring both 13–$56\times$ less offline compute and fewer tuned hyperparameters. Additional analysis reveals that METRO-WM finds shorter paths than the offline demonstrations, outperforms an oracle relying on the query's own demonstration, and maintains robust performance under extremely sparse dataset conditions.

1 INTRODUCTION

A central goal in autonomous machine intelligence is to learn an internal model of the world that is capable of predicting how the physical environment transforms when an action is applied (LeCun, 2022). However, predicting every low-level pixel or sensory detail of future states is often computationally intractable, and in many cases, irrelevant for decision-making. One of the ways to overcome this is for the world model to learn abstract, high-level dynamics in a reward-free manner, allowing it to be used at test time to plan towards novel goals that were not seen during training.

Joint Embedding Predictive Architectures (JEPAs) (LeCun, 2022; Assran et al., 2023; Bardes et al., 2024) provide a concrete realisation of this idea by learning compact representations through prediction in latent space. JEPA-based world models (Zhou et al., 2025; Sobal et al., 2025; Maes et al., 2026) build on these representations by pairing them with an action-conditioned latent dynamics model, making planning a viable alternative to policy learning. These models enable zero-shot, goal-conditioned model-predictive control (MPC), which evaluates candidate action sequences by comparing their predicted latent outcomes with the encoded goal. However, such flat planners are reliable only over short horizons: as the number of steps increases, prediction errors accumulate and latent distances become less informative. Yet many tasks we would like to perform, such as navigating around obstacles or multi-step object manipulation, require long sequences of actions. To be able to fully realise the promise of learned world models, we therefore need planners that remain reliable beyond short, locally solvable tasks.

To extend the planning horizon, recent hierarchical JEPA frameworks (Zhang et al., 2026; Masip et al., 2026) add a macro planner that generates intermediate latent sub-goals, decomposing long-horizon tasks into segments that a micro planner can solve reliably. However, these sub-goals are

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.

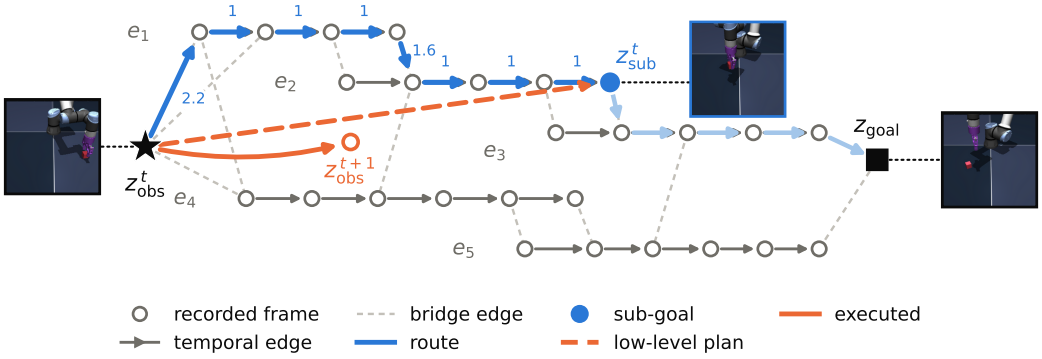

Figure 1: **One cycle of METRO-WM.** Rather than generating sub-goals, METRO-WM retrieves them from previously seen episodes. The macro planner searches the graph whose vertices represent observed frames encoded as latent states, with $e_1 \dots e_5$ in the figure indicating the specific episodes containing these frames. Directed *temporal edges* (solid grey) connect consecutive frames within each episode, while symmetric *bridge edges* (dashed grey) link visually similar frames across episodes (example endpoints in Appendix Figure 7). All edge costs are expressed in frame units: Temporal edges have unit cost, while bridge-edge costs are converted into frame-equivalent costs derived from empirical dataset quantiles. Given a goal z_{goal}, we use Dijkstra's algorithm to search backward from z_{goal} to find the lowest-cost route from every graph vertex. At each planning cycle, METRO-WM add the current observation z^t_{obs} to the graph, and identifies the shortest route (blue) to z_{goal} (which may stitch together multiple episodes). The furthest frame along this route that is within the micro planning horizon is then selected as the next sub-goal z^t_{sub}. After execution (solid orange), planning resumes from the reached state z^{t+1}_{obs} (see Appendix Figure 4).

not constrained to the small subset of latent space corresponding to physically realisable states. Caselli et al. (2026) found that unconstrained search around macro-actions generate poor quality sub-goals and proposed constraining the search to macro-actions derived from training trajectories. Extending these observations, we introduce the *realisability residual*, a decoder-free measure of sub-goal realisability that avoids the visual artefacts of decoders (Kurutach et al., 2018; Nair et al., 2018; Pertsch et al., 2020). Using this measure, we show that existing mitigations remain insufficient: state-of-the-art macro planners still routinely generate sub-goals that do not correspond to any physically realisable state.

This lack of a realisability guarantee motivates our method, METRO-WM, a hierarchical framework that replaces latent sub-goal generation with retrieval from recorded episodes[1]. See Figure 1 for an overview. Akin to navigating a transit network, METRO-WM's macro planner searches a graph of observed frames for routes to the goal, then selects vertices along the route as sub-goals for the micro planner. By retrieving sub-goals exclusively from recorded episodes, METRO-WM guarantees that each sub-goal encodes a physically realisable state.

Our contributions are threefold: *i)* The realisability residual, a decoder-free measure showing that the state-of-the-art hierarchical macro planners generate sub-goals that do not correspond to encodings of realisable states; *ii)* METRO-WM, a non-parametric macro planner that builds a graph over recorded (expert or random-action) episodes, connecting consecutive frames within each episode and visually similar frames across episodes. The planner uses shortest-path search to find routes (potentially stitching together multiple trajectories) and selects graph nodes along them as sub-goals, thereby guaranteeing their physical validity; *iii)* An evaluation on four benchmarks with two distinct micro planners LeWM (Maes et al., 2026) and PLDM (Sobal et al., 2025) where METRO-WM achieves superior long-horizon success rates by up to 37.33 pp over the next-best hierarchical approach while being up to $10.9\times$ faster and requiring $13 - 56\times$ less offline compute. METRO-WM also frequently finds paths shorter than the expert demonstrations (when provided), outperforms an oracle that uses ground-truth sub-goals, and remains robust in very sparse data regimes.

[1]We use the terms episodes and demonstrations interchangeably.

2 RELATED WORK

Hierarchical planning tackles the challenges of long-horizon control by decomposing a task into shorter segments, with sub-goals either *generated* by a trained model or *retrieved* from recorded experience. Within the domain of JEPAs and latent world models, existing hierarchical approaches are overwhelmingly generative, relying on continuous optimization and/or learned macro-policies to predict latent sub-goals (Zhang et al., 2026; Caselli et al., 2026; Masip et al., 2026). Conversely, the paradigm of retrieving sub-goals from data is well-established, but has been predominantly confined to classical reinforcement learning (RL) (Eysenbach et al., 2019; Emmons et al., 2020) and visual navigation (Savinov et al., 2018; Shah et al., 2021), where agents query episodic memory buffers and/or traverse spatial topological maps.

METRO-WM bridges this gap, bringing the physical guarantees of retrieval-based sub-goals to JEPA-based latent planning. This adaptation introduces three fundamental distinctions: *1)* we tackle the distinct problem of guiding a zero-shot latent micro planner while strictly preventing sub-goal hallucination rather than maximizing extrinsic rewards. *2)* we calibrate frame-based edge weights directly from the empirical dataset quantiles instead of relying on trained value functions or critics. *3)* our macro planner remains completely non-parametric and training-free. An extended discussion on previous works can be found in Appendix Section F.

3 PRELIMINARIES

Goal-conditioned latent planning. Let ϕ be a frozen encoder mapping an observation o to a latent state $z = \phi(o) \in \mathbb{R}^D$, and f a frozen latent dynamics model (predictor) trained alongside it, so $f(z_t, a_t)$ predicts the encoding of the next observation under action a_t. We take (ϕ, f) as given; in our experiments we use the released LeWorldModel (LeWM) (Maes et al., 2026) and PLDM (Sobal et al., 2025) checkpoints. Following convention, we chunk actions in blocks of 5 environment frames so a plan of h blocks spans $5h$ frames, where h is the latent planner's planning horizon. Given a goal observation, o_{goal}, and writing $z_{\text{goal}} = \phi(o_{\text{goal}})$ and $z_{\text{obs}} = \phi(o_t)$, a flat latent planner solves, at every replanning step,

$$a^{\star}_{1:h} \;=\; \arg\min_{a_{1:h}} \left\| \hat{z}_h - z_{\text{goal}} \right\|_2^2, \qquad \hat{z}_0 = z_{\text{obs}}, \quad \hat{z}_{t+1} = f(\hat{z}_t, a_t), \tag{1}$$

executes a prefix of $a^{\star}_{1:h}$, re-encodes, and repeats, minimizing by a sampling optimiser. Following LeWM, we adopt the cross-entropy method (Rubinstein, 1999) (CEM) in all our experiments. Eq. 1 evaluates new goals zero-shot by relying solely on a terminal cost. Consequently, it is only effective if the latent distance metric $\|\cdot - z_{\text{goal}}\|$ can successfully distinguish between states across the entire length of the planned trajectory. This strictly limits h: as h lengthens, f accumulates prediction errors and the latent distance metric saturates. Saturation serves as the fundamental bottleneck: even a perfect predictor cannot rank distant candidates, as demonstrated in Appendix Section D. We call Eq. 1 the *micro* planner π_{lo}, reliable only at short planning horizons.

Hierarchical latent planning. To enable longer horizon planning, hierarchical latent planners insert a *macro* level. A macro planner π_{hi} is given $(z_{\text{obs}}, z_{\text{goal}})$ and emits a latent sub-goal

$$z_{\text{sub}} \;=\; \pi_{\text{hi}}(z_{\text{obs}}, z_{\text{goal}}), \qquad \pi_{\text{hi}} : \mathbb{R}^D \times \mathbb{R}^D \;\to\; \mathbb{R}^D, \tag{2}$$

which replaces z_{goal} in Eq. 1 so π_{lo} only ever covers a distance it is competent over. While implementations of π_{hi} vary, ranging from learned macro actions and sub-goal policies to optimised expert actions, they all follow the structure of Eq. 2. However, we identify a core issue with this formulation: z_{sub} is produced by optimising over the entire latent space $\mathbb{R}^D$, but $\mathbb{R}^D$ is much larger than the set of valid, encodable states. If $\mathcal{S}$ represents the environment states, the set of realisable latents is $\mathcal{Z} = \{\, \phi(\text{render}(s)) : s \in \mathcal{S} \,\}$ where $\text{render}(\cdot)$ maps the underlying physical state to the visual observation. Because $\mathcal{S}$ is often low-dimensional, $\mathcal{Z}$ forms a very thin manifold within $\mathbb{R}^D$. Since nothing forces the optimization to keep z_{sub} inside $\mathcal{Z}$, the planner can output a sub-goal outside this manifold, representing a physically non-existent state.

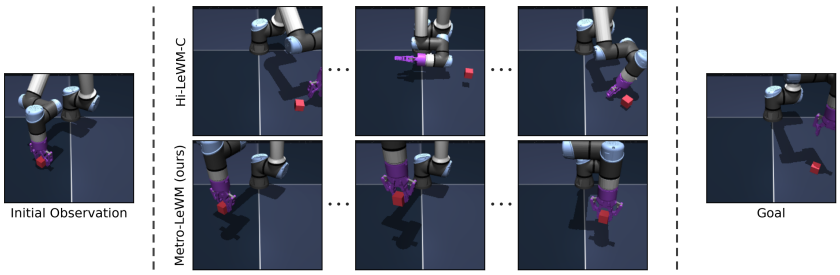

Figure 2: Decoder-free subgoal analysis. Top row: closest *realizable* states to the non-realisable sub-goals emitted by Hi-LeWM-C macro planner (Eq. 3), drawn by the environment's own renderer. Bottom row: Our METRO-WM sub-goals, which are real states retrieved from the training dataset. Micro actuation - constrained to *realisable* states - will not progress toward the final goal when pursuing sub-goals issued by Hi-LeWM-C, but will progress when using METRO-WM sub-goals.

4 ARE MACRO LATENT SUB-GOALS REALISABLE?

Previous works (Zhang et al., 2026; Caselli et al., 2026) visualise latent sub-goals using a decoder, trained on demonstrations from the dataset used to train the models, which confounds sub-goal quality with the decoder's reconstruction ability. We, therefore, introduce an evaluation protocol isolating the evaluation to the environment's renderer and frozen encoder ϕ, while remaining strictly independent of the provided demonstrations. Specifically, we define the **realisability residual** as:

$$r(z) \;=\; \min_{s \in \mathcal{S}} \big\| \phi\big(\mathrm{render}(s)\big) - z \big\|_2 \;=\; \mathrm{dist}\big(z, \mathcal{Z}\big). \tag{3}$$

where a large $r(z)$ means no physical state encodes to z. Exhaustive search over $\mathcal{S}$ is intractable, so we approximate the minimum with CEM. CEM only upper-bounds $r(z)$, so we rule out under-searching in two ways: we tune every CEM hyperparameter towards lower r, and we verify that substantially increasing the search budget leaves the median $r(z)$ unchanged (Appendix B).

We apply this to the macro-planner sub-goals of the state-of-the-art hierarchical LeWM, Hi-LeWM-C (Caselli et al., 2026), on OGB-CUBE (Park et al., 2025) and PUSHT (Chi et al., 2024). On both, CEM finds no nearby physical state: the median $r(z)$ is 0.9 on CUBE and 2.3 on PUSHT (Fig. 2; Appendix Fig. 5). The scale of $r(z)$ is not interpretable in isolation. However, a residual that stays above zero as the search budget grows indicates that no realisable state exists. This motivates our approach, which emits only sub-goals that are encodings of real states.

5 METRO-WM: SUB-GOAL SELECTION BY SHORTEST PATHS

Section 4 highlights that latent sub-goals in previous works are often unrealisable, yielding a clear design constraint: the sub-goal issued to π_{lo} must correspond to a physically plausible state. Any previously recorded frame trivially meets this requirement and the challenge then lies in selecting a *useful* state, one that is both within π_{lo}'s reach and on a route to meet the goal. To achieve this, METRO-WM replaces the prediction of Eq. 2 with a shortest-path search over prior experience.

5.1 THE GRAPH OF RECORDED FRAMES

To enable a shortest-path search over prior experience, we model the dataset as a weighted graph $G = (V, E, w)$. Assuming a corpus of M recorded trajectories $\mathcal{D} = \{\tau^{(m)}\}_{m=1}^{M}$, where $\tau^{(m)} = (o_1^{(m)}, a_1^{(m)}, \ldots, a_{L_m-1}^{(m)}, o_{L_m}^{(m)})$ has length L_m, the vertices are the encoded frames, $z_t^{(m)} = \phi(o_t^{(m)})$: $V = \{(m,t) : 1 \le m \le M,\ 1 \le t \le L_m\}$. The edges E connect these frames and are divided into two types:

Temporal edges join consecutive frames within the same episode, $E_{\mathrm{tmp}} = \{(m,t) \to (m,t{+}1)\}$, each costing one frame ($w(e) = 1$). They are transitions a demonstrator performed, making them feasible and strictly *directed* by construction as many manipulation tasks are irreversible.

Bridge edges connect a vertex to its k-nearest neighbours (kNN) *cross-episodes*, within a radius ε:

$$E_{\mathrm{brg}} = \big\{ u \leftrightarrow v \;:\; v \in \mathrm{kNN}_k(u),\ m(v) \neq m(u),\ \|z_u - z_v\|_2 \le \varepsilon \big\}. \tag{4}$$

A bridge asserts that two frames from different demonstrations represent a similar physical situation, so it is symmetric; bridges are what allow a route to leave one demonstration and continue along another. Since k-nearest-neighbour retrieval is not itself symmetric, we symmetrize the retrieved block by elementwise maximum, storing each undirected bridge exactly once.

5.2 BRIDGE EDGE WEIGHTS

A naive bridge weight based directly on latent distance violates the additive property necessary for shortest-path search. Because latent spaces saturate (Appendix Section D), accumulating latent distances fails to properly rank long, multi-hop routes. Hence, we express every bridge weight in frames, converting latent distance into frames using the demonstrations themselves. For each frame separation $d \in 1, \dots, H$, we compute $\Lambda(d) = Q_\alpha(\{\|z_{t+d}^{(m)} - z_t^{(m)}\|_2 : (m,t)\})$, which represents a low empirical quantile (α) of latent distances over that specific time gap. Restricting the grid to the waypoint budget H avoids extrapolating past π_{lo}'s reliable range and keeps Λ below saturation. Furthermore, we enforce monotonicity by taking the running maximum of $\Lambda(d)$ wherever the empirical quantile dips, ensuring that a larger latent gap is never charged fewer frames. The bridge weight, $w(u \leftrightarrow v) = \kappa(\|z_u - z_v\|_2)$, is the piecewise-linear interpolant of the knots $(\Lambda(d), d)_{d=1}^{H}$. For a latent gap δ in the interval $[\Lambda(d), \Lambda(d+1)]$, this interpolant is calculated and clipped to range:

$$\kappa(\delta) = \text{clip}\left(d + \frac{\delta - \Lambda(d)}{\Lambda(d+1) - \Lambda(d)}, 1, H\right). \tag{5}$$

A *low* quantile α makes κ deliberately conservative. By mimicking a slow demonstrator, it charges more frames for a gap, preventing the dangerous failure mode of under-pricing, which would trick the search into buying un-executable shortcuts.

Consolidating hyperparameters. The bridge radius ε, quantile α, and waypoint budget H all redundantly control the maximum sub-goal distance. To simplify tuning, we fix α to a conservative constant ($\alpha = 0.25$) and set $\varepsilon = \Lambda(H)$, leaving H as the *sole hyperparameter*. In our experiments, we tune only H alongside the micro planner's horizon h.

5.3 GRAPH SEARCH AND SUB-GOAL SELECTION

To initiate the search for each evaluation query, we insert its goal as a vertex g with latent representation z_{goal}, connecting it to its k-nearest admissible vertices via bridge edges. To ensure fair evaluation, we delete the ground-truth query episode's trajectory $m^\star$ from the graph. We then run Dijkstra's algorithm (Dijkstra, 1959) backward from g, yielding a cost-to-go $J(u)$ and an optimal successor $\text{succ}(u)$ for every vertex:

$$J(u) = \min_{\text{paths } u \rightsquigarrow g} \sum_e w(e), \qquad \text{succ}(u) = \arg\min_{v\,:\,u \to v} \big[w(u \to v) + J(v)\big]. \tag{6}$$

Crucially, because this search, run once per query, computes a global value function rather than a single trajectory, it is highly robust to execution errors. Regardless of where the micro planner (π_{lo}) lands after attempting an action, the current state can query the optimal continuation without needing to recalculate a path (See Appendix Figure 4).

Connecting the current state. Because the current observed state z_{obs} is continuously changing and is not a pre-existing vertex, it must be attached to the graph dynamically. Specifically, we evaluate the optimal path from z_{obs} to the z_{goal} by applying a one-step Bellman backup over its k-nearest neighbours in the graph. The neighbour that minimizes this backup is the optimal entry point $u^\star$:

$$u^\star = \arg\min_{u \in \text{kNN}_k(z_{\text{obs}}),\ \|z_{\text{obs}} - z_u\| \le \varepsilon} \Big[\underbrace{\kappa\big(\|z_{\text{obs}} - z_u\|_2\big)}_{\text{cost of entering}} + \underbrace{J(u)}_{\text{cost of finishing}}\Big], \qquad J(z_{\text{obs}}) = \min_u \big[\cdot\big], \tag{7}$$

where the entry cost is evaluated as a directed bridge, $w(z_{\text{obs}} \to u) = \kappa(\|z_{\text{obs}} - z_u\|_2)$. Because z_{obs} is inserted with out-edges only, this backup yields the exact shortest-path cost $J(z_{\text{obs}})$ from the current state to the goal (Proof in Appendix Section A). If no candidate lies within the ε radius, we aggressively attach to the nearest candidate regardless of distance to ensure the micro planner always has a valid path forward.

Selecting a sub-goal. Starting from the entry vertex $u^\star$, we traverse the shortest path, accumulating the total weight cost $C(\ell)$. We select the farthest vertex on this path that remains *within* the waypoint budget H:

$$u_0 = u^\star, \qquad u_{j+1} = \mathrm{succ}(u_j), \qquad C(\ell) = w(z_{\mathrm{obs}} \to u_0) + \sum_{j=0}^{\ell-1} w(u_j \to u_{j+1}),$$
$$\ell^\star = \max\{\, \ell \geq 1 \;:\; C(\ell) \leq H \,\}, \qquad z_{\mathrm{sub}} = z_{u_{\ell^\star}}. \tag{8}$$

The condition $\ell \geq 1$ guarantees that the search advances by at least one hop, even if the initial entry cost $w(z_{\mathrm{obs}} \to u_0)$ immediately overshoots the budget, ensuring that the walk always advances. In the case where there is no valid path from z_{obs} to z_{goal}, i.e. the graph is disjoint, we route towards the reachable vertex closest to the goal in latent space. Once graph progress is exhausted, we emit z_{goal} directly, allowing the system to gracefully degrade to direct goal-seeking rather than failing outright.

5.4 Summary: end-to-end pipeline

The METRO-WM pipeline operates across three distinct stages:

Offline preprocessing (once per corpus): We encode the dataset $\mathcal{D}$, estimate the empirical quantile function Λ, and set the bridge radius $\varepsilon = \Lambda(H)$. We then retrieve the k-nearest neighbours for every vertex to build the temporal edges E_{tmp} and bridge edges E_{brg}, pricing them via κ (Eq. 5).

Planning (once per episode): We encode the target goal observation, insert it into the graph as z_{goal}, and delete the query's ground-truth episode $m^\star$. We then run Dijkstra's algorithm from z_{goal} to compute the distance-to-goal J and the optimal successor succ for the entire graph (Eq. 6).

Online (every macro step): We encode the current observation z_{obs}, enter the graph via Eq. 7, walk the successor function within the waypoint budget H, and hand the selected sub-goal z_{sub} to micro planner π_{lo} (Eq. 8). Because π_{lo} is imperfect, it will inevitably land in an unpredicted state. At the next step, we simply re-encode this new state and re-enter the graph.

We show the graph traversal and sub-goal selection in Fig. 1 and re-entry in Appendix Fig. 4. We also provide the full algorithm in Appendix Section C.

6 Evaluation

6.1 Experimental Setup

Model and Benchmarks. We use LeWM (Maes et al., 2026) and PLDM (Sobal et al., 2025) as our frozen micro planners in all experiments. For METRO-WM, we plug the micro planners into METRO-WM, referred as METRO-LeWM and METRO-PLDM respectively, and use the micro planner's shipped checkpoints and CEM (Rubinstein, 1999) configurations and hyperparameters, tuning only its planning horizon h. We evaluate on the four goal-conditioned benchmarks LeWM was trained on: PUSHT (Chi et al., 2024; Florence et al., 2021), OGBench-CUBE (Park et al., 2025), TWOROOM (Zhou et al., 2025; Sobal et al., 2025), and REACHER (Tassa et al., 2018).

Metrics and Baselines. A goal is the observation d frames after the episode start, with $d \in \{25, 50, 75\}$. Each result is evaluated on the same 50 query episodes per benchmark, repeated thrice on different planner seeds. We report mean and standard deviation of the total success rate and latency at varying solver budgets (T) over all episodes. Our baselines include flat micro planners LeWM and PLDM, hierarchical LeWM (Hi-LeWM), Hi-LeWM-C, and its staged variant (Caselli et al., 2026).

Hyperparameters. For the flat micro planners, we extend their horizon h accordingly to d and evaluate it on 3 different budgets $T = 2d, 4d, 6d$. For the other baselines, we use the default shipped hyperparameter configuration from the authors for PUSHT and CUBE. As these baselines don't include checkpoints for TWOROOM and REACHER, we adopt their codebase for training them. We then do a hyperparameter sweep, following their paper, to evaluate these checkpoints and report the best performing result on 3 budgets $T = d, 2d, 4d$. Lastly, for METRO-WM, we sweep h and H. Each (h, H) cell is run at a max budget of $T = 4d$ and success rate is read off every smaller T. The best (h, H) moves with T, so we report the non-dominated set of this (h, H, T) search in success against wall clock latency.

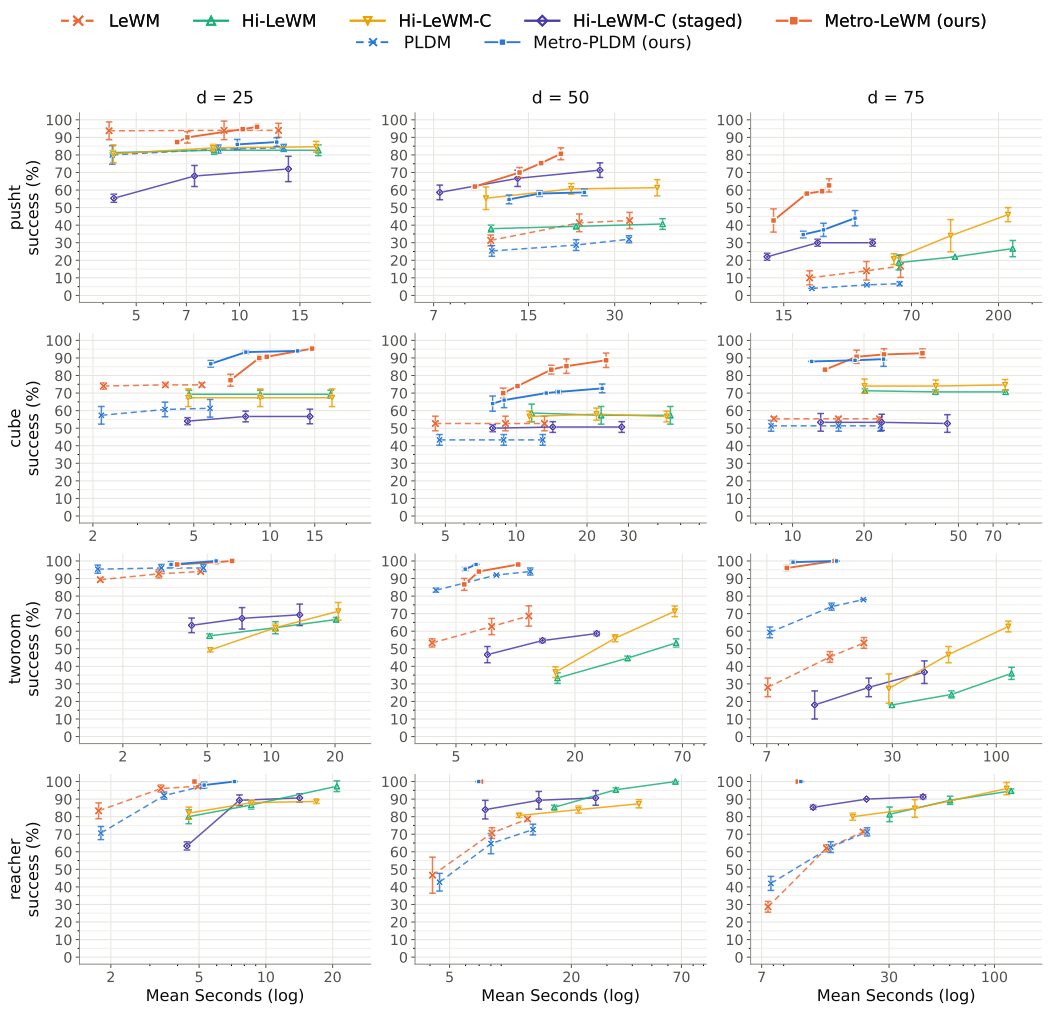

Figure 3: Performance and cost comparison between METRO-WM and baselines across varying budgets. Four benchmarks × three goal distances (planning horizons) d.

Complete details of all model, baselines, benchmarks, and hyperparameters to faciliate reproducibility can be found in Appendix Section G.

6.2 MAIN RESULT

Figure 3 plots success against online wall clock for all four benchmarks and three goal distances. All methods are evaluated with a range of planning budgets, defining a pareto front per method.

Against hierarchical baselines. At the longest horizon ($d = 75$), METRO-LeWM's best configuration beats the strongest hierarchical baseline (Hi-LeWM-C) by $+16.67$, $+18.00$, $+37.33$ and $+4.00$ points on PUSHT, CUBE, TWOROOM and REACHER respectively, while running 2.2–$10.9\times$ faster. Across all goal distances, it matches or beats every hierarchical baseline. METRO-PLDM does the same everywhere except PUSHT at $d = 50$. Only the hierarchical baselines' cheapest, less successful configurations ever run faster than METRO-WM's cheapest, most visibly at shorter horizons.

Against flat planning. Both METRO-LeWM and METRO-PLDM beat flat planning on the same base world model in all cases, and the margin grows with d. For instance, at $d = 75$, METRO-LeWM adds $+28.67$–46.67 over LeWM, while at $d = 25$, the gain is small except on CUBE. This is expected as the graph pays off when the task is long, which is where a hierarchy is supposed to help.

Table 1: One-time offline cost per benchmark. METRO-WM's "encode" is one forward pass of the frozen encoder over every recorded frame; "index" is the exact nearest-neighbour table plus the edge-cost computation. Hi-LeWM-C's training figures are wall clock at the original authors' recorded recipe; GPU-hours are wall clock × devices.

	METRO-WM preprocessing (1 GPU)			Hi-LeWM-C training (4 GPUs)		
benchmark	encode	index	**total**	grad. steps	wall	**GPU-hours**
PUSHT	17.1	6.0	**23.1** min	76,215	4.01 h	**16.0**
CUBE	34.2	4.7	**38.9** min	118,125	9.09 h	**36.4**
TWOROOM	7.6	1.4	**9.0** min	8,370	0.50 h	**2.0**
REACHER	24.4	4.7	**29.1** min	118,125	6.58 h	**26.3**

Table 2: Success (%) with and without graph re-entry, $d = 75$.

	METRO-LeWM		METRO-PLDM	
benchmark	re-entry	w/o re-entry	re-entry	w/o re-entry
PUSHT	$\mathbf{62.67 \pm 4.62}$	49.33 ± 3.06	$\mathbf{44.00 \pm 5.29}$	39.33 ± 6.43
CUBE	$\mathbf{92.67 \pm 3.06}$	84.00 ± 0.00	$\mathbf{89.33 \pm 5.03}$	88.67 ± 2.31
TWOROOM	$\mathbf{100.00 \pm 0.00}$	76.67 ± 4.16	$\mathbf{100.00 \pm 0.00}$	61.33 ± 5.77
REACHER	$\mathbf{100.00 \pm 0.00}$	99.33 ± 1.15	$\mathbf{100.00 \pm 0.00}$	98.00 ± 0.00

Overall. In the target long horizon ($d = 75$) regime, METRO-WMs are substantially faster and stronger than competitors, reaching 100% success on long horizon TWOROOM and REACHER.

Offline Costs. Table 1 shows the offline 'training' cost between METRO-WM and macro planner training. METRO-WM's graph costs 9–39 minutes on one GPU and takes no gradient steps while building a macro planner costs 13–56× that in GPU-time. Note that these training costs serve all three hierarchical baselines per benchmark.

6.3 ANALYSIS

In this section, we show further analyses by picking each benchmark's highest-success $d = 75$ result and running several ablations on them. We also experiment with fine-tuning a stronger micro planner and plugging it into METRO-WM instead of using the provided frozen micro planners and show that it leads to a boost in performance in Appendix Section E.2.

Impact of re-entry. METRO-WM re-enters the graph at the start of every macro step (Eq. 7), so wherever the micro planner lands is a legal entry point with its own optimal remaining route. The alternative is to commit by solving the route once at the beginning and providing the micro planner sub-goals along that route regardless of where it lands. Table 2 shows a drop in performance for committed routes, which is expected as the committed route has no way to re-cost the error the micro planner leaves behind. REACHER is an exception where re-entry has negligible difference as its environment is collected using a uniform random policy and, in majority of cases, the goal is often reachable in $<= 2$ macro plans, leaving little room for compounding errors.

Comparison with expert's own path. To ask whether the route is effective, we replace the graph with a privileged ORACLE: the query episode's own demonstration trajectory, $m^\star$, read at inference, aimed one hop ahead and localised to the state actually read. In other words, the ORACLE *1)* provides the micro planner with a sub-goal, retrieved from $m^\star$, that is one micro plan hop away from its current state, and *2)* finds the nearest state, in $m^\star$, to where the planner lands and retrieves the next sub-goal from there.

Table 3 shows both the success rate and the arrival ratio for both METRO-WM and ORACLE. The arrival ratio is the median of the number of frames that the approach took divided by the number of frames that the expert demonstration took to reach the goal over all successful episodes. Hence, an arrival ratio of < 1 means the route METRO-WM took is shorter than the route the demonstration took in the dataset. In particular, for CUBE, REACHER, and TWOROOM, METRO-WM reaches the goal sooner than the query's demonstration through stitching multiple demonstrations to form a shorter path. This is largely possible because the demonstrations in this benchmark tend to wander instead of

Table 3: Comparison with a privileged ORACLE, where the sub-goals come from the query's episode own trajectory at inference. Arrival ratio is the median, over the successful episodes, of frames-to-success divided by the expert demonstration's own first-hitting time; < 1 means arriving *sooner than the demonstration*.

		Success (%)		Arrival Ratio	
Approach	Benchmark	METRO-WM	ORACLE	METRO-WM	ORACLE
METRO-LeWM	PUSHT	62.67 ± 4.62	$\mathbf{64.00 \pm 2.00}$	$\mathbf{2.71}$	2.89
	CUBE	$\mathbf{92.67 \pm 3.06}$	72.00 ± 4.00	$\mathbf{0.44}$	1.32
	TWOROOM	$\mathbf{100.00 \pm 0.00}$	96.67 ± 2.31	$\mathbf{0.93}$	1.65
	REACHER	$\mathbf{100.00 \pm 0.00}$	93.33 ± 3.06	$\mathbf{0.65}$	2.38
METRO-PLDM	PUSHT	44.00 ± 5.29	$\mathbf{72.00 \pm 4.00}$	$\mathbf{3.08}$	3.26
	CUBE	$\mathbf{89.33 \pm 5.03}$	75.33 ± 1.15	$\mathbf{0.63}$	1.82
	TWOROOM	$\mathbf{100.00 \pm 0.00}$	95.33 ± 1.15	$\mathbf{0.57}$	1.10
	REACHER	$\mathbf{100.00 \pm 0.00}$	96.00 ± 0.00	$\mathbf{0.22}$	0.75

Table 4: Success (%) against corpus size, $d = 75$. 1% is 187 episodes on PUSHT and 100 elsewhere.

Approach	Corpus Size	CUBE	PUSHT	REACHER	TWOROOM
METRO-LeWM	100%	92.67 ± 3.06	62.67 ± 4.62	100.00 ± 0.00	100.00 ± 0.00
	10%	90.00 ± 5.29	46.00 ± 2.00	98.67 ± 1.15	100.00 ± 0.00
	3%	79.33 ± 1.15	38.00 ± 2.00	100.00 ± 0.00	100.00 ± 0.00
	1%	72.00 ± 2.00	31.33 ± 4.16	100.00 ± 0.00	100.00 ± 0.00
METRO-PLDM	100%	89.33 ± 5.03	44.00 ± 5.29	100.00 ± 0.00	100.00 ± 0.00
	10%	90.67 ± 1.15	29.33 ± 5.03	100.00 ± 0.00	100.00 ± 0.00
	3%	84.67 ± 1.15	26.67 ± 5.03	100.00 ± 0.00	95.33 ± 1.15
	1%	81.33 ± 1.15	10.67 ± 3.06	100.00 ± 0.00	95.33 ± 3.06

taking the shortest path straight to the goal, *e.g.* in CUBE, the robotic arm often lingers or hovers around before picking up the cube. In contrast, demonstrations in PUSHT are more likely to take the shortest path to push the T block to its goal; both METRO-WM and ORACLE took a longer path.

Dataset density. METRO-WM's success hinges on the density of the dataset, enabling it to stitch multiple episodes together to form a path to the goal. Since collecting training demonstrations is notoriously expensive and time-consuming in real-world applications, we question the effectiveness of METRO-WM under sparser conditions. To this end, we rebuild the graph from 1%, 3%, and 10% of the recorded episodes and show results of METRO-WM in Table 4. In particular, results for TWOROOM and REACHER shows that a hundred recorded episodes is sufficient to stitch a route that solves the task every time. Success rates for PUSHT and CUBE, on the other hand, drop as the number of demonstrations drop. Nonetheless, at 1%, METRO-LeWM's performance is still similar to the performance of the next best hierarchical baseline at their shipped solver budget ($T = 2d$), which was trained on the whole corpus (31.33 ± 4.16 against 34.00 ± 9.17 for PUSHT and 72.00 ± 2.00 against 74.00 ± 3.46 for CUBE).

7 CONCLUSION

We identify physically unrealisable sub-goals as a critical flaw in hierarchical latent planning and introduce a decoder-free realisability residual to quantify it. METRO-WM avoids this problem by framing long-horizon planning as shortest-path search over a dataset graph whose edge costs are measured in frames. This guarantees the physical plausibility of its sub-goals and mitigate micro planner execution errors through dynamic graph re-entry. Across different benchmarks, METRO-WM improves long-horizon success and often finds paths shorter than the offline demonstrations. We note that because METRO-WM relies on bridge edges to connect visually similar states across episodes, we anticipate it would struggle in environments with highly diverse layouts; stitching trajectories across fundamentally different scenes becomes structurally infeasible. While extending retrieval-based graph planning to accommodate such topological diversity remains an exciting direction for future work, METRO-WM ultimately establishes a highly effective, training-free foundation for robust long-horizon planning.

AI USE STATEMENT

In this work, we used generative AI tools, specifically Claude Opus 5 with Claude Code, to assist with code development, plotting figures, and interpreting empirical results. We also use Gemini Pro 3.1 and ChatGPT 5.6 Sol to improve the overall readability of the manuscript. While the initial codebase was human-generated, Claude was used extensively to debug, verify, and refactor the main backbone of the implementation. We have not used generative AI tools to formulate the core ideas, design the research methodology, or plan the experiments. We have thoroughly reviewed, tested, and verified all AI-modified code and text for accuracy and scientific integrity. We take full responsibility for the final content of this work, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

By preventing macro sub-goal state hallucination and significantly reducing computational overhead, we improve both the safety and environment footprint of autonomous planning systems. METRO-WM relies on retrieving offline demonstrations, deploying it in the real-world will require auditing the source data for biased behaviours, as well as addressing the inherent limitations of the underlying micro planner, which could still execute unpredictable low-level actions between valid sub-goals.

REPRODUCIBILITY STATEMENT

The source code for METRO-WM will be open-sourced pending internal approval. To ensure full reproducibility in the meantime, we have provided pseudocode of METRO-WM in Appendix Section C and detailed explanations of our setup in Appendix Section G. This includes comprehensive details of the datasets and underlying micro planners along with download links, complete descriptions of the network architectures, and all hyperparameters required to replicate our experiments.

REFERENCES

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 *IEEE / CVF Computer Vision and Pattern Recognition Conference (CVPR)*, 2023. doi: 10.1109/CVPR52729.2023.01499.

Seungho Baek, Taegeon Park, Jongchan Park, Seungjun Oh, and Yusung Kim. Graph-assisted stitching for offline hierarchical reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2025.

Adrien Bardes, Quentin Garrido, Jean Ponce, Xinlei Chen, Michael Rabbat, Yann LeCun, Mido Assran, and Nicolas Ballas. Revisiting feature prediction for learning visual representations from video. *Transactions on Machine Learning Research (TMLR)*, 2024.

Kevin Black, Mitsuhiko Nakamoto, Pranav Atreya, Homer Walke, Chelsea Finn, Aviral Kumar, and Sergey Levine. Zero-shot robotic manipulation with pretrained image-editing diffusion models. In *International Conference on Learning Representations (ICLR)*, 2024.

Niccolò Caselli, Francesco Massafra, Samuele Punzo, Salvatore Lo Sardo, Ippokratis Pantelidis, and Sathya Kamesh Bhethanabhotla. Mind the gap: Promises and pitfalls of hierarchical planning in LeWorldModel. *arXiv preprint arXiv:2607.12547*, 2026.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *International Journal of Robotics Research*, 2024. doi: 10.1177/02783649241273668.

Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1959. doi: 10.1007/BF01386390.

Yilun Du, Sherry Yang, Bo Dai, Hanjun Dai, Ofir Nachum, Josh Tenenbaum, Dale Schuurmans, and Pieter Abbeel. Learning universal policies via text-guided video generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Yilun Du, Sherry Yang, Pete Florence, Fei Xia, Ayzaan Wahid, Brian Ichter, Pierre Sermanet, Tianhe Yu, Pieter Abbeel, Joshua B. Tenenbaum, Leslie Pack Kaelbling, Andy Zeng, and Jonathan Tompson. Video language planning. In *International Conference on Learning Representations (ICLR)*, 2024.

Scott Emmons, Ajay Jain, Misha Laskin, Thanard Kurutach, Pieter Abbeel, and Deepak Pathak. Sparse graphical memory for robust planning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Ben Eysenbach, Russ R Salakhutdinov, and Sergey Levine. Search on the replay buffer: Bridging planning and reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Pete Florence, Corey Lynch, Andy Zeng, Oscar A Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. In *CoRL*, 2021.

Zhiao Huang, Fangchen Liu, and Hao Su. Mapping state space using landmarks for universal goal reaching. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Dinesh Jayaraman, Frederik Ebert, Alexei A. Efros, and Sergey Levine. Time-agnostic prediction: Predicting predictable video frames. In *International Conference on Learning Representations (ICLR)*, 2019.

Junsu Kim, Younggyo Seo, and Jinwoo Shin. Landmark-guided subgoal generation in hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Thanard Kurutach, Aviv Tamar, Ge Yang, Stuart Russell, and Pieter Abbeel. Learning plannable representations with causal InfoGAN. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Yann LeCun. A path towards autonomous machine intelligence. OpenReview, 2022.

Seungjae Lee, Jigang Kim, Inkyu Jang, and H. Jin Kim. DHRL: A graph-based approach for long-horizon and sparse hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. doi: 10.52202/068431-0993.

Kara Liu, Thanard Kurutach, Christine Tung, Pieter Abbeel, and Aviv Tamar. Hallucinative topological memory for zero-shot visual planning. In *International Conference on Machine Learning (ICML)*, 2020.

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

Sergi Masip, Jonathan Swinnen, Yutong Hu, Renaud Detry, and Tinne Tuytelaars. FF-JEPA: Long-horizon planning in world models with latent planners. *arXiv preprint arXiv:2606.09311*, 2026.

Saman Motamed, Laura Culp, Kevin Swersky, Priyank Jaini, and Robert Geirhos. Do generative video models understand physical principles? In *IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2026.

Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Suraj Nair and Chelsea Finn. Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation. In *International Conference on Learning Representations (ICLR)*, 2020.

Soroush Nasiriany, Vitchyr H. Pong, Steven Lin, and Sergey Levine. Planning with goal-conditioned policies. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Evgenii Opryshko, Junwei Quan, Claas Voelcker, Yilun Du, and Igor Gilitschenski. Test-time graph search for goal-conditioned reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2026.

Jyothish Pari, Nur Muhammad (Mahi) Shafiullah, Sridhar Pandian Arunachalam, and Lerrel Pinto. The surprising effectiveness of representation learning for visual imitation. In *Proceedings of Robotics: Science and Systems (RSS)*, 2022. doi: 10.15607/RSS.2022.XVIII.010.

Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. OGBench: Benchmarking offline goal-conditioned RL. In *International Conference on Learning Representations (ICLR)*, 2025.

Karl Pertsch, Oleh Rybkin, Frederik Ebert, Shenghao Zhou, Dinesh Jayaraman, Chelsea Finn, and Sergey Levine. Long-horizon visual planning with goal-conditioned hierarchical predictors. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Reuven Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology And Computing In Applied Probability*, 1(2):127–190, September 1999. ISSN 1573-7713. doi: 10.1023/a:1010091220143. URL http://dx.doi.org/10.1023/A:1010091220143.

Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric topological memory for navigation. In *International Conference on Learning Representations (ICLR)*, 2018.

Dhruv Shah, Benjamin Eysenbach, Gregory Kahn, Nicholas Rhinehart, and Sergey Levine. ViNG: Learning open-world navigation with visual goals. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021. doi: 10.1109/ICRA48506.2021.9561936.

Uladzislau Sobal, Wancong Zhang, Kyunghyun Cho, Randall Balestriero, Tim G. J. Rudner, and Yann LeCun. Learning from reward-free offline data: A case for planning with latent dynamics models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. doi: 10.52202/085713-1465.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. DeepMind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.

Ge Yang, Amy Zhang, Ari Morcos, Joelle Pineau, Pieter Abbeel, and Roberto Calandra. Plan2Vec: Unsupervised representation learning by latent plans. In *Learning for Dynamics & Control Conference (L4DC)*, 2020.

Lunjun Zhang, Ge Yang, and Bradly C Stadie. World model as a graph: Learning latent landmarks for planning. In *International Conference on Machine Learning (ICML)*, 2021.

Wancong Zhang, Basile Terver, Artem Zholus, Soham Chitnis, Harsh Sutaria, Mido Assran, Randall Balestriero, Amir Bar, Adrien Bardes, Yann LeCun, et al. Hierarchical planning with latent world models. *arXiv preprint arXiv:2604.03208*, 2026.

Gaoyue Zhou, Hengkai Pan, Yann LeCun, and Lerrel Pinto. DINO-WM: World models on pre-trained visual features enable zero-shot planning. In *International Conference on Machine Learning (ICML)*, 2025.

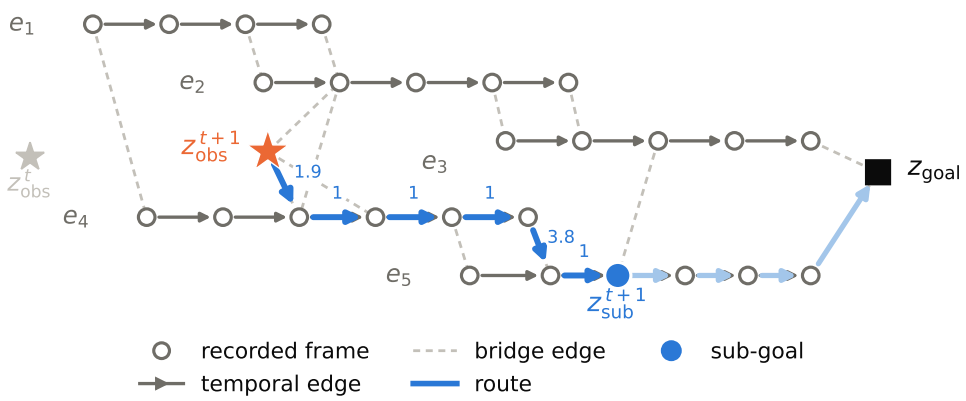

Figure 4: METRO-WM's graph re-entry. This follows Fig. 1: after the micro planner stops at z^{t+1}_{obs}, the next macro step re-enters the graph there following the optimal shortest route from z^{t+1}_{obs} to z_{goal}.

A GRAPH ENTRY

In Eq. 7, we evaluate the optimal path from z_{obs} to z_{goal} by dynamically adding z_{obs} in the graph and applying a one-step Bellman backup over its k-nearest neighbours. Figs. 1 and 4 illustrates this. Below is the proof, for completeness, that this backup yields the exact shortest-path from z_{obs} to z_{goal}.

Proposition 1 (Exactness of the entry backup)**.** *Let $G^\star$ be a weighted directed graph containing a goal vertex g, and let $J(v)$ denote the exact shortest-path cost from any vertex $v \in G^\star$ to g. If a new vertex z_{obs} is added to the graph with out-edges to a candidate set $\mathcal{N} \subseteq G^\star$ and no in-edges, then Eq. 7 computes the exact shortest-path cost from z_{obs} to g, and the original costs $J(v)$ remain unchanged.*

Proof. Because z_{obs} has no in-edges, no path originating within $G^\star$ can visit it. Therefore, the addition of z_{obs} cannot create any new shortcuts between existing vertices, leaving $J(v)$ unchanged for all $v \in G^\star$.

Furthermore, any path from z_{obs} to g must begin with an edge $z_{\text{obs}} \to u$ for some $u \in \mathcal{N}$. Since the path cannot revisit z_{obs}, the remainder of the route must lie entirely within $G^\star$. By definition, the minimal cost of this remaining route is $J(u)$. Thus, taking the minimum over all out-edges $u \in \mathcal{N}$ yields the exact shortest-path cost from z_{obs} to g. □

Because Proposition 1 holds for *any* candidate set $\mathcal{N}$, our entry rule introduces no approximation error. The one-step backup computes the true mathematical optimum for the graph it is run on; tuning the number of nearest neighbors (k) simply determines how densely z_{obs} is connected to that graph.

B REALISABILITY SEARCH OF MACRO LATENT SUB-GOALS

In Section 4, we show that latent sub-goals from existing hierarchical works are not realisable. In this Section, we explain the search in detail. The parameterisation of $\mathcal{S}$ is the recorded state itself, i.e. on PUSHT, it is 5 dimensions, including the agent's xy, block's xy, and its angle. We ran the search over the simulator's state space for each sub-goal produced by Hi-LeWM-C (Eq. 3) at two different compute budgets: 1) 3 CEM searches in parallel with 20 iterations of 384 candidates per iteration (23040 renders per sub-goal) and 2) 8 CEM searches with 30 iterations of 384 candidates per iteration (92160 renders per sub-goal). Despite increasing compute budget by fourfold, the median $r(z)$ barely drops $r(z)$: $0.92 \to 0.89$ for CUBE and $2.31 \to 2.29$ for PUSHT. The exact $r(z)$ doesn't mean much on its own since the latent space has no absolute scale. However, the fact that it plateaus above 0 tells us the optimization has failed to find a plausible solution.

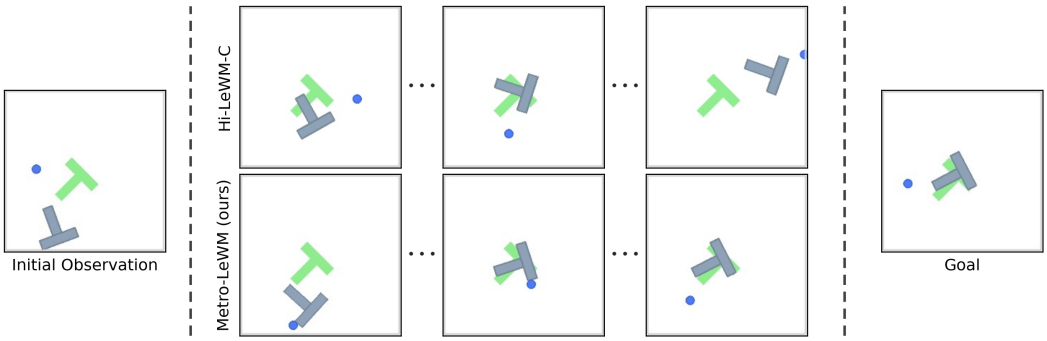

Figure 5: Top row: closest realisable Hi-LeWM-C's sub-goals from its macro planner found by Eq. 3 and drawn by the environment's own renderer. Bottom row: our method's, METRO-WM, sub-goals which are retrieved real states of other demonstrations in the dataset.

C METRO-WM'S ALGORITHM

Algorithm 1 METRO-WM: sub-goal selection by shortest paths over recorded frames. Lines 1–2 are offline and shared by every episode; lines 3–4 run once per episode; the loop runs at each macro replanning step.

Require: frozen encoder ϕ; corpus $\mathcal{D}$; micro planner π_{lo}; waypoint budget H; neighbours k; pricing quantile α

1: **offline, once:** encode $\mathcal{D}$; estimate the pricing curve Λ and set $\varepsilon \leftarrow \Lambda(H)$
2: **offline, once:** build E_{tmp} and E_{brg} (Eq. 4), symmetrizing bridges by elementwise max and weighting each by κ (Eq. 5)
3: **per episode:** $z_{\text{goal}} \leftarrow \phi(o_{\text{goal}})$; delete the held-out trajectory $m^\star$; insert g with k priced bridges
4: **per episode:** run Dijkstra's algorithm from $g \to J(u)$, $\text{succ}(u)$ at *every* u (Eq. 6)
5: **while** environment steps remain and goal not reached **do**
6: $z_{\text{obs}} \leftarrow \phi(o_t)$ {re-encode the state actually reached}
7: $\mathcal{N} \leftarrow \text{kNN}_k(z_{\text{obs}})$, excluding trajectory $m^\star$
8: **if** $J(u) = \infty$ for all admissible $u \in \mathcal{N}$ **then**
9: $z_{\text{sub}} \leftarrow$ nearest-to-goal candidate if it improves on z_{obs}, else z_{goal} {no route}
10: **else**
11: $u^\star, w(z_{\text{obs}} \to u_0), J(z_{\text{obs}}) \leftarrow$ Bellman entry backup over $\mathcal{N}$ (Eq. 7)
12: **if** $J(z_{\text{obs}}) \le H$ **then**
13: $z_{\text{sub}} \leftarrow z_{\text{goal}}$ {goal within one waypoint budget}
14: **else**
15: $z_{\text{sub}} \leftarrow z_{u_{\ell^\star}}$, walking succ from $u^\star$ on the remaining budget $H - w(z_{\text{obs}} \to u_0)$, taking the first hop unconditionally (Eq. 8)
16: **end if**
17: **end if**
18: run $\pi_{\text{lo}}(z_{\text{obs}}, z_{\text{sub}})$ {run micro planner given sub-goal}
19: **end while**

D LATENT METRIC SATURATION

Section 3 and 5 assert that the latent metric saturates and that this is a property of the representation rather than of the rollout. We provide evidence for this claim in Fig. 6, which plots the median, and IQR, latent gap between frames within the same demonstration divided by the latent gap between the same source frame to a random frame from a different demonstration over $5,000$ samples. This shows that the latent distance can only tell frames apart at small gaps, and how small depends on the benchmark and encoder. For instance, for PUSHT, between $d = 5$ and $d = 10$, y-axis moves from $0.29 \to 0.52$, indicating that a bigger gap in frames gives a bigger latent distance and the CEM planner can tell them apart. In contrast, between $d = 50$ and $d = 75$, y-axis moves from $0.98 \to 1.0$

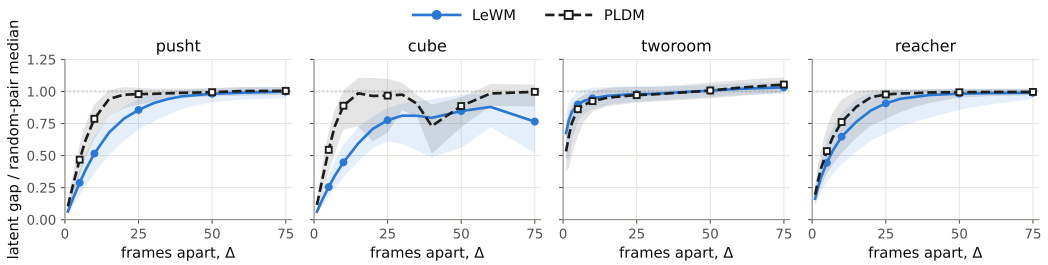

Figure 6: **Latent distance stops resolving time within a few dozen frames.** Median $\|z_t - z_{t-d}\|$ between two frames of the same demonstration d frames apart (band: interquartile range), divided by the median distance between frames of two *different*, randomly drawn demonstrations. A curve at 1 cannot tell a state d frames away from a random frame of the corpus.

- the cost can't say which is closer. This, on top of compounding prediction errors, is why the micro planner fails over longer horizons and why every edge in METRO-WM's graph is priced in frames.

E ADDITIONAL RESULTS

E.1 WHAT A BRIDGE LOOKS LIKE

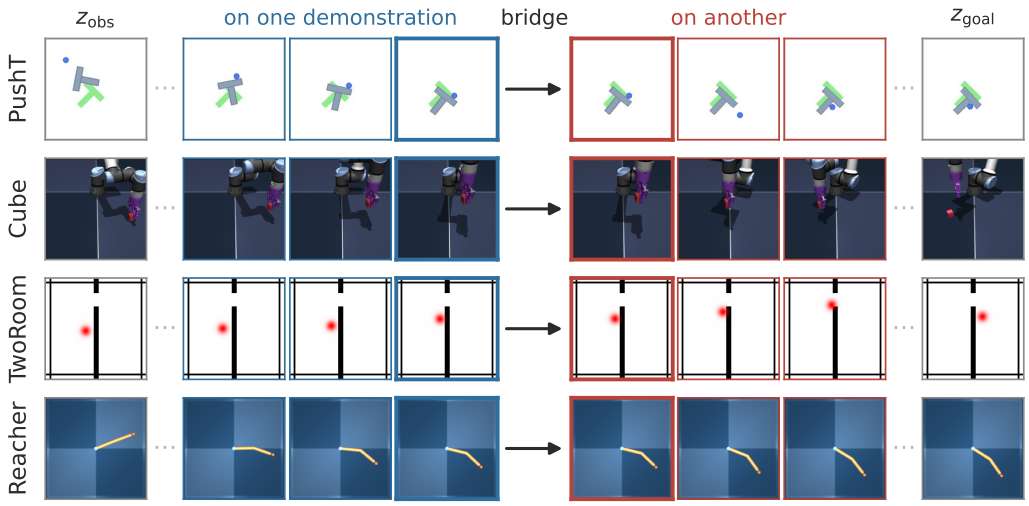

Figure 7: **A bridge, in pixels, on each benchmark.** Each row is part of one route computed by METRO-LeWM for one query of the evaluation set. The ends are the query's z_{obs} and z_{goal} and every panel between them is a recorded frame the route traverses, and $\cdots$ marks the stretches not shown. The route runs along one demonstration (blue), crosses a bridge, and continues along a different one (red). The two heavily outlined frames are that bridge's endpoints, which comes from different demonstrations in the dataset.

E.2 A STRONGER MICRO PLANNER

We hypothesise that an improvement to the micro planner converts into long-horizon success when plugged into METRO-WM. To this end, we further fine-tune the frozen LeWM, named LeWM (ft), improving its short horizon performance from 93.33 ± 5.03 to 97.33 ± 1.15 at $d = 25$. We then evaluate this model on $d = 75$ and show that despite a stronger $d = 25$ success rate, $d = 75$ flat planning leads to negligible differences in performance (Table 5). Plugging LeWM (ft) into

Table 5: We fine-tune PUSHT LeWM, dubbed LeWM (ft) to get a stronger $d = 25$ model and then evaluate it at $d = 75$. Δ is the gain over performance of the released LeWM weights, on the same 50 episodes.

Approach	Success (%)	Δ
LeWM (ft)	15.33 ± 3.06	$+1.33$
METRO-LeWM	72.00 ± 3.58	$+9.33$
ORACLE	81.33 ± 3.06	$+17.33$

METRO-WM and ORACLE, on the other hand, leads to substantial gains, indicating that macro sub-goal selection is critical to realizing micro-level gains.

F EXTENDED RELATED WORK

Hierarchical planning tackles the compounding errors of long-horizon control by decomposing a task into shorter segments, with sub-goals either *generated* by a trained model or *retrieved* from recorded experience. Hierarchical latent world models generate their sub-goals; retrieval has been developed mainly in goal-conditioned reinforcement learning (RL) and visual navigation. METRO-WM brings retrieval to latent world-model planning, so that every sub-goal is, by construction, the encoding of a real state.

Sub-goal generation in latent world models. Recent hierarchical latent world models generate sub-goals by searching at a coarser time scale (Zhang et al., 2026; Caselli et al., 2026) or by predicting the next sub-goal embedding directly (Masip et al., 2026). Nothing constrains these sub-goals to be encodings of real states. Masip et al. (2026) attributes some of its failures to predicted sub-goals that, once decoded, do not contain the actor, and Caselli et al. (2026) finds that hierarchical planning underperforms flat planning unless the search is restricted to macro-actions encoded from training demonstrations. Restricting the search does not restrict its output: Section 4 shows that this variant still emits sub-goals that no state encodes to. Every sub-goal METRO-WM emits is, by construction, the encoding of a recorded frame.

Sub-goal generation in pixel space. Outside JEPA-based world models, hierarchical visual planners generate visual sub-goals. Early approaches build on video prediction: time-agnostic prediction (Jayaraman et al., 2019) lets a predictor select which intermediate frame to output, yielding predictable "bottleneck" frames that serve as sub-goals for visual MPC; HVF (Nair & Finn, 2020) generates sub-goal images and optimises them so that each segment is solvable by visual MPC with an action-conditioned video model; and GCP (Pertsch et al., 2020) recursively predicts intermediate observations for coarse-to-fine planning, then executes the plan with a trained inverse model. In a direct-pixel ablation, LEAP (Nasiriany et al., 2019) finds image sub-goals outside the manifold of valid observations that exploit its value function. More recently, pretrained diffusion models generate sub-goals for learned low-level controllers: SuSIE (Black et al., 2024) edits the current image into a language-conditioned sub-goal for a goal-conditioned policy, while UniPi (Du et al., 2023) and VLP (Du et al., 2024) generate video plans followed by an inverse-dynamics model or a goal-conditioned policy. For these generated visual sub-goals, realisability rests on the generative prior, and visual realism does not imply physical plausibility (Motamed et al., 2026). METRO-WM instead retrieves its sub-goals from recorded frames.

Retrieval-based sub-goals and graph planning. Sub-goal retrieval is mostly studied in reinforcement learning (RL) settings. To retrieve sub-goals, graph-based methods build a graph over stored states and plan on it with shortest paths, both in visual navigation (Savinov et al., 2018; Shah et al., 2021) and in goal-conditioned reinforcement learning (RL) (Eysenbach et al., 2019; Emmons et al., 2020; Huang et al., 2019; Zhang et al., 2021; Baek et al., 2025). When nodes are recorded states, every sub-goal is realisable; methods whose nodes are cluster centroids or generated images (Zhang et al., 2021; Baek et al., 2025; Liu et al., 2020) give up this guarantee. Most works read them off a goal-conditioned value or Q-function (Eysenbach et al., 2019; Emmons et al., 2020; Huang et al., 2019; Zhang et al., 2021; Kim et al., 2021; Lee et al., 2022); others train a distance, as a

temporal-proximity classifier or contrastive score (Savinov et al., 2018; Yang et al., 2020; Liu et al., 2020) or as a learned temporal distance (Shah et al., 2021; Baek et al., 2025). VINN (Pari et al., 2022) likewise retrieves from demonstrations in a pretrained embedding space, but retrieves actions for imitation rather than sub-goals for planning. Closest to our setting, TTGS (Opryshko et al., 2026) adds no training of its own, but its edge weights come from the value function of an already-trained goal-conditioned agent.

G Detailed experimental setup

For reproducibility and completeness, we provide all details of our experiments. We will also release the code upon internal approval. All benchmarks and frozen micro planners used are directly downloaded from https://github.com/lucas-maes/le-wm.

G.1 Benchmarks

Table 6: Total number of demonstrations and frames for each benchmark in our experiments

benchmark	no. of episodes/demonstrations	no. of frames
PushT	18,685	2,261,996
Cube	10,000	1,970,000
TwoRoom	10,000	880,809
Reacher	10,000	1,970,000

PushT. A 2D task in which a circular agent pushes a T-shaped block onto a target pose. The recorded state is 7-dimensional (agent position, block position, block angle, agent velocity).

Cube (OGBench, `cube_single_expert`). A 7-DoF arm with a parallel gripper moving a cube to a target pose, simulated in MuJoCo (Todorov et al., 2012). Its realizable state space is 11-dimensional (six arm joints, the gripper driver, cube position and cube yaw).

TwoRoom. A navigation task through a doorway between two rooms. Its episodes are short (92 frames), which matters only for the hierarchical baselines' training cost: at a 90-frame training window it yields $14\times$ fewer training samples than Cube.

Reacher (dm_control). Joint-space reaching to a target configuration. The released corpus is generated by a **random policy**, not an expert, so no recorded episode reaches the goal on its own.

G.2 Micro planners

We use two base world models, LeWorldModel (LeWM) (Maes et al., 2026) and PLDM (Sobal et al., 2025), as our frozen micro planners. Both use the LeWM authors' released weights on all four benchmarks. They are based on the same architecture: the encoder is a ViT-*tiny* (12 layers, hidden width 192, 5.5M parameters) trained from scratch at 224×224 input resolution with patch size 14, Its CLS token passes through a projector MLP ($192 \to 2048 \to 192$, with batch normalization (BN) and GELU), giving a single **192-dimensional** vector (state) per frame. The predictor is an autoregressive transformer (10.8M parameters) over these states and has 6 layers, 16 self-attention heads, 192 width, 2048 feed-forward width, and a positional embedding. The action is conditioned through adaptive layer normalization.

Both planners also use the same CEM hyperparameters: 300 samples, top-k 30, variance scale 1.0, action block of $5 frames$. The number of CEM iterations is 30 on PushT and 10 on the other benchmarks. Each plan solves for h action blocks and executes them all open-loop before solving again till the budget T is reached. We sweep h only as part of Metro-WM's (h, H) search and never change any other solver setting.

G.3 HIERARCHICAL BASELINES

The three hierarchical baselines we compare against are the macro planners of Caselli et al. (2026), and all experiments are done using the authors' own code. All three share one architecture: the frozen LeWM micro planner, plus a trained *macro* predictor that maps a latent state and a 32-dimensional latent macro-action to the latent state one macro step later. A macro step is 25 frames (5 action blocks of 5 frames). The macro planner optimizes a sequence of h_h macro-actions using CEM, optimizing the last predicted latent to z_{goal}. The first predicted latent, after the first macro step, becomes the sub-goal, which the micro planner pursues by optimizing its local trajectory via its own independent CEM loop. They differ only in how the macro level proposes macro-actions and how often the macro planner replans:

- **Hi-LeWM** runs CEM directly over the latent macro-actions, within a box calibrated from 2,048 encoded expert action chunks, and warm-starts each solve from the previous solution.
- **Hi-LeWM-C** samples instead from a bank of encoded expert action sequences: each candidate is a bank exemplar plus a Gaussian residual, one candidate is left unperturbed, the residual is refitted to the elites with a floor of 10^{-3}, and the executed candidate is the best one found rather than the elite mean. We use the bank the authors specified to us, 65,536 sequences with residual scale 0.2. The bank is not leave-one-out, unlike METRO-WM's graph, so CEM may draw from the very demonstration it is evaluated on.
- **Hi-LeWM-C (staged)** uses the same bank solver but solves *once*, at $t = 0$. Its $h_h = 2$ predicted latents become fixed stage targets, each held for $\lceil d/2 \rceil$ steps.

Hi-LeWM and Hi-LeWM-C replan the macro level every 5 frames for all benchmarks, apart for PUSHT where it is replan every 2 frames at $d = 75$, following the authors' reported setting.

Hyperparameters. We use the authors' best reported solver budgets, which scale with d, in Table. 7.

Table 7: CEM hyperparameters of the hierarchical baselines, variance scale 1.0 at both levels. They come from the authors' released matrix, which covers PUSHT and CUBE; we apply the same rows on TWOROOM and REACHER.

d	macro: samples $\times$ iterations, top-k	micro: samples $\times$ iterations, top-k
25	900×20, 10	300×30, 150
50	1500×40, 10	900×30, 150
75	1200×60, 10	1200×30, 150

These baselines therefore do *not* share the micro solver settings of Section G.2; only the checkpoint is shared. Their micro planner plans 2 action blocks (10 frames), executes the first block and replans, with 30 iterations on every benchmark and more samples as d grows. For macro horizon h_h, following the authors, which sweeps $h_h \in \{1, 2, 3\}$ on PUSHT and CUBE, we sweep $h_h \in \{1, 2\}$ and report the performance that is better of the two for each benchmark and budget. The staged arm is fixed at $h_h = 2$ because it needs two stage targets.

Training on TWOROOM and REACHER. The authors release trained macro models for PUSHT and CUBE, which we use. For TWOROOM and REACHER we trained our own using the authors' model, loss and data code. Their released training entry point is not functional, so we wrote the missing training pipeline. We verified that pipeline by rebuilding the CUBE model from the authors' dumped configuration: its 425 tensors and its trainable set match the released checkpoint exactly. We also normalize pixels to ImageNet statistics, which their encoder path omits and the frozen LeWM encoder expects. Hyperparameters for the training are read off directly from their paper:

- Trained parts: only the macro planner, $12.5\,\text{M}$ of $30.5\,\text{M}$ parameters; the micro LeWM encoder, predictor, action encoder and projectors stay frozen.
- Loss: mean squared error between the predicted and actual latent of the next waypoint.
- Data: windows of 18 rows at frame-skip 5 (a 3-row history plus a span of up to 15), with 5 waypoints per window sampled at sorted random positions.

- Optimization: AdamW, learning rate 5×10^{-5}, weight decay 10^{-3}, global batch 128, bf16, gradient clipping at 1.0, and a linear warm-up over the first 1% of steps followed by cosine decay to zero, for 15 epochs.

Following PUSHT and CUBE, we evaluate the epoch-15 checkpoints. Training cost is in Table.1.

G.4 METRO-WM'S HYPERPARAMETERS

Unlike hierarchical baselines, where the macro planner CEM hyperparameters are tuned together with the micro planner CEM hyperparameters on top of macro-specific hyperparameters, *e.g.* interval in which macro planner replans, METRO-WM has only 2 hyperparameters: the waypoint budget H, in frames, and the micro planner's plan length h, in action blocks. Every other micro planner CEM hyperparameter follows what the micro planner was shipped with (Section G.2) and they stay fixed. H is quantized in steps of the action block $a = 5$, and we fixed the macro planner interval to $2ha$, one micro replan per sub-goal, throughout. We also tried tuning the number of edges k in the graph but got similar robust performance so we fixed $k = 4$ for efficiency in all experiments.

On PUSHT we sweep $h \in \{1, 3, 5\}$ against $H \in [1, 25]$, $[10, 40]$ and $[20, 40]$ respectively, totaling 471 evaluation runs on 3 different seeds $\{7, 8, 9\}$. On the other three benchmarks we sweep $h \in \{1, \dots, 5\}$ against $H \in [ha - 5,\ ha + 5]$, clamped at 1, so $H \leq 30$. As success rate is latching, for each (h, H) cell, we run once at max budget $T = 4d$ for all benchmarks and the success rate at any budget $<= T$ can be read from it. As each cell is cheap and doesn't require training, the sweep across all benchmarks took under a day on 8 NVIDIA A100 80GB GPUs for each micro planner.

To encourage reproducibility without rerunning the hyperparameter sweep, we provide the cells found, along with their budget (h, H, T), for each METRO-WM point in the frontier shown in Figure 3:

For METRO-LeWM:

- PUSHT: at d=25, $(3, 40, 30)$, $(5, 40, 50)$, $(3, 40, 60)$, $(5, 34, 100)$; at d=50, $(3, 29, 60)$, $(3, 25, 90)$, $(3, 20, 120)$, $(5, 27, 200)$; at d=75, $(3, 30, 90)$, $(3, 24, 150)$, $(3, 24, 180)$, $(5, 28, 300)$.
- CUBE: at d=25, $(1, 10, 30)$, $(1, 8, 50)$, $(1, 8, 60)$, $(1, 6, 100)$; at d=50, $(1, 10, 50)$, $(1, 10, 60)$, $(1, 8, 100)$, $(1, 7, 130)$, $(1, 7, 200)$; at d=75, $(1, 7, T)$ for $T \in \{90, 150, 200, 300\}$.
- TWOROOM: at d=25, $(1, 8, T)$ for $T \in \{30, 50, 60\}$; at d=50, $(1, 7, T)$ for $T \in \{50, 60, 100\}$; at d=75, $(1, 8, 90)$, $(1, 7, 150)$.
- REACHER: a single point per distance, $(1, 10, 30)$, $(1, 10, 50)$ and $(1, 10, 90)$ at d=25, 50, 75, each already at 100%.

For METRO-PLDM:

- PUSHT: at Δ=25, $(3, 15, T)$ for $T \in \{60, 90\}$; at Δ=50, $(3, 19, T)$ for $T \in \{90, 120, 180\}$; at Δ=75, $(3, 17, 150)$, $(3, 13, 180)$, $(3, 13, 300)$.
- CUBE: at Δ=25, $(1, 9, 30)$, $(1, 8, 50)$, $(1, 8, 100)$; at Δ=50, $(1, 9, 50)$, $(1, 9, 60)$, $(1, 6, 100)$, $(1, 7, 130)$, $(1, 5, 200)$; at Δ=75, $(1, 7, 90)$, $(1, 7, 150)$, $(1, 5, 200)$.
- TWOROOM: at Δ=25, $(1, 9, 30)$, $(1, 8, 50)$; at Δ=50, $(1, 8, T)$ for $T \in \{50, 60\}$; at Δ=75, $(1, 8, 90)$, $(1, 9, 150)$.
- REACHER: $(1, 10, 30)$ and $(1, 10, 50)$ at Δ=25, then a single point, $(1, 10, 50)$ and $(1, 10, 90)$, at Δ=50 and 75.