Emergent Neural Automaton Policies: Learning Symbolic Structure from Visuomotor Trajectories

Author Names Omitted for Anonymous Review. Paper-ID 21

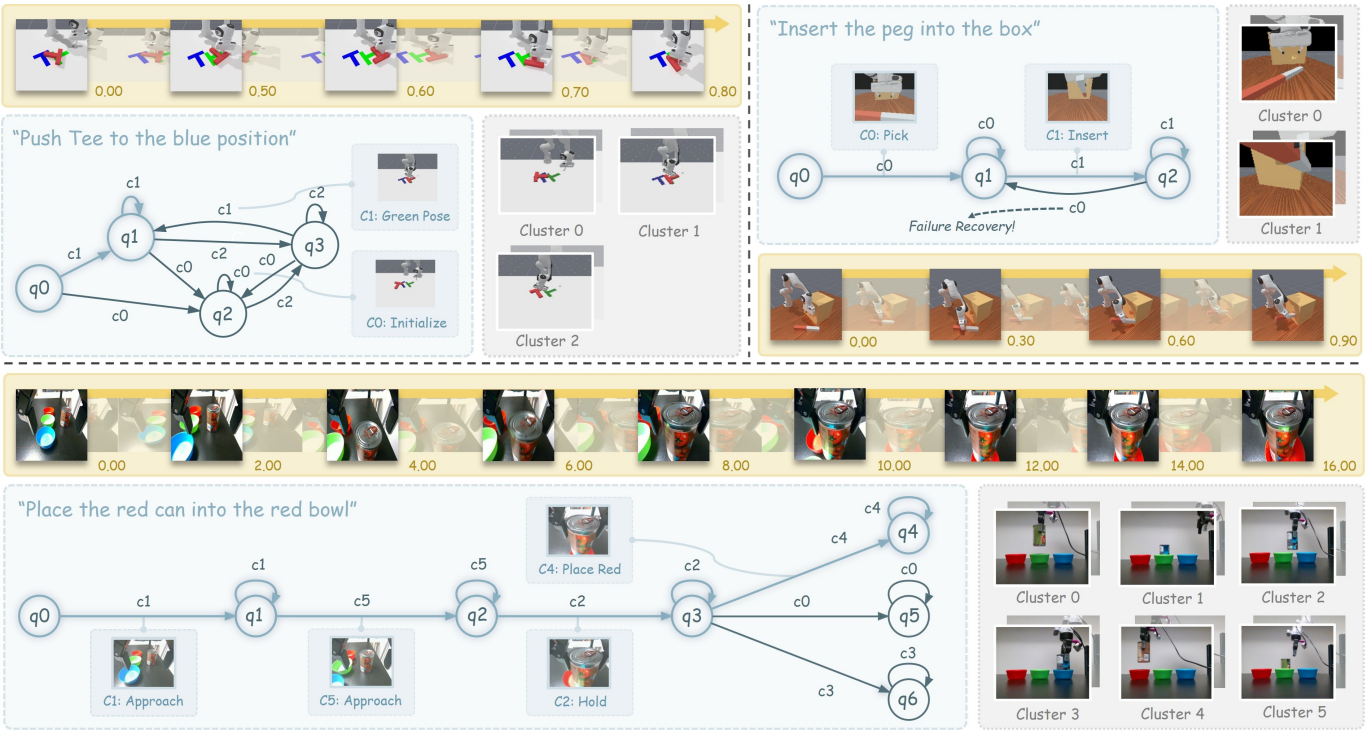

Fig. 1. **Unsupervised Discovery of Task Structures.** ENAP successfully recovers task-specific logic, including cyclic dependencies (**Top-Left**), logical branching for multi-modal goals (**Bottom**), and crucially, *autonomous failure recovery* (**Top-Right**, dashed line), where the automaton learns to transition back to a previous state ($q_2 \rightarrow q_1$) to retry insertion upon error. Semantic meanings (e.g., '*Hold*') are generated by GPT based on cluster representatives.

***Abstract*—Scaling robot learning to long-horizon tasks remains a formidable challenge. While end-to-end policies often lack the structural priors needed for effective long-term reasoning, traditional neuro-symbolic methods rely heavily on hand-crafted symbolic priors. To address the issue, we introduce ENAP (Emergent Neural Automaton Policy), a framework that allows a bi-level neuro-symbolic policy adaptively emerge from demonstrations. Specifically, we first employ adaptive clustering and an extension of the L^* algorithm to infer a Mealy state machine from visuomotor data, which serves as an interpretable high-level planner capturing latent task modes. Then, this discrete structure guides a low-level reactive residual network to learn precise continuous control via behavior cloning. By explicitly modeling the task policy with discrete transitions and continuous residuals, ENAP achieves high sample efficiency and interpretability without requiring task-specific labels. Extensive experiments on complex manipulation and long-horizon tasks demonstrate that ENAP outperforms state-of-the-art end-to-end VLA policies by up to 27% in low-data regimes, while offering a structured representation of robotic intent (Fig. 1).**

I. Introduction

Developing long-horizon execution capabilities is a critical step toward generalist robotic agents [47]. Such tasks inherently demand a synergy between high-level discrete planning and low-level continuous control, a challenge traditionally formulated within the realm of Task and Motion Planning (TAMP) [18, 57]. While recent end-to-end learning approaches, such as Vision-Language-Action (VLA) models, have demonstrated efficacy in short-horizon tasks, scaling these methods to multi-stage and long-horizon problems remains challenging [31, 20].

This limitation stems from a fundamental misalignment with the cognitive structure of intelligent decision-making. In psychology, decision-making processes in both humans and ideal robotic agents are naturally hierarchical: *System 2* governs symbolic reasoning and long-term planning, while *System 1* handles intuitive reactivity and sensorimotor control [4, 32]. This hierarchical architecture not only facilitates learning by introducing structural priors but also improves interpretability compared to unstructured baselines. In contrast, end-to-end learning attempts to approximate the non-convex optimization landscape of long-horizon tasks using a single, monolithic network, resulting in a black-box policy that is semantically opaque and highly data-intensive [20]. Conversely, while exist-

ing neuro-symbolic TAMP frameworks successfully leverage bi-level planning for both performance and interpretability enhancement, they often rely on rigid, hand-crafted symbolic priors (e.g., temporal logic predicates) [51, 49], failing to adaptively reveal the intrinsic structural manifold from data.

Therefore, this work seeks to answer a pivotal question:

How can the logical structure of System 2 naturally emerge from the perceptual flow of System 1?

Hybrid system theory offers a profound insight: complex dynamics can be decomposed into sequences of continuous local behaviors governed by discrete modes [28, 44]. Inspired by this, we argue that one appropriate policy representation for long-horizon tasks is neither monolithic network policies nor rigid symbolic policies, but a "Neural Automaton Policies". In this representation, a state machine encapsulates the global task phases (System 2), while continuous network functions model the local control within each phase (System 1). More importantly, this hybrid structure acts as a policy representation that is both kinematically feasible for robots and semantically interpretable for humans [38].

To this end, we introduce **ENAP** (**E**mergent **N**eural **A**utomaton **P**olicy), a framework that unsupervisedly reconstructs a task-level state machine planner from demonstrations (Fig. 2). **ENAP** seamlessly integrates this discrete topological planner with a downstream residual compensation network within a unified framework. This bi-level architecture unifies discrete reasoning (where humans excel) and continuous control (where robots excel), aiming for both high interpretability and strong compositionality. Specifically, we first employ adaptive clustering and an extended L^* algorithm to reconstruct the state machine from trajectory sets. Then, we combine this automaton-based planner with a residual network, learning the full policy via behavior cloning. Empirical results demonstrate that **ENAP** achieves at least an 8% absolute improvement over VLA models while using significantly 39% fewer parameters and offering superior interpretability.

In summary, our contributions are threefold:

- We propose **ENAP**, which learns a structured task abstraction and low-level reactive network in an emergent manner, offering a viable path for evolving from end-to-end reasoning to hierarchical cognitive architectures.
- We design a label-free neuro-symbolic framework that eliminates the reliance on expert priors. We incentivize structure discovery through an adaptive symbolic abstraction, improving both interpretability and compositionality.
- We demonstrate that **ENAP** consistently outperforms VLA models by at least 8% in low-data settings, and provide a rigorous Partially Observable Markov Decision Process-based theoretical justification for the framework.

II. Background

A. Problem Setup

We formulate the robotic task as a Partially Observable Markov Decision Process (POMDP) [7], defined by the tuple $\mathcal{P} = (S, \mathcal{O}, A, \mathcal{T}, \Omega, R, \gamma)$. Here, S represents the state space of the environment, $\mathcal{O}$ denotes the high-dimensional observation space (e.g., RGB-D images, proprioception), and $A \subseteq \mathbb{R}^d$ is the continuous action space. The transition dynamics are given by $\mathcal{T}(s'|s,a)$, and the observation model by $\Omega(o|s,a)$.

We operate in the Learning from Demonstration (LfD) setting. We assume access to a dataset of expert trajectories $\mathcal{D} = \{\tau_i\}_{i=1}^N$, collected from expert demonstrations or rollouts of a policy. Each trajectory is a sequence $\tau_i = (o_0, a_0, \ldots, o_{T_i}, a_{T_i})$, where the observation at time t is explicitly defined as a tuple $o_t := (I_t, p_t)$, comprising high-dimensional visual input I_t (e.g., RGB images) and proprioceptive state p_t (e.g., joint pose). The objective is to learn a structured policy that mimics the expert's behavior.

B. Preliminaries

*1) **Mealy Machine**:* Finite State Machines (FSMs) are mathematical models of computation used to design logic for sequential systems. In the context of robotic planning and control [24, 29], FSMs' nodes represent discrete operational modes and edges define transitions triggered by perceptual events. The two primary machines are *Moore machines* [16], where outputs are determined solely by the current state, and *Mealy machines*, where outputs depend on both the current state and the current input [48].

In this work, we adopt the *Probabilistic Mealy Machine* (PMM) as our structural backbone. While Moore machines are intuitive, Mealy machines offer greater reactivity and compactness [26]. Moreover, this structure mirrors the fundamental dynamics of a POMDP, where the agent's action and state evolution are jointly determined by the current latent state and observation [1]. Formally, a PMM is defined as follows:

Definition 2.1 (Probabilistic Mealy Machine): The Probabilistic Mealy Machine is $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, \lambda, q_0)$, where:

- Q is a finite set of states.
- Σ is a finite input alphabet.
- Γ is a finite output alphabet.
- $\delta(q'|q,\sigma)$, the probabilistic transition function representing the probability of moving to state q' given current state q and input σ;
- $\lambda(\gamma|q,\sigma)$, the probabilistic output function representing the probability of emitting output $\gamma \in \Gamma$ given q and σ;
- $q_0 \in Q$ is the initial state.

To bridge this automata-based definition with the POMDP formulation ($\mathcal{P}$) in our paper, we establish the following conceptual mapping: alphabet Σ corresponds to a discretization of the continuous observation space $\mathcal{O}$ (see Fig. 1 for example); nodes Q serve as discrete phases inferred from the POMDP history; output alphabet Γ represents the action space $\mathcal{A}$. Crucially, the transition function $\delta(q'|q,\sigma)$ models the high-level task progression (transitioning between phases), while the output function $\lambda(\gamma|q,\sigma)$ acts as a state-conditioned policy.

*2) **Automata learning and L^* algorithm**: Automata learning* aims to infer an FSM that explains a given set of observed sequences. Traditional algorithm is Angluin's L^* *Algorithm*

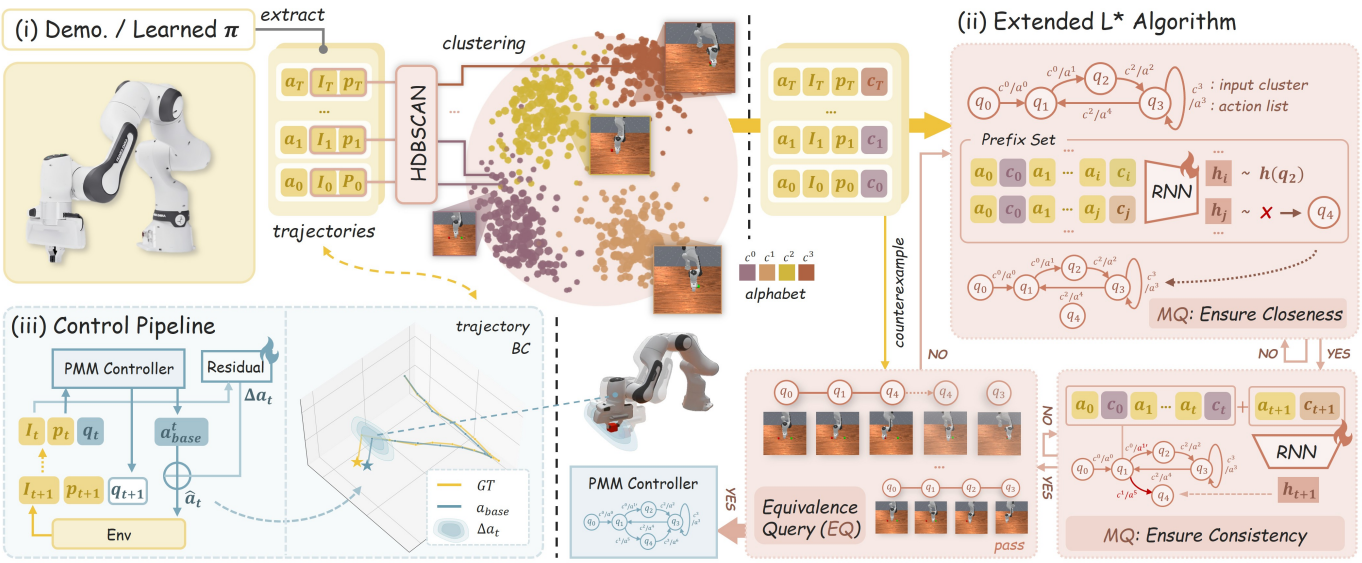

Fig. 2. **Overview of the ENAP Framework.** Our approach unifies structure discovery and hierarchical control through a three-stage pipeline: **(i) Symbol Abstraction:** Continuous trajectories from demonstrations are encoded and clustered via HDBSCAN [34] to discover a discrete alphabet Σ, mapping sensorimotor streams to symbolic sequences. **(ii) Structure Extraction via Extended L*:** We introduce a data-driven L^* algorithm that iteratively constructs a PMM by querying the dataset. The process maintains an observation table and enforces closedness (*expanding nodes*) and consistency (*refining edge*) constraints using an RNN-based history encoder. **(iii) Bi-level Control Pipeline:** The learned PMM serves as a high-level planner, providing a coarse action prior a^t_{base}. This is combined with a learned residual term $\Delta a_t \sim \pi_\psi(\cdot|q_t, o_t)$ to synthesize the final precise action $\hat{a}_t$.

TABLE I
COMPARISON OF SYMBOLIC STRUCTURE DISCOVERY APPROACHES.

Method	Continuous	Probabilistic	Label-Free
- PDDL / Rule-Based			
Bilevel Learning [6]	✓		
Neuro-Symbolic IL [10]	✓		
- Automata Extraction (NLP/Theory)			
RNN Extraction [2,3]			✓
WFA Extraction [5]		✓	✓
Transformer Extr. [4]			✓
- Robotic Automata Learning			
PDFA Learning [7]	✓	✓	
LATMOS [8]	✓		
Serialized FSM [9]	✓		
ENAP (Ours)	✓	✓	✓

[2]. Its core principle rests on the Myhill-Nerode theorem [14], which provides a rigorous definition of "node" (state) in FSMs: two history sequences are equivalent (i.e., belong to the same node) if and only if they yield identical future behaviors for all possible subsequent inputs. L^* leverages this to actively query the system to cluster history sequences into discrete states.

Specifically, L^* algorithm maintains an *Observation Table* $(\mathcal{U}, \mathcal{E}, T)$ to infer the automaton. Let $\Sigma^* := \{\sigma_1 \cdots \sigma_k \mid k \in \mathbb{N}, \sigma_i \in \Sigma\}$ denote the set of finite input sequences, and $\mathcal{U} \cdot \Sigma = \{u \cdot \sigma \mid u \in \mathcal{U}, \sigma \in \Sigma\}$ denote concatenation operation. Then, $\mathcal{U} \subset \Sigma^*$ is a set of *Prefixes*, which can be viewed as candidate history sequences leading to distinct states; $\mathcal{E} \subset \Sigma^*$ is a set of *Suffixes*, representing future test sequences used to distinguish between two prefix sequences $u_i, u_j \in \mathcal{U}$. The table entries $T : \mathcal{U} \cdot \mathcal{E} \to \Gamma^*$ record the queried system's output ("what happens if we execute history $u \in \mathcal{U}$ followed by test $e \in \mathcal{E}$") for a prefix followed by a suffix. The learning proceeds via interactions with a Minimally Adequate Teacher (MAT) [10]:

1) **Membership Query (MQ)**: the learner asks for the system output of a specific sequence $u \cdot e$, filling the table entries in T.
2) **Equivalence Query (EQ)**: The learner proposes a hypothesis automaton based on the current table. The teacher either confirms it or provides a *counter-example* trajectory that reveals an undiscovered behavior.

With queries, the algorithm can ensure two key properties of the table: *Closedness* and *Consistency*. *Closedness* ensures that for every discovered state, its transition to any next state is also covered by the known set $\mathcal{U}$. *Consistency* ensures that if two histories appear identical under current tests $\mathcal{E}$, they must remain identical after any one-step extension. If either property is violated, L^* expands $\mathcal{U}$ or $\mathcal{E}$ accordingly. We provide an illustrative example of L^* Algorithm in Appendix.

However, applying classical L^* directly to robotic learning presents a significant gap: standard L^* assumes discrete symbolic inputs and access to an omniscient oracle. Moreover, it is originally designed for deterministic automata. Therefore, we extend the $L*$ algorithm to enable learning of PMM.

*3) **Symbolic Structure Discovery**:* Symbolic structure discovery in robotics has explored a broad range of mechanisms; we provide a more comprehensive Related Works in Appendix. As summarized in Table I, existing approaches remain limited in their ability to jointly handle continuous/probabilistic robot dynamics and label-free symbolic abstraction.

III. METHODOLOGY

Method Overview. Given the demonstration dataset $\mathcal{D}$, our objective is to learn a structured policy π_{ENAP} comprising a

high-level PMM controller $\mathcal{M}$ and a low-level and compensatory residual network *Firstly*, we perform symbol abstraction to discretize the continuous observation space $\mathcal{O}$ into an alphabet Σ, and compress sequences of symbols into task phases (nodes) using an Recurrent Neural Network (RNN). *Second*, we introduce an extension of the L^* algorithm for continuous trajectory data. *Finally*, we train a residual policy network ψ via behavior cloning to ensure precise control, which takes the coarse action prior $\lambda(\cdot|q,\sigma)$ from the learned PMM and predicts a continuous action. (Fig. 2).

In what follows, we try to answer two questions: (i) *How to unsupervisedly construct an interpretable state machine from noisy demonstrations? (Sections III-A and III-B)* and (ii) *How to leverage such learned structure to guide the learning of hierarchical control policy? (Sections III-C and III-D)*

A. Adaptive Symbol Abstraction

In the **ENAP** framework, we follow Definition 2.1 and instantiate the PMM. Specifically, the node set Q represents the discrete latent task modes (e.g., *Reach*, *Grasp*, *Align*), while the edges encode the transitions between these nodes.

Given the continuous trajectory data, the first step of **ENAP** is to map observations $o_t := (I_t, p_t)$ to a semantic feature vector $z_t = \phi_\theta(o_t) \in \mathbb{R}^z$ via an encoder. Following this, we employ HDBSCAN, an adaptive clustering algorithm, to unsupervisedly discover the alphabet Σ within the feature space. Each cluster leads to one symbol. This step augments the trajectory data at each time step from (o_t, a_t) to a tuple (o_t, a_t, c_t), where $c_t = \text{HDBSCAN}(z_t) \in \Sigma$ is the assigned cluster label. The choice of encoder depends on whether the trajectory data are obtained from demonstrations or from rollouts of the learned policy.

1) **Scenario 1 (Policy Enhancement)**: The trajectory data are obtained from policy rollouts. In this setting, our objective is to extract a PMM to enhance interpretability or to construct a structured policy with improved performance. The encoder learned within the policy is directly reused.
2) **Scenario 2 (Policy Discovery)**: The trajectory data are obtained from demonstrations. Our objective is to learn a policy from scratch using the provided demonstrations (e.g., teleoperation data). In this setting, we fine-tune a raw visual encoder (e.g., DINO [41]) using our proposed algorithm (see Algorithm 2).

Given the clustered symbols, standard L^* algorithms rely on exact matching of sequences to distinguish nodes. In robotics, a sequence refers to a trajectory of historical actions and symbols $(a_{0:t}, c_{0:t})$. To handle the variable-length nature of robotic trajectories where different temporal spans may map to the same task phase, we train an RNN to map the history $(a_{0:t}, c_{0:t})$ to a fixed-size embedding $h_t \in \mathcal{H}$ (the number of h equals the number of trajectories multiplied by their respective time horizons). Specifically, we attach auxiliary prediction heads to the RNN during training and discard them afterward in order to optimize a multi-objective loss:

$$\mathcal{L}_{\text{RNN}} := \mathcal{L}_{\text{pred}} + \lambda\mathcal{L}_{\text{contrast}} = \mathcal{L}_{\text{act}} + \mathcal{L}_{\text{state}} + \lambda\mathcal{L}_{\text{contrast}} \quad (1)$$

where the loss terms $\mathcal{L}_{\text{act}}$ (MSE) and $\mathcal{L}_{\text{state}}$ (cross-entropy) are employed to predict the subsequent action a_{t+1} and symbolic state c_{t+1}, respectively. Phase-aware contrastive loss $\mathcal{L}_{\text{contrast}}$ to separate latent representations across different phases: it minimizes the cosine distance between h_t and h_{t+1} during periods of constant cluster assignment (self-loops, $c_t = c_{t+1}$) while maximizing it during phase transitions ($c_t \neq c_{t+1}$).

This architectural choice of RNN is theoretically motivated: the trained RNN induces clustering of continuous hidden states around saturated centers, in which each component is driven toward ± 1 under tanh activations. As a result, cosine similarity becomes a reliable metric for distinguishing automaton states (see Appendix for the formal proposition).

B. Structure Extraction via Extended L^* Algorithm

Leveraging the abstracted alphabet Σ and the phase embeddings $\mathcal{H}$, we propose an extension of the L^* algorithm. Due to the absence of an oracle, our algorithm queries the static demonstration database $\mathcal{D}$ to infer system behavior. We formulate this as an iterative process, driven by two redefined query mechanisms: a *Self-Supervised Membership Query* (MQ) that is used to construct a hypothetical state machine, and a *Non-deterministic Equivalence Query* (EQ) that validates the hypothesis (Algorithm 1).

*1) **Self-Supervised MQ** (**Phase 1**):* In our framework, the prefix set $\mathcal{U}$ serves as the collection of representative history embeddings, where each $u \in \mathcal{U}$ is a centroid summarizing a cluster of similar trajectory embeddings h_t that correspond to a unique latent task node q. Two embeddings h_t^i and $h_{t'}^j$ from i-th and j-th trajectories at different time steps may correspond to the same $u \in \mathcal{U}$, that is, task phase. The core challenge is to identify the PMM from the non-Oracle dataset $\mathcal{D}$. To this end, we define the Generalized MQ function $\text{MQ}_{\mathcal{D}}(u)$, which retrieves the immediate action a_t and next-step embeddings h_{t+1} from all trajectory segments in $\mathcal{D}$ that are τ_{sim}-cosine similar to the representative history u, thereby obtaining transitions for the candidate state.

$$\begin{aligned}\text{MQ}_{\mathcal{D}}(u) = \{(a_t^i, h_{t+1}^i) \mid \\ \exists\ t \text{ s.t. } \cos\big(h(\tau_i[:t]), u\big) \geq \tau_{\text{sim}}\}\end{aligned} \quad (2)$$

We initialize $\mathcal{U}$ using the initial state of the first trajectory at $t = 0$, that is, $\mathcal{U} = \{h_0^1\}$ and use $\text{MQ}_{\mathcal{D}}(u)$ to expand $\mathcal{U}$. The growth of $\mathcal{U}$ continues until the hypothesis PMM satisfies both *closedness* (every reachable next-step embedding is similar to some $u \in \mathcal{U}$), at which point the construction early-stops. Specifically, for every prefix u in current $\mathcal{U}$, we query $\text{MQ}_{\mathcal{D}}(u)$ to obtain all reachable next-step embeddings $\{h_{\text{next}}\}$. If a reachable h_{next} satisfies $\max_{u' \in \mathcal{U}} \cos(h_{\text{next}}, u') < \tau_{\text{sim}}$, we add the h_{next} into $\mathcal{U}$ as a new centroid, expanding the automaton's nodes until closedness is satisfied. Once $\mathcal{U}$ is closed, each $u \in \mathcal{U}$ corresponds to a node q. To this end, we derive Q.

Given constructed Q, edge from node q to q' under input symbol $c \in \Sigma$ will store two critical pieces of information: (1) An action prior distribution $\lambda(\cdot|q,c)$, represented as an empirical set of action vectors; and (2) A next-input set $\text{NIS}(q) \subset \Sigma$, which records all the valid input symbols observed at q. We

Algorithm 1 Extended L^* Algorithm

Input: Demonstration $\mathcal{D}$, RNN h_κ, Thresholds $\tau_{\text{sim}}, \epsilon_{\text{err}}$
Output: Probabilistic Mealy Machine $\mathcal{M}_{\text{final}}$
Construct Database $\mathcal{H}$.
Initialize prefix set $\mathcal{U} \leftarrow \{h_0^1\}$.
while Hypothesis is not *closed* or *consistent* **do**
  ***Phase 1*: Self-Supervised MQ**
  repeat
    Closed $\leftarrow$ True
    // Check if all transitions land in known states
    Construct transition destinations $\{h_{\text{next}}\}$ reachable from $\mathcal{U}$ via database $\mathcal{H}$.
    if $\max_{u' \in \mathcal{U}} \text{sim}(h_{\text{next}}, \text{Rep}(u')) < \tau_{\text{sim}}$ **then**
      $\mathcal{U} \leftarrow \mathcal{U} \cup \{h_{\text{next}}\}$; *Closed* $\leftarrow$ False
    end if
  until *Closed* is True
  Construct hypothesis $\mathcal{M}$ from $\mathcal{H}$ supported by $\mathcal{U}$.
  ***Phase 2*: Non-deterministic EQ**
  for all test trajectory $\tau \in \mathcal{D}_{\text{test}}$ **do**
    Track all paths in $\mathcal{M}$ consistent with τ's symbols.
    // Validity requires topology and action match
    if no valid path survives at step t **then**
      $\mathcal{U} \leftarrow \mathcal{U} \cup \{h(\tau_{:j}), \forall j \leq t\}$; **break** *// Add Counterexample*
    end if
  end for
end while
***Phase 3*: Stable Phase Pruning**
$\mathcal{M}_{\text{final}} \leftarrow$ **Prune**$(\mathcal{M})$
return $\mathcal{M}_{\text{final}}$

Algorithm 2 Iterative Residual Learning

Input: Dataset $\mathcal{D}$, Initial Encoder $\phi_{\theta^{(0)}}$, Iterations K
Output: Trained Policy $\pi_{\text{ENAP}} = (\mathcal{M}, \pi_\psi, \phi_\theta)$
for $k = 0$ **to** $K - 1$ **do**
  E-Step: Structure Extraction
  Extract features $\phi_{\theta^{(k)}}(o_t)$ for all $\tau \in \mathcal{D}$.
  Discretize features to $\Sigma^{(k)}$ via HDBSCAN.
  Train RNN on $(\Sigma^{(k)}, \mathcal{D})$ to get h_κ^k.
  $\mathcal{M}^{(k)} \leftarrow$ **Extended L***$(\mathcal{D}, h_\kappa^k, \dots)$ *// See Algorithm 1*
  M-Step: Policy Optimization
  Freeze PMM $\mathcal{M}^{(k)}$, initialize Residual Net $\pi_{\psi^{(k)}}$.
  repeat
    Sample batch (o_t, a_t, q_t) from $\mathcal{D}$.
    Compute loss $\mathcal{J} \leftarrow$ Eq. (5).
    Update $\theta^{(k)}, \psi^{(k)}$ via Gradient Descent.
  until Convergence
end for
return $(\mathcal{M}^{(K)}, \pi_{\psi^{(K)}}, \phi_{\theta^{(K)}})$

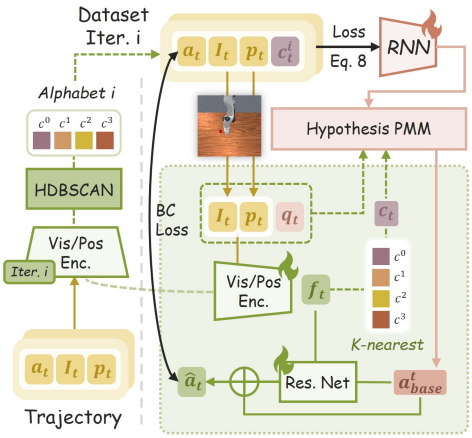

Fig. 3. **Iterative Structure-Policy Co-Evolution.** The training process alternates between dataset evolution (**Left**) and residual Learning (**Right**). A residual network refines the action prior from PMM and generate updated encoder for subsequent iterations.

construct transitions by mapping the retrieved data back to discrete states to ensure *consistency* (no invalid transitions exist). Specifically, for each state q and input symbol σ, we identify the destination state q' by finding the nearest representative $u' \in \mathcal{U}$ for each h_{next} from the query result. The transition function $\delta(\cdot|q, \sigma)$ is estimated from the frequency of transitions to q', and the action output $\lambda(\cdot|q, \sigma)$ is defined as the empirical set of actions a_t collected from all trajectory segments where the transition $q \xrightarrow{\sigma} q'$ occurred from t to $t+1$.

*2) **Non-deterministic EQ** (**Phase 2**):* For probabilistic robotic tasks, we relax the deterministic verification to a non-deterministic path search. Given a test trajectory $\tau = (o_0, a_0, \dots)$, we track all possible transition sequences in the PMM induced by its cluster-index sequence. If there exists at least one valid transition sequence (i.e., at every time step the expert action a_t lies within the action list of the edge), then the EQ is passed. Otherwise, the shortest prefix of the trajectory τ that causes the failure is returned as a counterexample and added to $\mathcal{U}$, forcing the learner to split states or introduce new transitions to resolve the ambiguity.

*3) **Stable Phase Pruning** (**Phase 3**):* To recover a compact structure from the initial temporal expansion, we merge a destination node q' into its source node q when the transition from q to q' is triggered by c and q contains a self-loop on c, which mirrors a stable phase.

C. PMM as a Reactive Controller

After obtaining a PMM from data, the inference logic of this controller operates through a deterministic predict-act-update cycle. At time step t, with current state q_t and grounded input c_t, the PMM first generates a coarse action prior a^t_{base}. Then, given the global guidance a^t_{base}, the fine-grained control is realized by the residual policy π_ψ and the encoder ϕ_θ.

$$a^t_{\text{base}} = \mathbb{E}[\lambda(\cdot|q_t, c_t)] \tag{3a}$$

$$c_t = \text{HDBSCAN}(\phi_\theta(o_t)), \tag{3b}$$

$$\hat{a}_t = \underbrace{a^t_{\text{base}}}_{\text{Fixed}} + \pi_\psi(q_t, \phi_\theta(o_t), a^t_{\text{base}}). \tag{3c}$$

Upon execution, the controller updates its internal state to q_{t+1}. In our formulation, the transition may remain ambiguous due to the probabilistic λ function. To alleviate, we exploit the

observed next symbol c_{t+1}. Specifically, after executing the action, the controller observes c_{t+1} and selects the successor state whose edge signature contains this observation. Formally, the next state is determined as follows, where $\mathbb{I}_{(\cdot)}$ prioritizes transitions whose next-input includes c_{t+1}, and $P(q' \mid q_t, c_t)$ is used to select the most probable transition when multiple nodes q' are reachable from q under input c_{t+1}:

$$q_{t+1} = \arg\max_{q' \in \delta(\cdot|q,o)} \Big[\mathbb{I}_{c_{t+1} \in \mathrm{NIS}(q')} + \epsilon \cdot P(q' \mid q_t, c_t) \Big], \tag{4}$$

D. Iterative Residual Learning and Refinement

To identify an effective feature space that maximizes performance, we propose an iterative Expectation–Maximization (EM) training paradigm (see Fig. 3 and Algorithm 2) to jointly train the encoder and the residual network. Instead of a static one-shot pipeline, this process alternates between refining the structural hypothesis (E-step) and optimizing the control policy (M-step), driven by the following joint objective:

$$\mathcal{J}(\theta, \psi) = \mathbb{E}_{\tau \sim \mathcal{D}} \Big[\big\| a_t - \hat{a}_t \big\|^2 + \lambda_{\text{reg}} \, \mathcal{L}_{\text{center}} \Big]. \tag{5}$$

where $\mathcal{L}_{\text{center}}$ is a regularization term pulling features towards cluster centers to maintain separability.

*1) **E-Step (Structure Extraction)**:* Given the current encoder parameters $\theta^{(k)}$, we freeze the feature space and perform the structure discovery pipeline. This yields updated cluster assignments $\Sigma^{(k+1)}$ and a refined PMM $\mathcal{M}^{(k+1)}$. Crucially, this step re-segments the task based on the feature space learned from the previous control iteration.

*2) **M-Step (Policy Optimization)**:* Fixing $\mathcal{M}^{(k+1)}$, we optimize the residual policy π_ψ and fine-tune the encoder ϕ_θ via Behavior Cloning. The gradient descent update is:

$$\theta^{(k+1)}, \psi^{(k+1)} \leftarrow \text{Optimizer} \Big(\nabla_{\theta,\psi} \mathcal{J}(\theta^{(k)}, \psi^{(k)}) \Big) \tag{6}$$

This EM-style training procedure ensures that the encoder learns a task-relevant manifold where "states" are both structurally consistent (for PMM) and control-effective (for residuals), avoiding trivial solutions such as collapsing to a single node or overfitting to every time step.

IV. Experiments

In this section, we present extensive experimental results to address the following questions:

- **Q1**: Can **ENAP** achieve improved control efficacy with extracted System 2?
- **Q2**: Can **ENAP** extract semantically meaningful structures from unlabeled demonstrations?
- **Q3**: Can **ENAP** exhibit the unique advantages of state machines over other symbolic structures?

A. Experiment Setup

We evaluate across three domains, as demonstrated in Fig. 4; detailed specifications are in the Appendix D. In the sequel, we denote our method as **ENAP** $(\cdot)$, where $\cdot$ specifies the policy from which the state machine is extracted. We use **ENAP*** (DINO) to indicate we train DINO from scratch using Algorithm 2. All tasks are evaluated with a success rate metric.

TABLE II
Comparison on Complex Manipulation Tasks.

Method	Param (M)	DualStack Cube (%)	Peg Insert (%)
- Oracle			
Oracle	2.98	98.3 ±0.4	86.7 ±0.8
- Traditional Imitation Learning			
Transformer [52]	63.81	38.7 ±6.0	51.8 ±5.5
GMM [35]	46.11	73.6 ±2.3	53.1 ±2.6
- Generative & VLA Policies			
Diffusion Policy [8]	114.39	41.2 ±7.2	31.1 ±6.8
OpenVLA [20]	7652.10	69.8 ±2.0	42.3 ±2.8
π_0 [6]	3288.52	73.4 ±1.2	51.6 ±1.4
- State-Machine-Based			
ENAP (Oracle)	**2.66**	98.8 ±0.3	85.6 ±0.6
ENAP* (DINO)	22.94	**76.0** ±2.0	**63.2** ±2.4

*1) **Complex Manipulation**:* `PegInsert`: focus on precision insertion; `DualStackCube`: stack blocks (green, red) in variable orders; `MultiGoalPushT`: push a Tee to the either of diverse goals. In the settings, we compare with classic behavior cloning models [35, 52] and recent large-scale / generative policies [8, 20, 6]. Moreover, to approximate an ideal extracted state machine, we obtain an *Oracle* policy, which reliably captures the task structure, using reward-shaping training or motion planning. All models are trained on identical demonstration sets on Maniskill [40] simulator.

*2) **Long-Horizon TAMP**:* CALVIN [37] is a widely used benchmark for long-horizon task and motion planning, requiring the robot to complete multi-stage manipulation sequences. We evaluate on two tasks with up to five stages. `Sequential` (`Seq.`): rotate block → push block → move slider → toggle switch → light bulb, where skills follow a fixed linear order. `Hierarchical` (`Hier.`): open drawer → lift block → place in drawer → lift second block → stack, where later actions depend on intermediate subgoals (e.g., robot must open the drawer before placing the cube into it). In the settings, we evaluate against multiple VLA baselines [36, 55, 45, 46], with their best-reported checkpoints.

*3) **Real-World Manipulation**:* `StackLego`: Precise Lego block assembly; `MultiPickPlace`: color-matched object sorting; `Hanger`: hang the hanger on a rack. In the settings, we benchmark against the fine-tuned $\pi_{0.5}$ model [19].

B. Control Efficacy & Efficiency (Q1)

The quantitative results (Table II) demonstrate **ENAP**'s ability to efficiently distill expert behavior. When using Oracle's encoder (**ENAP** (Oracle)), our method fully achieves comparable success rate while reducing the parameter count. This confirms that the extracted PMM serves as a compact, structured representation that captures the task's essence without the overhead of massive monolithic networks.

Crucially, even without access to a task-specific encoder, our framework remains effective (Table II, Table III). By employing our proposed Algorithm 2 with the DINOv2 backbone, we achieve competitive success rates. This highlights **ENAP**'s potential as a general-purpose algorithm capable of extracting effective policies without reliance on expert priors.

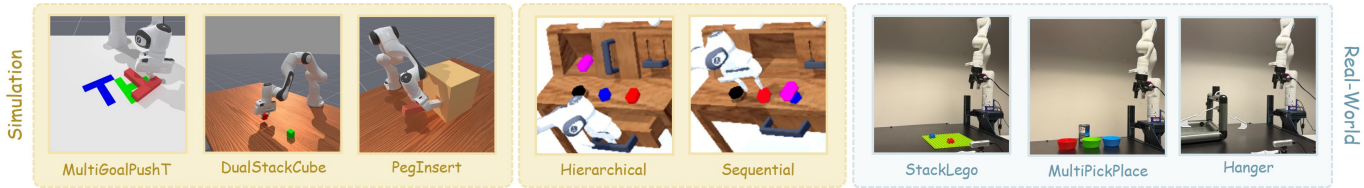

Fig. 4. **Experiments in Both Simulation and Real-World.** (**Left**) Complex manipulation; (**Middle**) TAMP; (**Right**) Real-world scenarios.

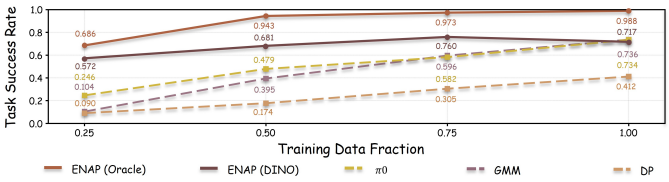

Fig. 5. **Data-scaling comparison on** `DualStackCube`. **ENAP** remains robust in low-data regimes, while other methods degrade significantly.

TABLE III
REAL-WORLD EVALUATION (17 TRIALS PER TASK, % SUCCESS RATE).

Method	**Param** (M)	**Speed** (ms)	Stack Lego	Pick Place	Hanger Task
$\pi_{0.5}$ [19]	3403	6841	58.82	76.47	64.71
ENAP* (DINO)	23	281	**88.24**	**94.12**	**94.12**

To rule out the possibility that our advantage stems merely from model size suitability, we evaluate performance across varying data fractions on `DualStackCube`. As shown in Fig. 5, **ENAP** maintains high efficacy even in extremely low-data regimes (25% data). This efficiency validates that the extracted PMM captures the true underlying task topology, acting as a strong inductive bias that generalizes beyond the training distribution.

For sequential reasoning tasks (Table IV; in the table, 3/5 and 5/5 denote completing the first 3 or 5 five subtasks, respectively.), we evaluate **ENAP** by distilling structure directly from the latent space of a pre-trained VLA. It demonstrates that our unsupervised structure discovery is applicable even to the token-level representations, thus offering a general enhancement to the interpretability and performance of existing black-box policies.

C. Emergent Structure Analysis (Q2)

For qualitative analysis, we refine the feature space by re-clustering with K-Means, using the cluster count inferred by HDBSCAN, to eliminate noise artifacts.

To validate the interpretability of our learned representation, we visualize the execution of a simplified real-world task: stacking Lego blocks with blue brick at two starting locations (Fig. 6). The extracted PMM autonomously discovers a decision point at node q_2 (the "*Grasp*" phase). During inference, the specific path selection is implicitly guided by the residual network: given the goal-conditioned visual input, the network predicts a precise action that leads to a next-step observation consistent with only one of the valid transition signatures.

To move beyond qualitative inspection, we conduct a rigorous structural evaluation on the `PegInsert` task, comparing policies derived from Oracle, DINO, and GMM encoders

TABLE IV
COMPARISON ON LONG-HORIZON TAMP TASKS.

Method/Param (M)	Seq. (%) 3/5	Seq. (%) 5/5	Hier. (%) 3/5	Hier. (%) 5/5
HULC (47) [36]	3.0 ±0.3	3.0 ±0.2	87.0 ±0.6	2.0 ±0.2
LCD (68) [55]	11.0 ±0.5	9.0 ±0.4	57.2 ±0.7	5.0 ±0.3
MDT (440) [45]	78.4 ±0.9	53.7 ±1.1	93.8 ±0.8	21.9 ±0.6
FLOWER (947) [46]	91.0 ±0.6	90.6 ±0.5	90.8 ±0.7	15.9 ±0.4
ENAP (FLOWER) (571)	**97.0** ±0.4	**96.8** ±0.3	**95.5** ±0.5	**28.2** ±0.6

TABLE V
MULTIGOALPUSHT PERFORMANCE COMPARISON.

Metric (%)	DP [8]	GMM [35]	π_0 [6]	**ENAP** (Oracle)
Either	0.41 ±0.05	0.50 ±0.05	0.25 ±0.02	**0.94** ±0.03
Nearest	0.38 ±0.07	0.38 ±0.03	0.13 ±0.04	**0.84** ±0.02

(Fig. 7). We introduce six metrics: (i) **Success Rate (SR↑)**; (ii) **Success Rate weighted by Node Count (SRN↑):** $SR/|Q|$, the efficiency of the state representation; (iii) **Action Prior Fidelity (APF):** The MSE between the PMM's coarse prior a_{base} and the ground truth (extremely low values may indicate overfitting to specific trajectories); (iv) **Loop-Transition Ratio (LVR):** The ratio of self-loops to total edges; (v) **Cluster Semantic Separability (CSS↑):** The ratio of intra-cluster to inter-cluster visual similarity (via CLIP). Higher values indicate semantically distinct task phases; (vi) **Action Separation Distance (ASD↑):** The average distance between mean actions of different edges. Higher values confirm that the automaton effectively partitions the control space into distinct primitives.

The radar chart reveals an insight: the best-performing model does not simply maximize or minimize all structural metrics. Firstly, **ENAP** (Oracle)'s **LVR** and **CSS** strike a balance between phase stability and dynamic responsiveness, indicating an optimal task structure is a resilient abstraction that partitions the control space distinctly for the residual policy to handle. The high prior uncertainty (**APF**) suggests that the Oracle-derived PMM avoids overfitting to specific trajectory noise, learning a generalized "coarse intent".

D. State-Machine-Enabled Property (Q3)

*1) **Recovery via Structural Loops**:* A unique advantage of the PMM controller is to perform autonomous failure recovery through structural loops. In the `StackLego` task, where precise alignment is challenging without force feedback, we observe that the agent frequently exploits the self-loop on the "Align" node. If the visual feedback does not match the "Insert" signature, the controller repeatedly re-attempts the adjustment. This behavior emerges naturally from the learned topology, enhancing robustness against uncertainties.

*2) **Compositionality**:* We further investigate **ENAP**'s compositional logic using the `MultiGoalPushT`, where the agent must push a T-block to one of two potential targets. The

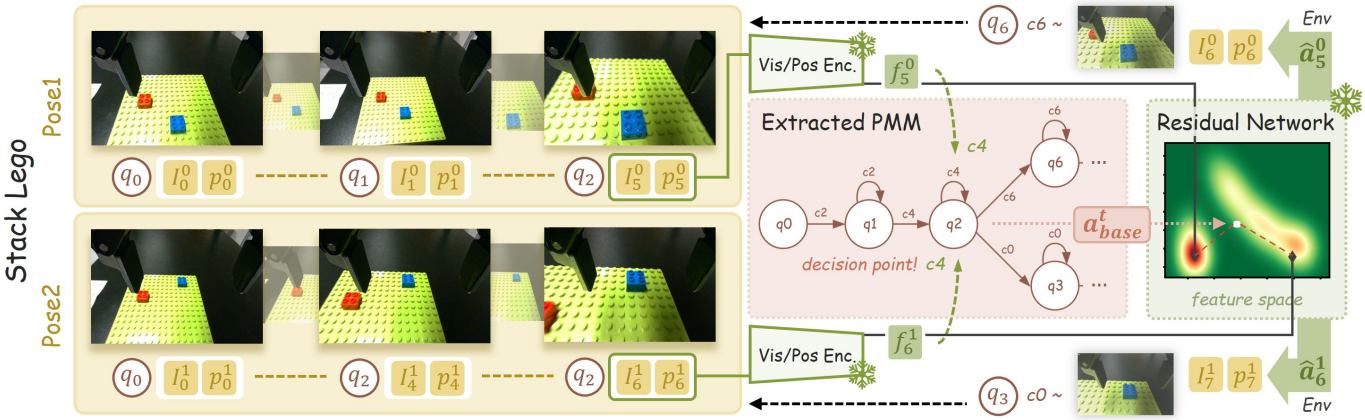

Fig. 6. **Mechanism of Logical Branching at Decision Points.** We visualize how **ENAP** handles multi-modal `StackLego` tasks. At the critical decision node (q_2), the extracted PMM identifies a branching structure based on distinct future outcomes (q_3 vs q_6). By conditioning on the goal-specific observation, the residual policy guides the transition into the correct logical mode. For the state machine involving more Lego start poses, please refer to Section E-B.

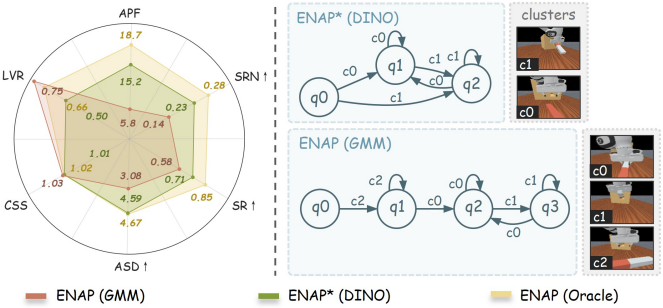

Fig. 7. **Comparison of Learned Policies from** `PegInsert`. **(Left)** Radar chart of six structural metrics. **(Right)** Visualized PMMs for DINO and GMM variants. For the PMM from Oracle, see Fig. 1.

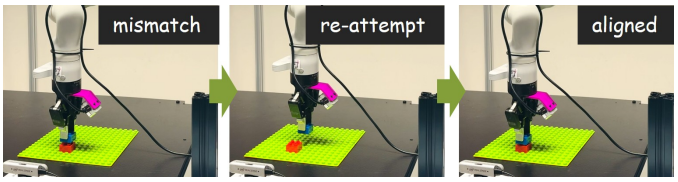

Fig. 8. **Failure Recovery in** `StackLego`.

`MultiGoalPushT` task presents a challenging optimization landscape due to its multi-modal distribution, where the initial state maps to divergent optimal trajectories, causing monolithic policies to average conflicting gradients and fail. The training data is a mixture of trajectories from two separate policies sharing a vision encoder. As shown in Table V, **ENAP** significantly outperforms all baselines in both overall success rate ("Either" Goal) and intention adherence ("Nearest" Goal).

E. Ablation Studies

We investigate two critical components governing structure discovery: (1) the similarity threshold τ_{sim}, which determines the granularity of the automaton; and (2) the training of the RNN, which motivates state differentiation. As visualized in Fig. 9, performance with an intermediate τ_{sim} and all RNN training losses peaks, validating the efficacy of our design. Additional ablations are provided in the Appendix.

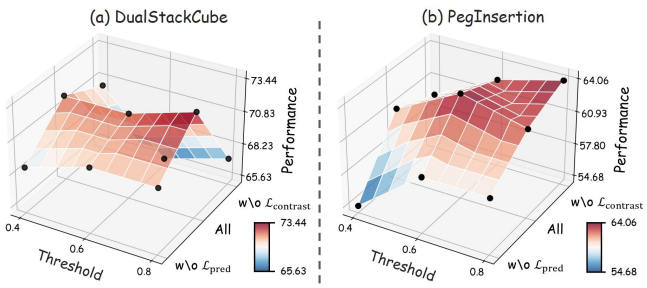

Fig. 9. **Ablation on Structure Discovery Parameters.** Success rates across varying similarity thresholds τ_{sim} and RNN loss configurations. Black dots indicate measured data points, while the surface shows interpolated values.

V. Conclusion

What is a good abstraction? Defining a "good" abstraction is ill-posed, as the appropriate granularity differs across contexts. In our paper, abstraction quality is performance-driven: the optimal structure is the one that best minimizes the downstream policy objective, which is why different network backbones yield divergent abstractions (Fig. 7).

Limitations. First, the quality of the emergent structure is sensitive hyperparameters. Furthermore, scaling to cross-task and cross-embodiment settings remains an open direction, for instance, by extracting abstractions over the semantic token spaces of generalist VLA models.

In this work, we presented **ENAP**, a framework for learning emergent neural automaton policies from visuomotor trajectories. By unifying active automata inference with residual control, **ENAP** unsupervisedly discovers a bi-level structure where a discrete Mealy machine governs high-level planning and networks handle continuous execution. This representation not only enables sample-efficient learning in long-horizon tasks but also provides an explicit, interpretable map of the agent's latent decision logic, thus offering a viable path toward generalizable and interpretable intelligence.

References

[1] Christopher Amato, Blai Bonet, and Shlomo Zilberstein. Finite-state controllers based on mealy machines for centralized and decentralized pomdps. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pages 1052–1058, 2010. 2, 12

[2] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2): 87–106, 1987. 3

[3] Masataro Asai and Alex Fukunaga. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *Proceedings of the aaai conference on artificial intelligence*, volume 32, 2018. 12

[4] Paul B Badcock, Karl J Friston, Maxwell JD Ramstead, Annemie Ploeger, and Jakob Hohwy. The hierarchically mechanistic mind: an evolutionary systems theory of the human brain, cognition, and behavior. *Cognitive, Affective, & Behavioral Neuroscience*, 19(6):1319–1351, 2019. 1

[5] Mattijs Baert, Sam Leroux, and Pieter Simoens. Learning task specifications from demonstrations as probabilistic automata. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8267–8274. IEEE, 2025. 12

[6] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024. 6, 7

[7] Anthony R Cassandra. A survey of pomdp applications. In *Working notes of AAAI 1998 fall symposium on planning with partially observable Markov decision processes*, volume 1724, 1998. 2

[8] 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. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025. 6, 7

[9] Rohan Chitnis, Tom Silver, Joshua B Tenenbaum, Tomas Lozano-Perez, and Leslie Pack Kaelbling. Learning neuro-symbolic relational transition models for bilevel planning. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4166–4173. IEEE, 2022. 12

[10] Alexander Clark. Distributional learning of some context-free languages with a minimally adequate teacher. In *International Colloquium on Grammatical Inference*, pages 24–37. Springer, 2010. 3

[11] Bruno Da Silva, George Konidaris, and Andrew Barto. Learning parameterized skills. *arXiv preprint arXiv:1206.6398*, 2012. 12

[12] Mohamad H Danesh, Anurag Koul, Alan Fern, and Saeed Khorram. Re-understanding finite-state representations of recurrent policy networks, 2021. *URL https://arxiv.org/abs/2006*, 3745. 12

[13] Sahil Rajesh Dhayalkar. Neural networks as universal finite-state machines: A constructive deterministic finite automaton theory. *arXiv preprint arXiv:2505.11694*, 2025. 12

[14] Michael R Fellows and Michael A Langston. An analogue of the myhill-nerode theorem and its use in computing finite-basis characterizations. In *30th Annual Symposium on Foundations of Computer Science*, pages 520–525. IEEE, 1989. 3

[15] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proceedings of the international conference on automated planning and scheduling*, volume 30, pages 440–448, 2020. 12

[16] Georgios Giantamidis, Stavros Tripakis, and Stylianos Basagiannis. Learning moore machines from input–output traces. *International Journal on Software Tools for Technology Transfer*, 23(1):1–29, 2021. 2

[17] Edward Groshev, Maxwell Goldstein, Aviv Tamar, Siddharth Srivastava, and Pieter Abbeel. Learning generalized reactive policies using deep neural networks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, pages 408–416, 2018. 12

[18] Huihui Guo, Fan Wu, Yunchuan Qin, Ruihui Li, Keqin Li, and Kenli Li. Recent trends in task and motion planning for robotics: A survey. *ACM Computing Surveys*, 55(13s):1–36, 2023. 1

[19] Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025. 6, 7

[20] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024. 1, 6

[21] George Konidaris and Andrew Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. *Advances in neural information processing systems*, 22, 2009. 12

[22] George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61:215–289, 2018. 12

[23] Anurag Koul, Sam Greydanus, and Alan Fern. Learning finite state representations of recurrent policy networks. *arXiv preprint arXiv:1811.12530*, 2018. 12

[24] Hadas Kress-Gazit, Morteza Lahijanian, and Vasumathi Raman. Synthesis for robots: Guarantees and feedback for robot behavior. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1):211–236, 2018. 2

[25] Bowen Li, Tom Silver, Sebastian Scherer, and Alexander Gray. Bilevel learning for bilevel planning. *arXiv preprint arXiv:2502.08697*, 2025. 12

[26] Yongming Li and Witold Pedrycz. The equivalence between fuzzy mealy and fuzzy moore machines. *Soft Computing*, 10(10):953–959, 2006. 2

[27] Pierrick Lorang, Hong Lu, Johannes Huemer, Patrik Zips, and Matthias Scheutz. Few-shot neuro-symbolic imitation learning for long-horizon planning and acting. *arXiv preprint arXiv:2508.21501*, 2025. 12

[28] Jan Lunze and Françoise Lamnabhi-Lagarrigue. *Handbook of hybrid systems control: theory, tools, applications*. Cambridge University Press, 2009. 2

[29] Xusheng Luo and Changliu Liu. Simultaneous task allocation and planning for multi-robots under hierarchical temporal logic specifications. *IEEE Transactions on Robotics*, 2025. 2

[30] Tengfei Ma, Patrick Ferber, Siyu Huo, Jie Chen, and Michael Katz. Online planner selection with graph neural networks and adaptive scheduling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5077–5084, 2020. 12

[31] Yueen Ma, Zixing Song, Yuzheng Zhuang, Jianye Hao, and Irwin King. A survey on vision-language-action models for embodied ai. *arXiv preprint arXiv:2405.14093*, 2024. 1

[32] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science robotics*, 7(66):eabm6074, 2022. 1

[33] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B Tenenbaum, and Jiajun Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. *arXiv preprint arXiv:1904.12584*, 2019. 12

[34] Leland McInnes, John Healy, and Steve Astels. Hdbscan: Hierarchical density based clustering. *Journal of Open Source Software*, 2(11):205, 2017. 3

[35] Geoffrey J McLachlan and Suren Rathnayake. On the number of components in a gaussian mixture model. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(5):341–355, 2014. 6, 7

[36] Oier Mees, Lukas Hermann, and Wolfram Burgard. What matters in language conditioned robotic imitation learning over unstructured data. *IEEE Robotics and Automation Letters*, 7(4):11205–11212, 2022. 6, 7

[37] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3):7327–7334, 2022. 6

[38] Christoph Molnar, Giuseppe Casalicchio, and Bernd Bischl. Interpretable machine learning–a brief history, state-of-the-art and challenges. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 417–431. Springer, 2020. 2

[39] Tong Mu, Yihao Liu, and Mehran Armand. Look before you leap: Using serialized state machine for language conditioned robotic manipulation. *arXiv preprint arXiv:2503.05114*, 2025. 12

[40] Tongzhou Mu, Zhan Ling, Fanbo Xiang, Derek Yang, Xuanlin Li, Stone Tao, Zhiao Huang, Zhiwei Jia, and Hao Su. Maniskill: Generalizable manipulation skill benchmark with large-scale demonstrations. *arXiv preprint arXiv:2107.14483*, 2021. 6

[41] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marcin Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaa El-Nouby, and Mahmoud Assran. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023. 4

[42] Hanna M Pasula, Luke S Zettlemoyer, and Leslie Pack Kaelbling. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research*, 29: 309–352, 2007. 12

[43] Karl Pertsch, Youngwoon Lee, and Joseph Lim. Accelerating reinforcement learning with learned skill priors. In *Conference on robot learning*, pages 188–204. PMLR, 2021. 12

[44] Michael Poli, Stefano Massaroli, Luca Scimeca, Sanghyuk Chun, Seong Joon Oh, Atsushi Yamashita, Hajime Asama, Jinkyoo Park, and Animesh Garg. Neural hybrid automata: Learning dynamics with multiple modes and stochastic transitions. *Advances in Neural Information Processing Systems*, 34:9977–9989, 2021. 2, 12

[45] Moritz Reuss, Ömer Erdinç Yağmurlu, Fabian Wenzel, and Rudolf Lioutikov. Multimodal diffusion transformer: Learning versatile behavior from multimodal goals. *arXiv preprint arXiv:2407.05996*, 2024. 6, 7

[46] Moritz Reuss, Hongyi Zhou, Marcel Rühle, Ömer Erdinç Yağmurlu, Fabian Otto, and Rudolf Lioutikov. Flower: Democratizing generalist robot policies with efficient vision-language-action flow policies. *arXiv preprint arXiv:2509.04996*, 2025. 6, 7

[47] Pierre Sermanet, Tianli Ding, Jeffrey Zhao, Fei Xia, Debidatta Dwibedi, Keerthana Gopalakrishnan, Christine Chan, Gabriel Dulac-Arnold, Sharath Maddineni, Nikhil J Joshi, et al. Robovqa: Multimodal long-horizon reasoning for robotics. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 645–652. IEEE, 2024. 1

[48] Muzammil Shahbaz and Roland Groz. Inferring mealy machines. In *International Symposium on Formal Methods*, pages 207–222. Springer, 2009. 2

[49] Tom Silver. *Neuro-Symbolic Learning for Bilevel Robot Planning*. PhD thesis, Massachusetts Institute of Technology, 2024. 2

[50] Tom Silver, Rohan Chitnis, Aidan Curtis, Joshua B Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Planning with learned object importance in large problem instances using graph neural networks. In *Pro-*

ceedings of the AAAI conference on artificial intelligence, volume 35, pages 11962–11971, 2021. 12

[51] Tom Silver, Ashay Athalye, Joshua B Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Learning neuro-symbolic skills for bilevel planning. *arXiv preprint arXiv:2206.10680*, 2022. 2

[52] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 6

[53] Zeming Wei, Xiyue Zhang, and Meng Sun. Extracting weighted finite automata from recurrent neural networks for natural languages. In *International Conference on Formal Engineering Methods*, pages 370–385. Springer, 2022. 12

[54] Weixiao Zhan, Qiyue Dong, Eduardo Sebastián, and Nikolay Atanasov. Latmos: Latent automaton task model from observation sequences. *arXiv preprint arXiv:2503.08090*, 2025. 12

[55] Edwin Zhang, Yujie Lu, William Wang, and Amy Zhang. Language control diffusion: Efficiently scaling through space, time, and tasks. *arXiv preprint arXiv:2210.15629*, 2022. 6, 7

[56] Yihao Zhang, Zeming Wei, and Meng Sun. Automata extraction from transformers. *arXiv preprint arXiv:2406.05564*, 2024. 12

[57] Zhigen Zhao, Shuo Cheng, Yan Ding, Ziyi Zhou, Shiqi Zhang, Danfei Xu, and Ye Zhao. A survey of optimization-based task and motion planning: From classical to learning approaches. *IEEE/ASME Transactions on Mechatronics*, 2024. 1

Appendix A
Related Works

A. Neuro-Symbolic Learning for Robotics

Neuro-symbolic approaches in robotics aim to coordinate the generalization capabilities of symbolic reasoning with the perceptual flexibility of neural networks, particularly within the domain of Task and Motion Planning (TAMP). While foundational frameworks like PDDLStream [15] successfully integrated discrete search with continuous motion planning, they relied on manually specified symbolic models. To overcome this, recent research has pivoted toward learning abstractions directly from data. In the realm of action abstraction, approaches have evolved from discovering temporal options via skill chaining [21] to learning deep parameterized skills [43, 11] that enable continuous control within symbolic plans. Complementary work in state abstraction seeks to ground continuous sensorimotor data into discrete symbols [22], utilizing techniques ranging from deep generative models [3] to neuro-symbolic concept learners [33] to construct latent states compatible with symbolic planners. Building upon these learned abstractions, model learning focuses on inferring the transition dynamics that govern symbolic interactions. This area has progressed from learning probabilistic rules in stochastic domains [42] to employing Graph Neural Networks (GNNs) for modeling relational dynamics [9] and hybrid automata for stochastic transitions [44]. Finally, to enable execution across varying task configurations, Generalized planning methods train neural policies to approximate symbolic plans [17], often utilizing GNNs to reason about object importance in large-scale environments [50] or to select appropriate planners [30].

While prior neuro-symbolic methods successfully integrate learning and planning, they typically require pre-defined symbolic predicates. In contrast, **ENAP** introduces a label-free, self-supervised framework that autonomously discovers interpretable automata directly from raw visuomotor data, bridging the gap between low-level sensorimotor control and high-level symbolic reasoning without manual engineering.

B. Symbolic Structure Discovery

Discovering symbolic structures from continuous data is pivotal for enhancing interpretability and planning efficiency in robotics. A broad spectrum of methods has been proposed to infer such structures, ranging from learning PDDL operators and predicates for bilevel planning [25] to discovering symbolic graphs and rules via neuro-symbolic imitation learning [27]. While these approaches successfully capture high-level task logic, Finite State Machines (FSMs) offer a unique advantage: they provide a compact, topological representation of sequential dynamics that supports formal verification and reactive control. Specifically, Mealy machines have been shown to offer richer structural representations than Moore machines, making them particularly effective for POMDP controllers [1].

Driven by these advantages, a prominent line of research focuses on automata extraction from data. Building on classical approaches like the L^* algorithm and the Myhill-Nerode theorem [13], recent works have successfully extracted Deterministic Finite Automata (DFA) from RNNs to verify formal properties [12, 23] and from Transformers to reveal their operational mechanisms [56]. To address stochasticity, extensions such as Weighted Finite Automata (WFA) extraction have been proposed for natural language processing [53]. In the robotics domain, latent automata models like LATMOS [54] and serialized state machines [39] have been developed to factorize long-horizon tasks into executable subgoals. Similarly, learning probabilistic automata (PDFA) has proven effective for capturing temporal dependencies and expert preferences directly from demonstrations [5].

While existing methods excel in specific niches, they typically require either discretized inputs or predefined knowledge like cluster counts. **ENAP** bridges these gaps by combining self-supervised continuous embedding with adaptive automata inference, enabling the emergent discovery of probabilistic, task-relevant structures without manual specification.

Appendix B
L^* Algorithm Walkthrough: Classical vs. Ours

A. L^* Algorithm

To provide intuition for the L^* algorithm, we present a concrete example of learning a target FSM that accepts strings with an even number of a's and b's.

We start with $\mathcal{U} = \mathcal{E} = \{\epsilon\}$. Since $\text{row}(a) \neq \text{row}(\epsilon)$, the table is not closed. We promote a to $\mathcal{U}$. The resulting table is closed and consistent, yielding hypothesis $\mathcal{M}_1$. However, $\mathcal{M}_1$ incorrectly rejects the counterexample bb.

T_1	ϵ
ϵ	1
a	0
b	0
aa	1
ab	0

Fig. 1. **Initial Hypothesis**

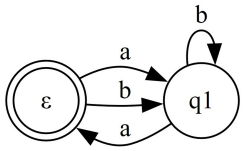

Fig. 2. **Hypothesis $\mathcal{M}_1$**

We add the counterexample bb (and its prefix b) to $\mathcal{U}$. The new table becomes inconsistent: states a and b appear identical ($\text{row}(a) = \text{row}(b) = [0]$), but their extensions differ ($\text{row}(a \cdot a) \neq \text{row}(b \cdot a)$). Then, we add the distinguishing suffix a to $\mathcal{E}$.

T_2	ϵ	a
ϵ	1	0
a	0	1
b	0	0
bb	1	0
aa	1	0
ab	0	0
ba	0	0
bba	0	1
bbb	0	0

Fig. 3. **Closed but Inconsistent**

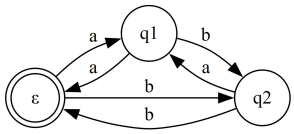

Fig. 4. **Hypothesis $\mathcal{M}_2$**

With $\mathcal{E} = \{\epsilon, a\}$, the table is closed but still inconsistent for other pairs (e.g., b and ab on suffix b). After adding suffix

T_3	ϵ	a	b
ϵ	1	0	0
a	0	1	0
b	0	0	1
$b \cdot b$	1	0	0
$a \cdot b$	0	0	0
$a \cdot b \cdot b$	0	1	0
$a \cdot a$	1	0	0
$b \cdot a$	0	0	0
$b \cdot b \cdot a$	0	1	0
$b \cdot b \cdot b$	0	0	1
$a \cdot b \cdot a$	0	0	1
$a \cdot b \cdot b \cdot a$	1	0	0
$a \cdot b \cdot b \cdot b$	0	0	0

Fig. 5. **Final Machine**

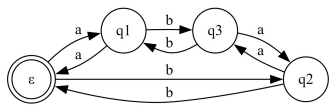

Fig. 6. **Final $\mathcal{M}_3$**

b, the final table is both closed and consistent, identifying 4 distinct states isomorphic to the target.

B. Extended L^* Algorithm (Algorithm 1)

*1) **Experiment Setup**:* To provide a transparent walk-through of the structure extraction process, we utilize the discrete 4×4 FrozenLake environment. And we use RNNs with random initialization. Since the state space is inherently discrete, we bypass the clustering phase and map the grid position directly to the input alphabet Σ, such that the symbol c_t corresponds to the grid index (e.g., c_{14} represents position 14). We collect two successful trajectories from Start (c_0) to Goal (c_{15}) that share a common history but diverge at a decision point.

- τ_1: $0 \xrightarrow{D} 4 \xrightarrow{D} 8 \xrightarrow{R} 9 \xrightarrow{D} \mathbf{13} \xrightarrow{R} 14 \xrightarrow{R} 15$
- τ_2: $0 \xrightarrow{D} 4 \xrightarrow{D} 8 \xrightarrow{R} 9 \xrightarrow{R} \mathbf{10} \xrightarrow{D} 14 \xrightarrow{R} 15$

*2) **Iteration**:* This setup is specifically chosen to demonstrate how the algorithm iteratively discovers the topological branching at state c_9 and the subsequent state convergence at c_{14}. In the following figures, edges are annotated with the tuple: input symbol, action list, probability, and $\mathrm{NIS}(\cdot)$.

The algorithm initializes with the start state q_0 (history ϵ) and expands via *Self-Supervised MQ* to find reachable states. It identifies q_1 (history 0) via transition c_0. However, due to limited exploration, the initial hypothesis (Fig. 8) only loops at q_1. Running the *NFA-style EQ* on trajectory τ_1 reveals a failure: the sequence $0 \xrightarrow{D} 4 \xrightarrow{D} 8$ cannot be explained. The prefix $(0, 4, 8)$ is returned as a counterexample.

Adding the counterexample to the prefix set $\mathcal{U}$ triggers *Closedness* checks, expanding the state space to cover the history $0 \rightarrow 4 \rightarrow 8$. The new hypothesis (Fig. 9) successfully models the linear path up to q_3. However, it fails to validly explain the divergence into c_{13} or c_{10}. The prefix $(0, 4, 8, 9, 13)$ is returned as a counterexample.

Fig. 7. **FrozenLake Map**.

Incorporating the new counterexample, the algorithm discovers the critical branching state q_4. The MQ reveals two distinct transitions from q_4: one leading to q_5 and another leading to q_4 itself (self-loop for $\cdots - 10$). The hypothesis (Fig. 10) now captures the "lower" path (τ_1) correctly but still misinterprets the "upper" path (τ_2) convergence, failing at $(0, 4, 8, 9, 10, 14)$.

The final counterexample forces the expansion of the "lower" path state q_6 (history $\cdots - 10$). Crucially, the MQ discovers that both q_5 (from τ_1) and q_6 (from τ_2) transition to the same next-step embedding region under input c_{14}, which maps to a unified state q_7. The resulting hypothesis (*Stable Phase Pruning* applied; see Fig. 11) correctly models the full topology: common past $\rightarrow$ branching $\rightarrow$ convergence. The EQ finds no further counterexamples, and the hypothesis stabilizes.

In classical L^*, the algorithm constructs a full transition table by querying the output for every prefix u extended by every symbol $\sigma \in \Sigma$. However, in our setting with demonstrations, many (u, σ) combinations are physically unobserved. To address this sparsity, our Generalized MQ retrieves only the empirically observed next-step embeddings h_{next} from the dataset, instead of exhaustively querying all extensions.

Appendix C
Theoretical Motivation for RNN Encoder

Our framework relies on the premise that the continuous hidden states of a trained RNN can represent the discrete states of an FSM. This architectural choice is theoretically motivated by the *saturation* (the hidden states $h_t \in \mathbb{R}^h$ tend to cluster around the vertices of the hypercube $\{\pm 1\}^d$ as training progresses) dynamics of RNNs with $\tanh$ activations. This allows us to treat the continuous vector space as an approximation of a discrete state space. The following proposition formalizes the conditions under which cosine similarity serves as a reliable metric for distinguishing these discrete states.

Proposition C.1 (Identifiability via ϵ-Saturated RNNs): Let $h_1, h_2 \in \mathbb{R}^h$ be two normalized hidden state vectors. Let $\tilde{h}_1, \tilde{h}_2 \in \{\pm 1\}^d$ be their corresponding saturated discrete versions. Assume the RNN operates in an ϵ-saturated regime, such that $\|h_i - \tilde{h}_i\|_2 \leq \epsilon$ for $i \in \{1, 2\}$. Then, if the cosine similarity satisfies:

$$\cos(h_1, h_2) \geq 1 - \kappa \quad \text{with} \quad \sqrt{\kappa} < \sqrt{2}\left(\frac{1}{\sqrt{d}} - \epsilon\right),$$

The two vectors correspond to the same discrete automaton state (i.e., $\tilde{h}_1 = \tilde{h}_2$).

Proof: By the triangle inequality, the distance between the discrete states is bounded by:

$$\begin{aligned}\|\tilde{h}_1 - \tilde{h}_2\|_2 &= \|\tilde{h}_1 - h_1 + h_1 - h_2 + h_2 - \tilde{h}_2\|_2 \\ &\leq \|\tilde{h}_1 - h_1\|_2 + \|h_1 - h_2\|_2 + \|h_2 - \tilde{h}_2\|_2 \\ &\leq 2\epsilon + \|h_1 - h_2\|_2.\end{aligned}$$

Since saturated RNN states take discrete values in $\{\pm 1\}^d$, the minimum non-zero Euclidean distance between any two

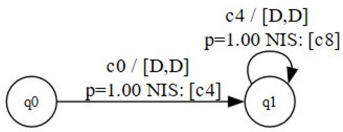

Fig. 8. **Iteration 1**: Initial hypothesis.

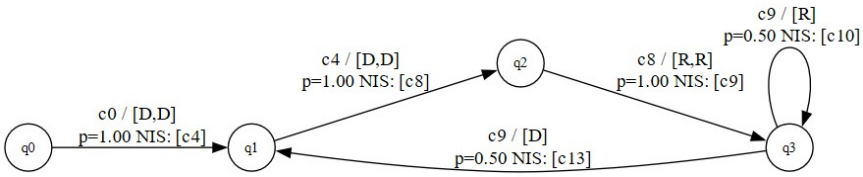

Fig. 9. **Iteration 2**: Linear chain extension.

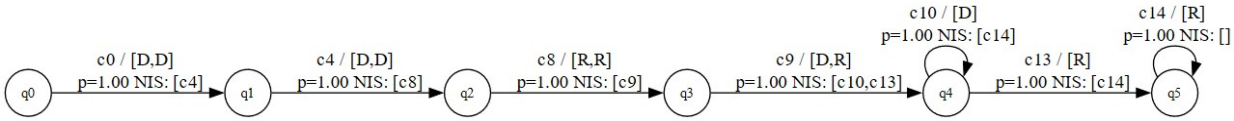

Fig. 10. **Iteration 3**: Discovery of the decision point.

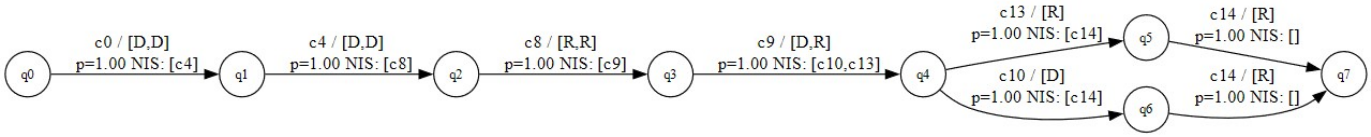

Fig. 11. **Iteration 4**: Stabilized hypothesis capturing the full topology.

distinct vertices is 2. Then, using the relationship between Euclidean distance and cosine similarity for normalized vectors: $\|h_1 - h_2\|_2 = \sqrt{2(1 - \cos(h_1, h_2))}$. Given the condition $\cos(h_1, h_2) \geq 1 - \kappa$, we have $\|h_1 - h_2\|_2 \leq \sqrt{2\kappa}$. Substituting this into the inequality:

$$\|\tilde{h}_1 - \tilde{h}_2\|_2 \leq 2\epsilon + \sqrt{2\kappa}.$$

For $\tilde{h}_1$ and $\tilde{h}_2$ to be the same state, the error bound must be strictly less than the minimum distance between distinct discrete states. The condition $\sqrt{\kappa} < \sqrt{2}(1/\sqrt{d} - \epsilon)$ ensures that the vectors fall within the same decision boundary of the discrete state, implying $\tilde{h}_1 = \tilde{h}_2$.

■

This result confirms that by enforcing saturation and using a strict cosine threshold, we can rigorously recover the underlying discrete phases from the continuous RNN embeddings.

Appendix D
Experiments: Implementation Details

A. Task Specifications

We evaluate **ENAP** across three domains, each with distinct robotic platforms and observation spaces. For *Complex Manipulation*, we utilize the ManiSkill benchmark (Franka Emika Panda arm), providing the agent with RGB images and proprioceptive states. For *Long-Horizon TAMP*, we adopt the CALVIN environment (Franka Panda arm), where observations include base and wrist camera feeds alongside language task descriptions. Finally, our *Real-World Manipulation* experiments employ a Kinova Gen3 arm, accessing base/wrist camera streams and end-effector pose. Specific task definitions and success criteria are detailed below.

1) ***PegInsertionSide (Sim)****:* This task requires picking up a bi-colored peg and inserting either the orange or white end into a horizontal box hole. The environment features significant variability: peg lengths are sampled from $U[0.15, 0.20]$m and radii from $U[0.015, 0.025]$m, with initial poses randomized in position and rotation. Success is defined as inserting either end at least 0.03m into the hole (radius $r_{\text{peg}} + 0.02$m).

2) ***DualStackCube (Sim)****:* The agent must stack two colored cubes (Red/Green) in any valid vertical configuration. Cube positions are initialized uniformly within a $0.2\text{m} \times 0.4\text{m}$ region. A trial is considered successful only if the top cube aligns within half-width of the bottom cube, remains static, and is fully released by the gripper.

3) ***MultiGoalPushT (Sim)****:* A multi-modal variant of the Push-T task where the agent must maneuver a T-shaped block to the nearest of two potential target poses. The T-block's initial pose is fully randomized. Success is achieved when the intersection-over-union (IoU) between the block and the designated target silhouette exceeds 0.90.

4) ***Sequential (Sim)****:* The task enforces a strict causal chain: (1) rotate the red block right ($> 60°$ yaw); (2) push the blue block left (x-displacement < -0.1m); (3) move the slider right (joint $\Delta < -0.15$); (4) toggle the LED switch; and (5) turn on the light bulb.

5) ***Hierarchical (Sim)****:* The task combines articulated object manipulation and imposes temporal logic (e.g., a drawer must be opened before placing the block inside it): (1) open the drawer (joint $\Delta \geq 0.12$); (2) lift the red block from the table ($z > 0.05$m); (3) place it inside the drawer; (4) lift the pink block from the slider ($z > 0.03$m); and (5) stack it onto another object (static stability required).

6) ***Hanger (Real)****:* The task requires unhooking a clothes hanger from one end of a rack and transferring it to the opposite side. An obstacle in the middle prevents direct lateral

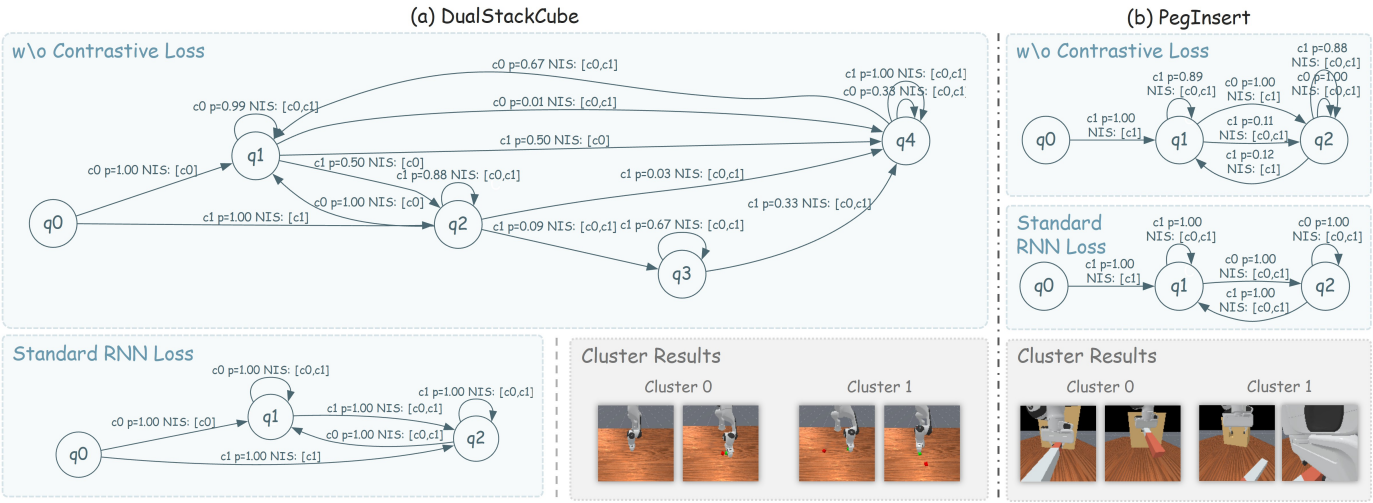

Fig. 12. **Impact of Phase-Aware Contrastive Loss.** Without phase-aware contrastive loss, the automaton is fragmented with ambiguous transitions.

movement, forcing a complex lift-over trajectory. Success is defined as the hanger being stably hooked on the target side.

*7) **MultiPickPlace (Real)**:* It involves sorting three colored cans (red, green, blue) into corresponding bowls (red, green, blue) fixed on the table. The cans are initialized randomly on the same side, requiring the agent to identify and match object-container pairs.

*8) **StackLego (Real)**:* The task demands high-precision assembly without force feedback. A blue Lego brick is initialized at discrete locations on a green base plate and must be stacked onto a fixed red brick. We adopt a graded success metric: 1.0 for a full insertion (studs interlocked) and 0.5 for a stable stack (brick resting on top) without interlocking.

TABLE I
NETWORK ARCHITECTURES FOR COMPLEX MANIPULATION.

Component	ENAP (Oracle)	ENAP* (DINO)
- Encoder Architecture		
Visual Backbone	NatureCNN (3-layer Conv)	DINOv2 ViT-S/14 (Frozen)
Visual Head	Linear → 256	Spatial Softmax → MLP → 256
Proprioception	Linear → 256	Fourier (7-freq) → SpecNorm → 128
- Feature Dimensions (Vis + Pos [+ Delta])		
`StackCube`	512 (256+256)	384 (256+128)
`Peg` / `PushT`	512 (256+256+0)	512 (256+128+128)
- Residual Policy Network (Hidden Units)		
`StackCube`	512	512
`Peg` / `PushT`	256	256
- PMM Structure Discovery		
RNN Encoder	Hidden Dim 64	
State Embed	Learnable Embedding (Dim 16)	

TABLE II
NETWORK ARCHITECTURES FOR LONG-HORIZON TAMP.

Component	ENAP (FLOWER)
- Encoder Architecture	
Visual Backbone	Florence-2 DaVit (Frozen)
Language Encoder	Florence-2 Text Encoder
Fusion	Concat → Florence-2 Encoder → Mean Pool
Feature Dim	1024
- Residual Policy Network	
Architecture	Linear (2118 → 256) → ReLU → Linear (256 → 70)
Hidden Units	256
Output	70-dim Action Sequence (10 × 7)
- PMM Structure Discovery	
RNN Encoder	Hidden Dim 64
State Embed	Learnable Embedding (Dim 16)
Input Dim	Action (70) + State Embed (16)

TABLE III
NETWORK ARCHITECTURES FOR REAL-WORLD MANIPULATION.

Component	ENAP* (DINO)
- Encoder Architecture	
Visual Backbone	DINOv2 ViT-S/14 (Frozen)
Visual Head	Spatial Softmax → MLP (768→128)
Proprioception	Fourier (7-freq) → SpecNorm → 128
Fusion	Base (128) + Wrist (128) + Prop (128) = 384
- Policy Network	
Residual MLP	Linear (771 → 256) → ReLU → Output (3)
Gripper Head	MLP (Feat + Center → 128 → 2)
Controller	Cartesian PID (K_p=5.0, K_d=0.4)
- PMM Structure Discovery	
RNN Encoder	Hidden Dim 64
State Embed	Learnable Embedding

B. Network Architecture

To ensure reproducibility, we detail the specific architectures for the encoders, RNNs, and residual networks, along with the hyperparameters used for training and structure discovery. We provide detailed network specifications for the complex manipulation tasks, the long-horizon TAMP tasks, and the real-world task in Table I, Table II and Table III, respectively.

APPENDIX E
EXPERIMENTS: FURTHER ANALYSIS

A. Impact of Contrastive Loss on PMM

To further validate our design, we visualize the impact of the phase-aware contrastive loss $\mathcal{L}_{\text{contrast}}$ on the extracted PMM structure in Fig. 12. We apply K-means clustering with the number of effective clusters inferred by HDBSCAN to filter out noise points. When training the RNN without this loss (Top), the resulting automaton is fragmented and highly

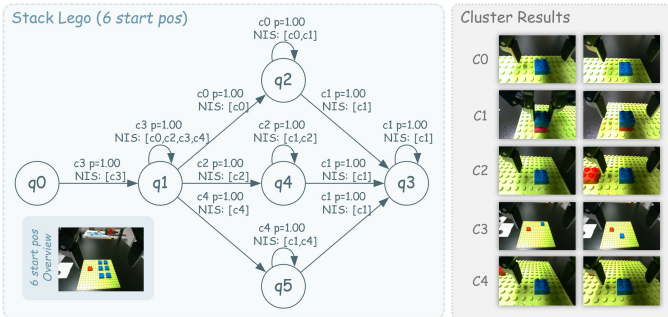

Fig. 13. **Generalization of Structural Branching.** Instead of overfitting to six distinct branches, the model discovers three logical pathways (q_2, q_4, q_5), grouping visually and kinetically similar poses (e.g., along the line of sight) into shared phases.

stochastic, with numerous spurious transitions. This indicates that the latent states fail to effectively summarize history. This confirms that explicitly shaping the latent geometry is crucial for discovering interpretable and control-relevant structures.

B. Adaptive Abstraction in Complex Multi-Modal Tasks

We further investigate the scaling of structure discovery by extending the StackLego task to six initial positions. We assign points from the noise cluster obtained by HDBSCAN to the nearest valid cluster. As shown in Fig. 13, the extracted PMM does not naively spawn six separate branches; instead, it adaptively groups kinetically similar poses (e.g., those aligned along the camera's optical axis) into three shared latent modes. This demonstrates that **ENAP** is not merely memorizing task instances but is optimizing for a compact policy representation that minimizes the residual learning burden, effectively discovering the "minimal sufficient structure" for control.

APPENDIX F
DISCUSSION: A POMDP PERSPECTIVE

We formulate the justification for **ENAP**'s bi-level architecture by aligning it with the Bellman equation. Consider the true value function $V(q, o, s)$ of the underlying POMDP, conditioned on the latent task mode q, observation o, and environment state s. The Bellman equation decomposes as:

$$V(q,o,s) = \sum_{a,q'} P(q', a \mid q, o) \Big[R(s,a) + \gamma \sum_{s',o'} P(s', o' \mid s, a)\, V(q', o', s') \Big] \tag{7}$$

where the joint transition-policy distribution $P(q', a|q, o)$ represents the coupled decision process: choosing an action a and transitioning to a new internal mode q'. In our framework, this distribution is explicitly factorized using the PMM structure:

$$P(q', a \mid q, o) = \underbrace{\delta(q'|q,c)}_{\text{Topological Transition}} \cdot \sum_{a^t_{\text{base}}} \underbrace{\lambda(a^t_{\text{base}}|q,c)}_{\text{Action Prior}} \cdot \underbrace{\pi_\psi(a \mid q, o, a^t_{\text{base}})}_{\text{Residual Correction}} \tag{8}$$

The PMM captures the global dynamics (δ) and coarse intent (λ), leaving the neural network to focus solely on the local residual term π_ψ. This reduces the complexity of the function approximation task, explaining the framework's high sample efficiency.

Furthermore, by augmenting the state space to $\tilde{s} = (q, o)$, we enable planning over the belief space. The value function over this augmented state is given by:

$$\hat{V}(\tilde{s}) = \max_a \left[\hat{r}(\tilde{s}, a) + \gamma \sum_{\tilde{s}'} \hat{P}_{\tilde{s}}(\tilde{s}'|\tilde{s}, a) \hat{V}(\tilde{s}') \right] \tag{9}$$

Assuming the learned PMM states Q effectively summarize the history h (i.e., q is a sufficient statistic), we have the approximation $\hat{V}(\tilde{s}) \approx \mathbb{E}_{s \sim P(s|h)}[V(q, o, s)]$. Our explicit recording of next-input signatures on PMM edges serves to approximate the transition dynamics $\hat{P}_{\tilde{s}}$, allowing the agent to deterministically track its state $\tilde{s}$ and execute optimal planning.