Tokenizer Choice Shapes Generalization in State-Centric Learning for Planning

Vishal Pallagani[1]* **Nitin Gupta**[2] **John A. Aydin**[2] **Biplav Srivastava**[2]
[1] Patronus AI [2]University of South Carolina

Abstract

State-centric learned planning trains a goal-conditioned transition model over state representations and recovers actions by matching predicted successor states to symbolic successors, but prior work has relied mainly on Weisfeiler–Leman (WL) representations, leaving the role of the representation map itself, the *tokenizer*, unclear. We ask how much of held-out, within-domain size generalization is determined by tokenizer choice. We answer with a controlled factorial study: five tokenizer families (WL, shortest-path, GraphBPE, SimHash, and a deterministic random reference) are swapped inside one fixed pipeline, crossed with two predictors, two predictive formulations, and two training protocols, and evaluated on held-out interpolation and strictly larger extrapolation instances of classical planning domains. Tokenizer choice is the dominant factor in the studied configurations: it explains 57% of held-out performance variance, more than predictor architecture or any other pipeline factor (each at most 1.2%), and the conclusion is robust to removing the largest domain. WL is strongest overall, yet no tokenizer wins everywhere; the leading representation changes by domain, leaders reverse between interpolation and extrapolation in 5 of 8 configuration families, and a per-domain analysis traces successes and failures to measurable properties of each representation. With the right tokenizer, tree predictors that are orders of magnitude smaller exceed the best published symmetry-aware transformer on 3 of 4 shared benchmark domains. Representation choice is therefore a first-order modeling decision for state-centric learned planning rather than a fixed preprocessing step. Code: `https://github.com/ai4society/planfm-state-centric-tokenization`.

1 Introduction

Classical planning [Ghallab et al., 2004] has traditionally been framed as symbolic search guided by domain models and heuristics. In generalized planning [Srivastava et al., 2008, Rivlin et al., 2020, Groshev et al., 2018], however, the objective is broader: a solver should transfer performance across families of related instances rather than solve one problem well at a time in isolation. Learning has therefore become a natural complement to classical planning because it can amortize reusable structural regularities across related problems. One prominent direction is action-centric learning, where models map problem descriptions directly to actions or plans. Plansformer [Pallagani et al., 2022], PlanGPT [Rossetti et al., 2024], and the more recent symmetry-aware transformer SymT [Fritzsche et al., 2026] are representative examples. These approaches have produced strong results, but the evolving world state remains implicit in the learned sequence model.

A parallel line of work studies state-centric transition-model planning. In this formulation, a learned transition predictor maps the current state and goal representations to either a next-state representation or a latent state update. A symbolic successor generator then selects executable successors that

*Work done while a Ph.D. student at the University of South Carolina.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

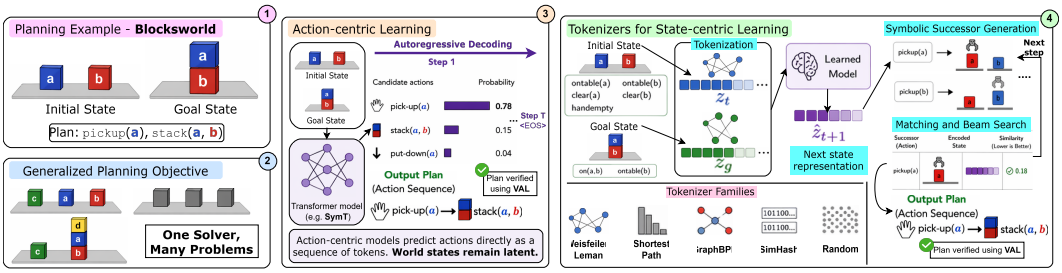

Figure 1: Overview of the setting studied in this paper. Left panel: generalized planning seeks transfer across related instances. Middle panel: action-centric models decode action sequences directly and keep world states implicit. Right panel: in the state-centric setting, tokenized state and goal representations condition a learned transition predictor, whose predicted next-state representation is matched against symbolic successors with beam search to recover a valid action sequence.

best match that prediction. This keeps state evolution explicit at inference time and allows every step to be checked against the domain transition model. Recent state-centric planners have shown that compact transition predictors paired with relational WL features [Chen et al., 2024b] can generalize well on held-out planning instances [Gupta et al., 2026]. Yet that success leaves an important question open: how much performance can be attributed to the state-centric prediction-and-search formulation, and how much to the representation it uses? Figure 1 summarizes this contrast and highlights the representation layer studied in this paper.

We use *tokenizer* in a broad machine-learning sense to denote the representation map that converts raw inputs into fixed-dimensional features consumed by a downstream predictor. In language models, tokenizers segment text. Here, for planning, the inputs are symbolic states and goals; the tokenizers in scope are therefore graph- and feature-based state encodings rather than textual subword segmenters. This distinction matters because the tokenizer determines which invariances, counts, and relational structure are visible to the transition predictor and symbolic search. The generalization we study is within-domain size generalization: tokenizers and predictors are fit on small training instances of a domain and evaluated on held-out instances of the same domain, split into interpolation (sizes within the training range) and extrapolation (strictly larger than every training instance).

Our contributions are: **(1)** a controlled empirical finding: across a factorial study of 40 configurations, tokenizer choice is the dominant factor in held-out generalization, explaining 57% of performance variance versus at most 1.2% for predictor, formulation, or training protocol, with leader reversals between interpolation and extrapolation in 5 of 8 configuration families; **(2)** a mechanistic per-domain analysis that traces tokenizer successes and failures to measurable representation properties (one-step prediction quality, embedding collisions, and search-budget dependence), together with controls isolating goal conditioning and dimensionality; and **(3)** the instrument that establishes both: a unified state-centric transition-model planning pipeline that supports interchangeable tokenizer families within a common symbolic search-and-validation stack, with a domain-dependent and pooled multi-domain evaluation design.

2 Related Work

This paper relates to action-centric generalized planning, state-centric transition-model planning, learned planning heuristics, and graph tokenization. Action-centric planners learn a direct mapping from a task description or planning context to actions or complete plans. Action schema networks [Toyer et al., 2018], Plansformer [Pallagani et al., 2022], PlanGPT [Rossetti et al., 2024], and SymT [Fritzsche et al., 2026] are representative examples, with SymT showing that symmetry-aware training improves extrapolation relative to earlier transformer baselines. State-centric transition-model planning instead keeps world-state evolution explicit. Earlier latent-space planning work learned compact state encodings and transition operators [Asai and Fukunaga, 2018, Asai et al., 2021], while recent generalized planning work studies goal-conditioned successor prediction with symbolic plan recovery [Gupta et al., 2026]. The closest prior work is Gupta et al. [2026], which compares compact transition predictors with WL and fixed-size factored representations. We extend that direction

by holding the state-centric search pipeline fixed and varying the tokenizer itself across five families, together with predictor, formulation, and training protocol.

A long line of work learns representations or policies for generalization directly: concept-language and feature-pool approaches [Martín and Geffner, 2004, Fern et al., 2006, Bonet and Geffner, 2018, Bonet et al., 2019, Francès et al., 2021], relational and graph-neural policies [Toyer et al., 2018, Garg et al., 2020, Karia and Srivastava, 2022, Kokel et al., 2021, Silver et al., 2024], and expressiveness analyses that explain when such representations must fail, with `logistics` a known hard case [Ståhlberg et al., 2022]. Those works learn representations for generalization; this study instead measures how swapping fixed representation families changes generalization inside one held-fixed pipeline, and learned feature-pool representations are a natural sixth tokenizer family for the framework. The closest empirical literature outside direct plan generation studies learned heuristics. Neural heuristics for classical planning have been analyzed with attention to sampling, uncertainty, and optimization stability [Ferber et al., 2022, Shen et al., 2020], while more recent comparisons argue that graph kernels and structured classical features can remain competitive and data-efficient [Chen et al., 2023, 2024c,a, Chen and Thiébaux, 2024]. Several tokenizers in our study are rooted in established graph-feature constructions: WL color refinement, shortest-path histograms, and SimHash [Shervashidze et al., 2011, Borgwardt and Kriegel, 2005, Charikar, 2002], while GraphBPE adapts byte-pair encoding to graph substructures [Shen and Póczos, 2024]. Closely related graph-tokenization work studies learned vocabularies for graph prediction and graph-transformer interfaces [Wang et al., 2025, Guo et al., 2026]. Our study connects these threads by asking how tokenization changes the feature space presented to a downstream transition predictor and, in turn, the symbolic search decisions induced by its predictions.

3 Tokenized Latent-Space Planning

We study a state-centric transition-model planning pipeline that predicts next-state embeddings and recovers executable actions by symbolic successor search, extending earlier transition-model planners [Gupta et al., 2026] to multiple tokenizer families and beam-search decoding.

3.1 Problem Setup

Let a planning problem be specified by a PDDL [McDermott et al., 1998] domain $d \in \mathcal{D}$, a problem instance $p \in \mathcal{P}_d$, an initial state $s_0 \in \mathcal{S}$, and a goal condition $g_p \in \mathcal{G}$. For any symbolic state s, let $\mathrm{atoms}(s)$ denote its grounded atom set. Let $\mathrm{atoms}_g(p)$ denote the grounded goal-atom set induced by g_p, and let O_p denote the problem objects together with domain constants. Figure 1 instantiates these symbols in BlocksWorld: $O_p = \{a, b\}$; $\mathrm{atoms}(s_0)$ contains `ontable(a)`, `ontable(b)`, `clear(a)`, `clear(b)`, and `handempty`; and $\mathrm{atoms}_g(p)$ contains `on(a,b)` and `ontable(b)`. Each training instance yields a symbolic trajectory $\xi_p = (s_0, s_1, \ldots, s_T)$, where s_{t+1} is the expert successor of s_t; in Figure 1, $\xi_p = (s_0, s_1, s_2)$ follows `pickup(a)` then `stack(a,b)`, with s_1 containing `holding(a)` and s_2 satisfying the goal atoms.

At step t, every tokenizer receives the same symbolic input

$$I(p,t) = \big(\mathrm{atoms}(s_t), \mathrm{atoms}_g(p), O_p, d, p\big)$$

and constructs an intermediate representation $r_{\tau,t} = T_\tau(I(p,t))$. Its state and goal encoders produce fixed-dimensional vectors

$$z_t = E_\tau^{\mathrm{state}}(r_{\tau,t}), \qquad z_g = E_\tau^{\mathrm{goal}}(\mathrm{atoms}_g(p), O_p, d, p), \qquad z_t, z_g \in \mathbb{R}^{m_\tau},$$

where m_τ is the output dimension of tokenizer family τ. For search, $\phi_\tau(s, g_p, p)$ denotes the same state encoder applied to an arbitrary symbolic state s, so $\phi_\tau(s_t, g_p, p) = z_t$. Symbolic successors are embedded with this map and compared to the predictor's next-state embedding.

The state and goal encoders use the same tokenizer-specific feature map. For WL, T_{WL} is the `wlplan` instance-learning-graph (ILG) backend, so $r_{\mathrm{WL},t} = G^{\mathrm{ilg}}_{p,s_t} = (V, E, \lambda_0)$ with object nodes O_p, fact nodes $F_{s_t} = \{\nu_a : a \in \mathrm{atoms}(s_t)\}$, and bidirectional incidence edges labeled by argument position. The goal encoder uses the same ILG construction on the goal atoms $\mathrm{atoms}_g(p)$. For SimHash, shortest-path, and GraphBPE, T_τ is a tokenizer-specific sparse feature map or object graph constructed directly from the same $\mathrm{atoms}(s_t)$, $\mathrm{atoms}_g(p)$, and O_p. Appendix A.2 gives the exact constructions.

3.2 Predictive Formulations

We compare two end-to-end predictive formulations over latent states. In the *state formulation*, the transition predictor outputs the next latent state directly, $\hat{z}_{t+1} = f_\theta(z_t, z_g, h_t)$, where θ denotes the predictor parameters, and h_t denotes the recurrent hidden state for the LSTM and is omitted for the stateless XGBoost predictor. In the *delta formulation*, the transition predictor outputs a latent update $\widehat{\Delta z}_t = f_\theta(z_t, z_g, h_t)$ and sets $\hat{z}_{t+1} = z_t + \widehat{\Delta z}_t$.

The two formulations differ in both target parameterization and predictor implementation. The LSTM uses masked cosine loss in the state formulation and masked mean-squared error in the delta formulation, while XGBoost is trained as a multi-output regressor on flattened pairs $[z_t; z_g]$ with targets z_{t+1} or $z_{t+1} - z_t$. We therefore analyze them as two end-to-end planning formulations.

3.3 Latent-Space Search

At test time, we run beam search rather than greedy successor matching over grounded symbolic successors of the current state. We use the Python planner `pyperplan` [Alkhazraji et al., 2020] as the successor generator, though any PDDL-compatible planner that enumerates applicable ground actions can serve this role. For a current symbolic state s, let $\mathcal{N}(s)$ denote the applicable successor set. Each successor $s' \in \mathcal{N}(s)$ is ranked by distance to the latent prediction:

$$\text{score}(s') = \begin{cases} 1 - \cos(\hat{z}_{t+1}, \phi_\tau(s', g_p, p)), & \text{state,} \\ \|\hat{z}_{t+1} - \phi_\tau(s', g_p, p)\|_2, & \text{delta.} \end{cases}$$

Beam search accumulates these scores along a partial plan and retains the k candidates with the lowest cumulative distance, with $k = 3$ in all reported runs; Appendix A.11 reports the full $k \in \{1, 3, 5\}$ grid, whose rankings are stable from $k = 3$ to $k = 5$. The search budget scales with object count as $H(p) = \max(100, 10|O_p|)$. Candidate plans are validated with VAL [Howey et al., 2004], so solved rate reflects executable plans that reach the goal rather than internal search termination alone.

3.4 Training Protocols

We compare two training protocols. In DOMAIN-DEPENDENT, each domain has its own fitted tokenizer and its own predictor. In ALL-DOMAINS, one tokenizer and one predictor are shared across all domains. For tokenizers that naturally admit a shared feature space, the pooled representation is learned by joint fitting. WL requires separate treatment because the underlying WL feature generator is domain-specific, so the pooled condition uses a domain-block union construction rather than a single shared vocabulary: if d indexes a domain and $\phi_{\tau,d}(s, g_p, p)$ is its local state embedding, then $\Phi_\tau^{\text{union}}(d, s, g_p, p) = [0, \ldots, 0, \phi_{\tau,d}(s, g_p, p), 0, \ldots, 0]$, with only the block for domain d populated. WL transfer results are therefore reported separately in the empirical analysis.

3.5 Tokenizer Families

All tokenizers operate on symbolic planning states, but they impose different invariances and feature spaces.

WL. WL tokenization applies Weisfeiler–Leman color refinement to the ILG $G_{p,s}^{\text{ilg}}$, where fact nodes are colored by predicate identity and edges record argument positions, and returns the resulting color histogram [Shervashidze et al., 2011]. We use two refinement iterations, no pruning, and multiset hashing. The goal vector is obtained by embedding a pseudo-state containing only goal atoms through the same feature generator.

SimHash. SimHash extracts a sparse symbolic feature vector $x(s, g_p, p)$ from predicate counts, state-goal overlap indicators, and structural counts, and then applies a seeded Gaussian random projection matrix R, giving $\phi_{\text{sim}}(s, g_p, p) = \mathbb{I}[x(s, g_p, p)^\top R \geq 0] \in \{0, 1\}^{128}$. This follows the locality-sensitive hashing perspective of [Charikar, 2002].

Shortest-path. Shortest-path tokenization builds separate state and goal object graphs and concatenates predicate counts, state-goal overlap indicators, and path-length histograms up to a fixed

radius, $\phi_{\mathrm{sp}}(s, g_p, p) = [c_{\mathrm{pred}}(s), c_{\mathrm{pred}}(g_p), c_{\mathrm{overlap}}(s, g_p), h_{\mathrm{sp}}(s), h_{\mathrm{sp}}(g_p)]$, which is closely related to shortest-path graph kernels [Borgwardt and Kriegel, 2005].

GraphBPE. GraphBPE constructs a joint state-goal object graph, repeatedly merges frequent adjacent label pairs, and embeds a state by the histogram of merged labels that remain after replaying the learned merge rules [Shen and Póczos, 2024].

Random. The random reference tokenizer uses deterministic seeded vectors. State embeddings are sums of random vectors attached to predicate names, while goal embeddings are sums of vectors attached to exact goal atoms. This provides an unstructured but reproducible point of comparison.

Appendix A.1 summarizes notation, and Appendices A.2 and A.4 give the full tokenizer constructions and inference details used in the reported study.

4 Experimental Design

4.1 Benchmarks

We evaluate on four classical planning domains written in PDDL: `blocks`, `gripper`, `logistics`, and `visitall-from-everywhere`. We adopt the benchmark collection of Fritzsche et al. [2026] unchanged, including its `train`, `validation`, `test-interpolation`, and `test-extrapolation` assignment, so that the comparison with SymT in Section 5 is exact. Interpolation instances have sizes within the training range, while extrapolation instances are strictly larger than every training instance in actual object counts (Table 1). Expert trajectories are the state sequences obtained by executing each instance's reference plan from the benchmark step by step, so a plan with T steps yields $T + 1$ states and no successor-selection ambiguity arises during training.

Table 1: Instance scale by split, measured in object counts, with mean extrapolation plan length. Extrapolation instances are strictly larger than all training instances in every domain, so the extrapolation split tests genuine size generalization rather than resampling of the training distribution.

Domain	Train objects	Interp. objects	Extrap. objects	Mean extrap. plan length
`blocks`	4–7	5	9–17	117.8
`gripper`	6–12	7–11	16–46	109.0
`logistics`	13–15	13–15	22–37	293.3
`visitall-from-everywhere`	1–16	2–15	21–121	460.6

The domains differ in relational structure and scaling pattern: `blocks` is a manipulation domain with stack relations, `gripper` is a transport domain with two rooms and two grippers, `logistics` combines truck and airplane transport across cities, and `visitall-from-everywhere` is a graph-traversal domain over large connectivity structures. Exact split counts and benchmark statistics are reported in Appendix A.5.

4.2 Predictor Families

Each tokenizer is paired with two transition predictors: an LSTM-based recurrent predictor and a tree-based multi-output regressor based on XGBoost. The LSTM retains temporal dependence across trajectory prefixes, while XGBoost treats each transition independently. Full predictor-construction details, optimization settings, hardware details, and runtime summaries are given in Appendix A.3 and Appendix A.17.

4.3 Held-Out Evaluation Protocol

For each tokenizer, predictor, formulation, and training protocol, we report two solved-rate summaries over held-out domain-split cells:

$$\mathrm{Macro}_8 = \frac{1}{2|\mathcal{D}|} \sum_{d \in \mathcal{D}} \sum_{s \in \{\texttt{interp},\texttt{extrap}\}} a_{d,s}, \tag{1}$$

$$\mathrm{Macro}_{\mathrm{ext}} = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} a_{d,\texttt{extrap}}, \tag{2}$$

where $a_{d,s}$ is the mean solved rate in domain d and split s. Both metrics weight every domain equally regardless of its problem count. We rank methods by Macro_8, which averages only over the held-out interpolation and extrapolation splits. We report $\mathrm{Macro}_{\mathrm{ext}}$ to isolate extrapolation. Validation is used only for model selection in both predictor families and is therefore excluded from all ranking metrics reported in the paper. All experiments are run with seeds `{13, 23, 37}`; reported means are seed-averaged, ribbons and error bars in the appendix figures show one standard deviation over seeds, and the confidence intervals in the comparison figures are nonparametric bootstrap intervals over held-out problems, which is the appropriate uncertainty for paired method comparisons. For paired statistical comparisons, we use held-out per-problem evaluation records: paired sign-flip tests on per-problem seed means and Holm correction within each comparison set. Appendix A.8 also reports an additive effect summary over the full held-out factorial study.

5 Results

5.1 Comparison with SymT on Extrapolation

SymT [Fritzsche et al., 2026] is the symmetry-aware transformer planner whose published benchmark, adopted unchanged in Section 4.1, defines our splits; its scores below are the published values, not a rerun (Appendix A.6). Table 2 compares the best SymT extrapolation score reported for each domain with the strongest single state-centric configuration in this study, DOMAIN-DEPENDENT + WL + XGBoost + delta. This is one fixed configuration used in every row, selected by the aggregate Macro_8 ranking rather than per domain. It exceeds the best reported SymT extrapolation score in three of the four domains; `logistics` remains unsolved by both approaches. At the aggregate level, it reaches 67.5% on $\mathrm{Macro}_{\mathrm{ext}}$ versus 35.8% for the strongest overall SymT variant. Appendix Table 10 records the domain-wise SymT values used in this comparison. Notably, SymT is a 25–35M parameter transformer while DOMAIN-DEPENDENT + WL + XGBoost + delta operates at 10^4–10^5 tree nodes (Appendix Table 11), showing that a compact state-centric configuration can outperform this transformer baseline; the controlled tokenizer comparisons below isolate representation effects within our pipeline.

Table 2: Extrapolation comparison against SymT on the shared benchmark. "Ours" uses the same configuration in every row: DOMAIN-DEPENDENT + WL + XGBoost + delta.

Domain	Best SymT (%)	Ours (%)	Improvement (pts)
`blocks`	13.0	70.0	+57.0
`gripper`	79.0	100.0	+21.0
`visitall-from-everywhere`	64.0	100.0	+36.0
`logistics`	0.0	0.0	0.0

5.2 Held-Out Tokenizer Ranking

Figure 2 reports all 40 configurations. The left panel gives Macro_8 and the right panel gives $\mathrm{Macro}_{\mathrm{ext}}$. WL attains the top held-out score in 7 of the 8 protocol-predictor-formulation combinations. The only exception is ALL-DOMAINS + XGBoost + state formulation, where GraphBPE reaches 48.2% on Macro_8, ahead of shortest-path at 43.5% and WL at 38.4%. The highest-scoring configuration is DOMAIN-DEPENDENT + WL + XGBoost + delta formulation at 74.0% on Macro_8 and 67.5% on $\mathrm{Macro}_{\mathrm{ext}}$. Averaged over the eight combinations, WL reaches 52.3% on Macro_8, followed by GraphBPE at 43.1%, shortest-path at 37.5%, random at 29.7%, and SimHash at 27.3%.

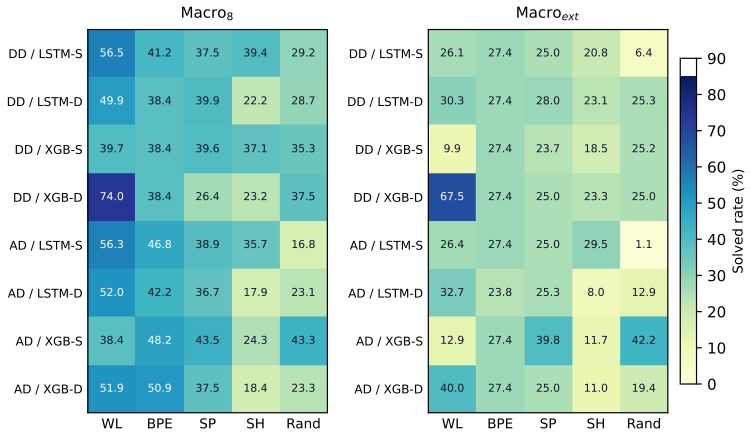

Figure 2: Held-out result matrix over all 40 tokenizer-predictor-formulation-protocol configurations. Left: Macro_8, averaged over the eight held-out domain-split cells. Right: $\mathrm{Macro}_{\mathrm{ext}}$, averaged over the four extrapolation cells. Entries are averaged over three seeds. Row labels abbreviate training protocol (DD: DOMAIN-DEPENDENT, AD: ALL-DOMAINS), predictor (LSTM, XGB: XGBoost), and formulation (S: state, D: delta); higher is better, and the best cell in each panel is DD/XGB-D under WL.

Appendix Table 12 lists the winners for these eight combinations under Macro_8. The largest margin occurs for DOMAIN-DEPENDENT + XGBoost + delta formulation, where WL exceeds the runner-up GraphBPE by 35.6 percentage points. Two combinations are close: DOMAIN-DEPENDENT + XGBoost + state formulation is nearly tied between WL and shortest-path, and ALL-DOMAINS + XGBoost + delta formulation separates WL and GraphBPE by only 1.0 point. Appendix Figure 8 reports the paired confidence intervals, and Appendix Table 15 reports the additive effect summary over the 120 seed-level held-out scores; the tokenizer effect is robust to removing the largest domain, rising from $\eta^2 = 0.575$ to 0.641 (Appendix A.14).

5.3 Extrapolation

The average held-out interpolation solved rate is 51.4% across the 40 configurations, whereas the average extrapolation solved rate is 24.5%. Across the eight protocol-predictor-formulation combinations, the interpolation winner and the extrapolation winner differ in 5 cases, and the mean Spearman correlation between interpolation and extrapolation tokenizer rankings is 0.43. The strongest reversal occurs for DOMAIN-DEPENDENT + XGBoost + state formulation: WL leads on interpolation at 69.4%, but its extrapolation score falls to 9.9%, allowing GraphBPE and shortest-path to overtake it. Figure 3 shows the interpolation-to-extrapolation shift directly. The left panel exhibits a positive relationship between interpolation and extrapolation, but many points lie far below the identity line. Appendix Figure 9 resolves the same reversal combination by combination.

Using paired sign-flip tests on held-out per-problem seed means, with Holm correction within each comparison set, the average combination exhibits 9.0 significant tokenizer contrasts on extrapolation but only 3.9 on interpolation. Because the extrapolation split contains 273 held-out problems versus 52 for interpolation (a 5.25:1 ratio), the larger count of significant contrasts partly reflects greater statistical power rather than effect size alone; the qualitative conclusion that tokenizer differences are larger on extrapolation therefore rests primarily on the 5-of-8 rank reversals and the mean Spearman correlation of 0.43 between interpolation and extrapolation tokenizer rankings. Appendix Table 16 reports the full counts, and Appendix Figure 8 shows the corresponding winner-versus-runner-up confidence intervals.

The right panel of Figure 3 resolves extrapolation by problem size for the strongest state-centric configuration, DOMAIN-DEPENDENT + XGBoost + delta formulation, showing all five tokenizers. In `blocks`, WL remains strong as size grows, whereas GraphBPE, SimHash, and the random tokenizer collapse on the larger instances and shortest-path stays near zero throughout. In `visitall-from-everywhere`, the pattern reverses: shortest-path, WL, and the random tokenizer

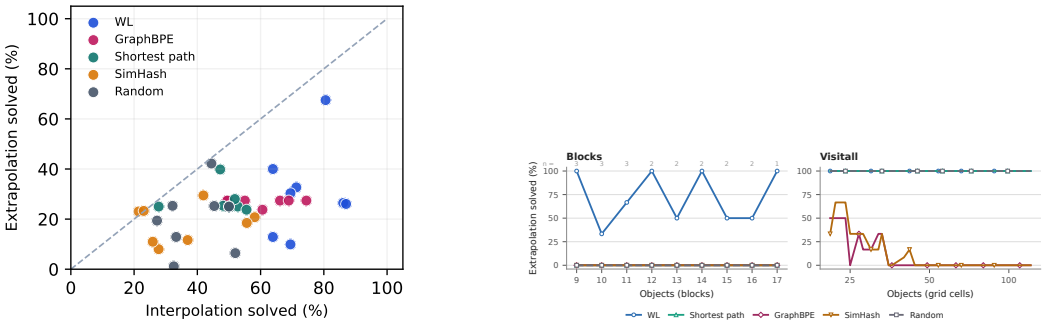

Figure 3: Held-out extrapolation diagnostics. Left: interpolation versus extrapolation solved rate across all 40 configurations, one point per configuration colored by tokenizer; points below the dashed identity line lose performance under extrapolation, and the vertical spread at fixed interpolation shows that extrapolation separates tokenizers more sharply. Right: size-resolved extrapolation solved rate for all five tokenizers under DOMAIN-DEPENDENT + XGBoost + delta in `blocks` and `visitall-from-everywhere`; the horizontal axis is the actual object count, the muted row above the `blocks` panel gives the number of benchmark instances per size, and no seed band is drawn because the seed variance of these XGBoost cells is exactly zero (Appendix A.7). WL degrades gracefully on `blocks` while every other tokenizer stays at zero, and the pattern reverses on `visitall-from-everywhere`, where shortest-path, WL, and Random (dashed) remain at 100% throughout.

remain near-perfect across the extrapolation range, while GraphBPE and SimHash remain far lower. We show `blocks` and `visitall-from-everywhere` in the main paper because they expose tokenizer-dependent size effects most clearly; `gripper` is nearly saturated in this setting and `logistics` is uniformly dominated by the search cap. Appendix Figures 13 and 14 report the full four-domain size curves under both training protocols and their protocol average.

5.4 Domain Specialization and Pooled Multi-Domain Training

Throughout this subsection, transfer refers to pooled multitask training across the four benchmark domains (ALL-DOMAINS), not generalization to domains unseen in training. Figure 4 reports held-out domain means. Each gray point corresponds to one protocol-predictor-formulation-seed setting, averaged over the interpolation and extrapolation problems of that domain; the colored point is the tokenizer mean over those 24 held-out settings. The domain pattern is highly specialized. WL is strongest on `blocks` with a held-out domain mean of 64.1%, while GraphBPE is strongest on `gripper` at 99.1%. Shortest-path is strongest on `visitall-from-everywhere` at 99.4%, well ahead of GraphBPE at 42.6% and also ahead of WL at 85.1%. `Logistics` remains the hardest held-out domain: even the strongest tokenizer, WL, averages 15.3%.

Pooled training helps some tokenizers but not others. Appendix Figure 10 and Appendix Table 17 compare DOMAIN-DEPENDENT and ALL-DOMAINS at matched tokenizer-predictor-formulation settings. Among tokenizers that use a shared pooled feature space in ALL-DOMAINS, the largest per-problem positive shift is the random reference tokenizer paired with XGBoost and the state formulation, at 56.6 per-problem percentage points (95% CI [50.8, 62.2]); however, this shift is disproportionately driven by the `visitall-from-everywhere` domain, which contributes 219 of the 273 extrapolation problems, and the corresponding domain-balanced Macro_8 gain for that configuration is 8.0 points. Among structured graph tokenizers, the largest statistically significant positive shift is shortest-path + XGBoost + state formulation at 4.0 per-problem percentage points (95% CI [0.9, 7.1]). GraphBPE shows smaller positive shifts. WL is reported separately because the all-domain setting uses the union representation defined in Section 3.4; under ALL-DOMAINS + WL + XGBoost + delta formulation, performance decreases by 6.2 paired points relative to DOMAIN-DEPENDENT.

Appendix Figure 10 makes two trends explicit: positive transfer occurs for GraphBPE and shortest-path under several settings and is largest for Random + XGBoost + state, while WL + XGBoost + delta and Random + XGBoost + delta move left of zero, indicating that pooling can wash out domain-specific structure that remains useful at test time.

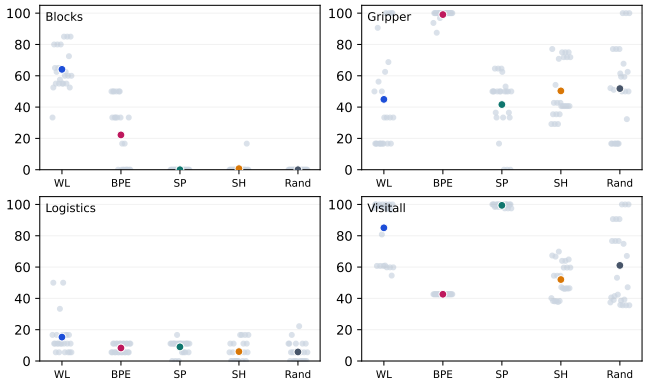

Figure 4: Held-out domain specialization. Each panel is one domain; gray points show the 24 held-out domain means for each tokenizer-domain pair (one per protocol-predictor-formulation-seed setting, averaged over that domain's interpolation and extrapolation problems), and colored markers show the tokenizer average. The leading tokenizer changes by domain: WL on `blocks`, GraphBPE on `gripper`, shortest-path on `visitall-from-everywhere`, and none reaching 20% on `logistics`.

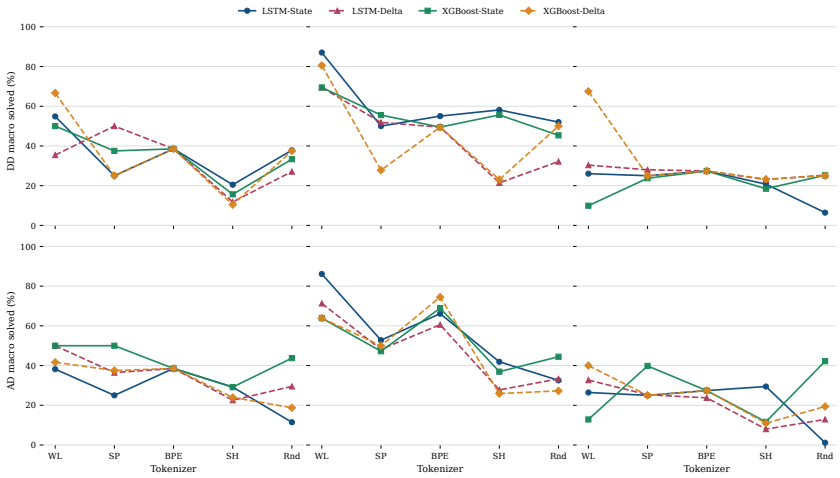

Figure 5: Split-specific predictor and formulation interactions. Top row: DOMAIN-DEPENDENT. Bottom row: ALL-DOMAINS. Left to right: validation, held-out interpolation, and held-out extrapolation. Lines correspond to LSTM-state, LSTM-delta, XGBoost-state, and XGBoost-delta.

The formulation comparison also depends on the tokenizer. WL benefits from the XGBoost delta formulation under DOMAIN-DEPENDENT. Under ALL-DOMAINS, the delta formulation continues to produce the two strongest XGBoost results: WL at 51.9% and GraphBPE at 50.9% on Macro_8. Under the ALL-DOMAINS state formulation, GraphBPE (48.2%) and shortest-path (43.5%) overtake WL (38.4%), reflecting the pattern visible in Figure 5.

5.5 Predictor and Formulation Interactions

Figure 5 resolves the full predictor-tokenizer interaction structure. The top row gives DOMAIN-DEPENDENT and the bottom row gives ALL-DOMAINS; within each row, the three panels report validation, held-out interpolation, and held-out extrapolation. Each line corresponds to one predictor-formulation pair. Averaged over all 40 configurations, XGBoost reaches 38.5% on Macro_8 and 26.5% on $\mathrm{Macro}_{\mathrm{ext}}$, compared with 37.5% and 22.6% for LSTM. The average predictor-family gap is therefore modest on Macro_8 but larger on extrapolation, where the tree-based predictor more often preserves tokenizer differences on the hardest held-out instances.

The strongest interaction in the study is WL with XGBoost. Under DOMAIN-DEPENDENT, switching from the state formulation to the delta formulation increases Macro_8 by 34.4 points and $\mathrm{Macro}_{\mathrm{ext}}$ by 57.6 points. The same change under ALL-DOMAINS remains positive, with gains of 13.6 points on Macro_8 and 27.1 points on $\mathrm{Macro}_{\mathrm{ext}}$. This pattern is not universal. Under XGBoost, shortest-path and SimHash are generally stronger in the state formulation, whereas GraphBPE is comparatively stable across the two formulations. The figure shows that these reversals are split-specific rather than uniform: for example, several tokenizer pairs are nearly tied on validation yet separate sharply on extrapolation.

The training protocol also changes which interactions are favorable: GraphBPE improves consistently under pooled training (12.5 points on Macro_8 for XGBoost-delta, 9.7 for XGBoost-state), shortest-path gains 16.1 points on $\mathrm{Macro}_{\mathrm{ext}}$ under XGBoost-state, whereas WL + XGBoost + delta drops by 22.1 points on Macro_8 and 27.5 on $\mathrm{Macro}_{\mathrm{ext}}$ when moving from DOMAIN-DEPENDENT to ALL-DOMAINS. These contrasts help explain why no single tokenizer dominates every combination, even though WL remains the strongest held-out tokenizer overall.

5.6 Search Saturation and Failure Modes

Appendix Table 18 and Figure 11 report search effort and failure modes. For the best state-centric configuration, DOMAIN-DEPENDENT + WL + XGBoost + delta, `visitall-from-everywhere` extrapolation is solved completely with mean budget use of only 11.4%, whereas `logistics` extrapolation consumes the full budget on every held-out instance and is never solved. `Blocks` lies between these extremes, with 70.0% solved, 30.0% cap hits, and 57.2% mean budget use on extrapolation.

Cap hits alone do not identify the cause of the `logistics` failure, so Appendix A.9 attributes it directly: one-step expert-successor ranking is at chance for all five tokenizers on `logistics` extrapolation, embedding collisions make the expert successor exactly indistinguishable from an alternative successor on 69–100% of decision steps, and a tenfold search budget leaves extrapolation at 0/18 for every tokenizer while raising WL interpolation from 22.2% to 66.7%. `Logistics` extrapolation is therefore representation-limited rather than merely budget-limited, consistent with known expressiveness limits on this domain [Ståhlberg et al., 2022, Martín and Geffner, 2004, Fern et al., 2006], whereas its interpolation split is partially budget-limited. The same appendix joins one-step prediction quality to search outcomes in every domain, identifying three distinct failure modes (at-chance prediction, recoverable noise where beam search is essential, and irreversible decisions), and Appendices A.12–A.13 add controls isolating goal conditioning and dimensionality, a stress-test domain with irreversible decisions, and sensitivity sweeps over WL refinement depth, shortest-path radius, beam width, and seeds.

6 Conclusion

Learning for generalized planning has recently advanced along both action-centric and state-centric lines. This paper shows that, within state-centric transition-model planning, tokenizer choice is a major determinant of held-out generalization. The contribution is not the qualitative point that representation matters but its controlled quantification: tokenizer choice explains 57% of held-out variance versus at most 1.2% for any other pipeline factor, leaders reverse between interpolation and extrapolation in 5 of 8 configuration families, and tree predictors of 10^4–10^5 nodes with the right representation exceed a 25–35M parameter transformer on 3 of 4 shared benchmark domains. WL is strongest overall, but no single tokenizer dominates, and the per-domain analysis with the goal-conditioning and dimensionality controls (Appendices A.9, A.12) identifies the structure of the feature space, rather than goal access or capacity, as the driver. We view these domain-level mechanisms as testable hypotheses; a predictive theory connecting representation expressiveness to domain transition structure is the natural next step, and establishing empirically that representation is first-order is its prerequisite. In future work, we plan to broaden domain coverage to PDDL domains with numeric fluents, temporal constraints, and larger object counts, to add learned feature-pool representations in the concept-language tradition as a sixth tokenizer family, and to evaluate these representations on downstream applications such as plan validity prediction and plan summarization, where tokenizer quality may matter independently of planning performance.

Acknowledgments and Disclosure of Funding

This work was supported by the National Science Foundation under Award No. 2454027. The authors declare no additional competing interests beyond the affiliations listed in this paper.

References

Y. Alkhazraji, M. Frorath, M. Grützner, M. Helmert, T. Liebetraut, R. Mattmüller, M. Ortlieb, J. Seipp, T. Springenberg, P. Stahl, et al. Pyperplan. *Zenodo*, 2020.

M. Asai and A. Fukunaga. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

M. Asai, H. Kajino, A. Fukunaga, and C. Muise. Classical planning in deep latent space. *arXiv preprint arXiv:2107.00110*, 2021.

B. Bonet and H. Geffner. Features, projections, and representation change for generalized planning. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 4667–4673, 2018.

B. Bonet, G. Francès, and H. Geffner. Learning features and abstract actions for computing generalized plans. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, pages 2703–2710, 2019.

K. M. Borgwardt and H.-P. Kriegel. Shortest-path kernels on graphs. In *Proceedings of the 5th IEEE International Conference on Data Mining*, pages 74–81, 2005.

M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 380–388, 2002.

D. Z. Chen and S. Thiébaux. Graph learning for numeric planning. *Advances in Neural Information Processing Systems*, 37:91156–91183, 2024.

D. Z. Chen, F. Trevizan, and S. Thiébaux. Graph neural networks and graph kernels for learning heuristics: Is there a difference? In *NeurIPS 2023 Workshop on Generalization in Planning*, 2023.

D. Z. Chen, S. Thiébaux, and F. Trevizan. Learning domain-independent heuristics for grounded and lifted planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20078–20086, 2024a.

D. Z. Chen, F. Trevizan, and S. Thiébaux. Return to tradition: Learning reliable heuristics with classical machine learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pages 68–76, 2024b.

D. Z. Chen, F. Trevizan, and S. Thiébaux. Return to tradition: Learning reliable heuristics with classical machine learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pages 68–76, 2024c.

P. Ferber, F. Geißer, F. Trevizan, M. Helmert, and J. Hoffmann. Neural network heuristic functions for classical planning: Bootstrapping and comparison to other methods. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, pages 583–587, 2022.

A. Fern, S. Yoon, and R. Givan. Approximate policy iteration with a policy language bias: Solving relational markov decision processes. *Journal of Artificial Intelligence Research*, 25:75–118, 2006.

G. Francès, B. Bonet, and H. Geffner. Learning general planning policies from small examples without supervision. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, pages 11801–11808, 2021.

M. Fritzsche, E. Gestrin, and J. Seipp. Symmetry-aware transformer training for automated planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 36236–36244, 2026.

S. Garg, A. Bajpai, and Mausam. Symbolic network: Generalized neural policies for relational MDPs. In *Proceedings of the 37th International Conference on Machine Learning*, pages 3397–3407, 2020.

M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.

E. Groshev, M. Goldstein, A. Tamar, S. Srivastava, and P. 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.

Z. Guo, E. Diao, C. Yang, and C. Shi. Graph tokenization for bridging graphs and transformers. In *The Fourteenth International Conference on Learning Representations*, 2026. URL `https://openreview.net/forum?id=jCctxI1BGF`.

N. Gupta, V. Pallagani, J. A. Aydin, and B. Srivastava. On sample-efficient generalized planning via learned transition models. *arXiv preprint arXiv:2602.23148*, 2026.

R. Howey, D. Long, and M. Fox. Val: Automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence*, pages 294–301, 2004.

R. Karia and S. Srivastava. Relational abstractions for generalized reinforcement learning on symbolic problems. In *Proceedings of the 31st International Joint Conference on Artificial Intelligence*, pages 3135–3142, 2022.

H. Kokel, A. Manoharan, S. Natarajan, B. Ravindran, and P. Tadepalli. RePReL: Integrating relational planning and reinforcement learning for effective abstraction. In *Proceedings of the 31st International Conference on Automated Planning and Scheduling*, pages 533–541, 2021.

M. Martín and H. Geffner. Learning generalized policies from planning examples using concept languages. *Applied Intelligence*, 20(1):9–19, 2004.

D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. Pddl-the planning domain definition language, 1998.

V. Pallagani, B. Muppasani, K. Murugesan, F. Rossi, L. Horesh, B. Srivastava, F. Fabiano, and A. Loreggia. Plansformer: Generating symbolic plans using transformers. *arXiv preprint arXiv:2212.08681*, 2022.

O. Rivlin, T. Hazan, and E. Karpas. Generalized planning with deep reinforcement learning. *arXiv preprint arXiv:2005.02305*, 2020.

N. Rossetti, M. Tummolo, A. E. Gerevini, L. Putelli, I. Serina, M. Chiari, and M. Olivato. Learning general policies for planning through GPT models. In *Proceedings of the 34th International Conference on Automated Planning and Scheduling*, volume 34, pages 500–508, 2024. doi: 10.1609/icaps.v34i1.31510.

W. Shen, F. Trevizan, and S. Thiébaux. Learning domain-independent planning heuristics with hypergraph networks. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling*, pages 574–584, 2020.

Y. Shen and B. Póczos. Graphbpe: Molecular graphs meet byte-pair encoding. *arXiv preprint arXiv:2407.19039*, 2024.

N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011.

T. Silver, S. Dan, K. Srinivas, J. B. Tenenbaum, L. P. Kaelbling, and M. Katz. Generalized planning in PDDL domains with pretrained large language models. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence*, pages 20256–20264, 2024.

S. Srivastava, N. Immerman, and S. Zilberstein. Foundations of generalized planning. Technical report, Technical Report UM-CS-2008-039, Univ. of Massachusetts, Amherst, 2008.

S. Ståhlberg, B. Bonet, and H. Geffner. Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling*, pages 629–637, 2022.

S. Toyer, F. Trevizan, S. Thiébaux, and L. Xie. Action schema networks: Generalised policies with deep learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

L. Wang, K. Hassani, S. Zhang, D. Fu, B. Yuan, W. Cong, Z. Hua, H. Wu, N. Yao, and B. Long. Learning graph quantized tokenizers. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=oYSsbY3G4o`.

A Supplementary Material

A.1 Notation Summary

Table 3 collects the notation used throughout the paper. Arguments that are fixed for a given problem instance (O_p, d, p) are suppressed after first use in each section.

Table 3: Notation summary.

Symbol	Meaning
d, p, s_0, g_p	Domain, problem instance, initial state, goal condition
$\mathrm{atoms}(s)$	Grounded atom set of symbolic state s
$\mathrm{atoms}_g(p)$	Grounded goal-atom set induced by g_p
O_p	Problem objects together with domain constants
$\xi_p = (s_0, \ldots, s_T)$	Expert state trajectory from executing the reference plan
$I(p,t)$	Symbolic tokenizer input $(\mathrm{atoms}(s_t), \mathrm{atoms}_g(p), O_p, d, p)$
$r_{\tau,t} = T_\tau(I(p,t))$	Intermediate representation of tokenizer family τ
$z_t, z_g \in \mathbb{R}^{m_\tau}$	State and goal embeddings; m_τ is the family's output dimension
$\phi_\tau(s, g_p, p)$	State-encoder embedding of an arbitrary state s
$\psi_\tau(g_p, p)$	Goal-encoder embedding
$\hat{z}_{t+1}, \widehat{\Delta z}_t$	Predicted next latent state (state form) and latent update (delta form)
$\mathcal{N}(s)$	Applicable successor set of s
$H(p) = \max(100, 10\lvert O_p\rvert)$	Search-step budget
$\mathrm{Macro}_8, \mathrm{Macro}_{\mathrm{ext}}$	Domain-balanced held-out and extrapolation-only solved-rate macros

A.2 Tokenizer Formulations and Representation Statistics

This appendix records the tokenizer constructions used in the study and the corresponding representation statistics. We give the fitted tokenizers, feature maps, and transformation rules explicitly.

Following Section 3, let $\mathrm{atoms}(s)$ denote the grounded atom set of any symbolic state s, let $\mathrm{atoms}_g(p)$ denote the grounded goal-atom set induced by goal condition g_p, and let O_p be the set of problem objects together with domain constants. After lowercasing and parenthesis stripping, each atom is written as $(r, o_1, \ldots, o_k)$ with predicate symbol $\mathrm{pred}(a) = r$ and argument tuple $\mathrm{args}(a) = (o_1, \ldots, o_k)$. For every fitted tokenizer family $\tau \in \{\mathrm{WL}, \mathrm{SimHash}, \mathrm{SP}, \mathrm{GraphBPE}\}$, training produces a fitted tokenizer

$$\mathcal{F}_\tau^{\mathrm{DD},d} = \mathrm{Fit}_\tau\big(\mathcal{X}_d^{\mathrm{train}}\big), \qquad \mathcal{F}_\tau^{\mathrm{AD}} = \mathrm{Fit}_\tau\big(\cup_{d\in\mathcal{D}} \mathcal{X}_d^{\mathrm{train}}\big), \tag{3}$$

where $\mathcal{X}_d^{\mathrm{train}}$ is the set of tokenizable trajectory states from domain d. Given the fitted tokenizer, the state and goal encoders are

$$z_t = E_\tau^{\mathrm{state}}\big(\mathrm{atoms}(s_t), \mathrm{atoms}_g(p), O_p; \mathcal{F}_\tau\big), \qquad z_g = E_\tau^{\mathrm{goal}}\big(\mathrm{atoms}_g(p), O_p; \mathcal{F}_\tau\big). \tag{4}$$

All tokenizer families consume the same symbolic input tuple $(\mathrm{atoms}(s), \mathrm{atoms}_g(p), O_p, d, p)$, but they do not construct the same intermediate object. WL is the only tokenizer that delegates graph construction to the `wlplan` instance-learning-graph backend. SimHash builds a sparse symbolic feature dictionary directly from atoms and object counts. Shortest-path builds separate state and goal object graphs from those same atoms. GraphBPE builds one joint state-goal object graph with learned merge rules. Only WL departs from the shared-fitting pattern above, because its AD setting uses the domain-block union representation from Section 3.4 rather than one genuinely shared vocabulary.

For WL in the all-domain setting, the code fits one domain-specific WL tokenizer per domain and stores domain dimensions m_d together with cumulative offsets o_d. If $u_d \in \mathbb{R}^{m_d}$ is the active domain's local WL embedding, the global union vector is

$$\Phi_{\mathrm{WL}}^{\mathrm{union}}(d, s, g_p, p)[j] = \begin{cases} (u_d)_{j-o_d}, & o_d \le j < o_d + m_d, \\ 0, & \text{otherwise}, \end{cases} \tag{5}$$

so inference always populates exactly one domain block and leaves the remaining coordinates at zero.

WL tokenization. The reported WL implementation does not reimplement graph construction. It calls the `wlplan` feature generator with `feature_algorithm=wl`, `graph_representation=ilg`, `iterations=2`, `pruning=none`, and `multiset_hash=True`, and delegates ILG construction to that backend. For a parsed problem-state pair (p, s), the backend instantiates a directed typed incidence graph

$$G^{\mathrm{ilg}}_{p,s} = (V, E, \lambda_0), \qquad V = O_p \cup F_s, \qquad F_s = \{\nu_a : a \in \mathrm{atoms}(s)\}, \tag{6}$$

where each object in O_p is a node and each grounded state atom contributes a fact node ν_a. If $a = (r, o_1, \ldots, o_k)$, the initial node colors satisfy $\lambda_0(\nu_a) = c_r$ for a predicate-specific color c_r, while object nodes receive the shared object color used by the backend. The edge set records predicate argument incidence in both directions,

$$E = \big\{(o_i, \nu_a, i), (\nu_a, o_i, i) : a = (r, o_1, \ldots, o_k) \in \mathrm{atoms}(s),\ i = 1, \ldots, k\big\}, \tag{7}$$

so the ILG is a bipartite graph between objects and grounded facts with typed edge labels given by argument position. The `fit` stage parses every training trajectory line into a `wlplan` `State`, wraps it inside `ProblemDataset`/`DomainDataset`, and calls `collect` once over the full training corpus. Two rounds of color refinement are then applied,

$$\lambda_t(v) = \mathrm{hash}\Big(\lambda_{t-1}(v), \{\{(\lambda_{t-1}(u),\, \ell_{vu}) : (v, u, \ell_{vu}) \in E\}\}\Big), \qquad t \in \{1, 2\}, \tag{8}$$

where ℓ_{vu} is the argument-position label on the directed edge $(v, u) \in E$, so the multiset explicitly distinguishes neighbors reached by different argument positions and the refinement is not invariant to argument permutations. Multiset hashing is enabled and pruning is disabled. If $\mathcal{C}_{\mathrm{WL}}$ denotes the color vocabulary collected from all training states seen during `fit`, the embedding is the resulting histogram

$$\phi_{\mathrm{WL}}(s, g_p, p)[c] = \sum_{v \in V} \mathbb{I}[\lambda_2(v) = c], \qquad c \in \mathcal{C}_{\mathrm{WL}}. \tag{9}$$

At inference time, the code parses the problem PDDL again, converts the current symbolic atoms to `wlplan` atoms, forms a one-state `ProblemDataset`, and calls `embed`. The goal encoder uses the same embedding pipeline after converting the goal atom set into a pseudo-state s_g with atom set $\mathrm{atoms}_g(p)$, which induces the goal ILG with fact-node set $F_g = \{\nu_a : a \in \mathrm{atoms}_g(p)\}$, so

$$\psi_{\mathrm{WL}}(g_p, p) = \phi_{\mathrm{WL}}(s_g, g_p, p). \tag{10}$$

Thus, the WL state channel is an ILG over current-state atoms only, while goal information enters through a separate goal pseudo-state embedding. In the notation above, $\phi_{\mathrm{WL}}(s, g_p, p)$ does not depend on g_p explicitly; the goal enters only through $\psi_{\mathrm{WL}}(g_p, p)$. WL is the only tokenizer here whose state channel comes from an external ILG backend rather than from a graph or feature map defined directly in this repository.

SimHash tokenization. SimHash first extracts a sparse symbolic feature vector $x_{\mathrm{sim}}(s, g_p, p) \in \mathbb{R}^{m_{\mathrm{sim}}}$ over a dictionary K_{sim} collected from the training corpus. The implementation includes state predicate counts, predicate arity counts, binary-edge counts, self-reference indicators, goal predicate and goal arity counts, state-goal overlap keys, goal-missing keys, object and atom counts, and the scalar density feature $|\,\mathrm{atoms}(s)|/|O_p|$. With seeded Gaussian projection matrix $R \in \mathbb{R}^{m_{\mathrm{sim}} \times 128}$,

$$\phi_{\mathrm{sim}}(s, g_p, p) = \mathbb{I}\big[x_{\mathrm{sim}}(s, g_p, p)^\top R \geq 0\big] \in \{0, 1\}^{128}. \tag{11}$$

The projection matrix is sampled once during `fit` from $\mathcal{N}(0, 1)$ using the reported seed and then reused for every transform call. The goal encoder is $\psi_{\mathrm{sim}}(g_p, p) = \phi_{\mathrm{sim}}(\varnothing, g_p, p)$.

Shortest-path tokenization. Shortest-path tokenization constructs undirected object graphs from state and goal atoms separately. The state graph is $G_s = (O_p, E_s)$ with

$$\{u, v\} \in E_s \iff \exists a \in \mathrm{atoms}(s) \text{ such that } |\,\mathrm{args}(a)| = 2 \text{ and } \{u, v\} \subseteq \mathrm{args}(a), \tag{12}$$

and the goal graph G_g is defined analogously from $\mathrm{atoms}_g(p)$. Unary predicates never create graph edges; they only contribute count features. Predicates with arity three or higher are likewise excluded from the edge set because the edge condition requires exactly two distinct arguments; in PDDL domains that contain ternary or higher-arity predicates, those predicates contribute only to

the count features and not to the path histograms. All four benchmark domains used in this study have predicate arity at most two in their relational structure, so this restriction does not affect the reported results. For path radius $L = 5$, the path histograms are

$$h_s^{(\ell)} = \sum_{\{u,v\} \subseteq O_p} \mathbb{I}[\mathrm{dist}_{G_s}(u,v) = \ell]\,, \qquad h_g^{(\ell)} = \sum_{\{u,v\} \subseteq O_p} \mathbb{I}\big[\mathrm{dist}_{G_g}(u,v) = \ell\big]\,, \tag{13}$$

for $\ell = 1, \ldots, L$. The full embedding concatenates state predicate and arity counts, goal predicate and arity counts, overlap and missing-goal indicators, path histograms $\{h_s^{(\ell)}\}_{\ell=1}^{L}$ and $\{h_g^{(\ell)}\}_{\ell=1}^{L}$, and the structural scalars $|O_p|$, $|\,\mathrm{atoms}(s)|$, and $|\,\mathrm{atoms}_g(p)|$. During `fit`, the implementation collects the union of all observed feature keys over the training corpus and forcibly retains the path bins for every $\ell \leq 5$, ensuring a stable dense feature order at inference time. As with SimHash, the goal encoder is the empty-state specialization $\psi_{\mathrm{sp}}(g_p, p) = \phi_{\mathrm{sp}}(\varnothing, g_p, p)$.

GraphBPE tokenization. GraphBPE begins from an object graph $G_{p,s,g_p} = (O_p, E, \lambda_0)$ whose adjacency is induced by co-occurrence inside state or goal atoms,

$$\{u,v\} \in E \iff \exists a \in \mathrm{atoms}(s) \cup \mathrm{atoms}_g(p) \text{ with } u \neq v \text{ and } \{u,v\} \subseteq \mathrm{args}(a). \tag{14}$$

The initial object label records which predicates mention each object in the current state and in the goal:

$$\lambda_0(o) = \mathrm{join}\Big(\mathrm{sort}\big(S_o \cup G_o\big)\Big), \tag{15}$$

where

$$\begin{aligned} S_o &= \{\texttt{s:}\,\mathrm{pred}(a) : o \in \mathrm{args}(a),\ a \in \mathrm{atoms}(s)\}, \\ G_o &= \{\texttt{g:}\,\mathrm{pred}(a) : o \in \mathrm{args}(a),\ a \in \mathrm{atoms}_g(p)\}, \end{aligned} \tag{16}$$

with the reserved label `EMPTY` used when the set is empty. At BPE iteration r, the implementation counts unordered adjacent label pairs

$$f_r(\alpha, \beta) = \sum_G \#\Big\{\{u,v\} \in E_G : \{\lambda_r(u), \lambda_r(v)\} = \{\alpha, \beta\}\Big\}, \tag{17}$$

selects the most frequent pair (α_r, β_r), forms merged label $m_r = (\alpha_r + \beta_r)$, and greedily contracts matching edges in every training graph. When multiple edges in the same training graph match the selected pair, the implementation processes them sequentially in node-index order; each contraction immediately updates the labels of the contracted node, so later matches within the same round reflect the already-merged labels. This means each node participates in at most one contraction per round and the resulting intermediate graph is deterministic given the node ordering. The code runs at most 100 merge rounds and stops earlier if no pair occurs more than once or the target vocabulary size of 1000 is reached. At inference time, the learned merge list is replayed deterministically on a fresh graph and the embedding is the final label histogram,

$$\phi_{\mathrm{bpe}}(s, g_p, p)[c] = \sum_{o \in O'_p} \mathbb{I}[\lambda_R(o) = c]\,, \qquad c \in \mathcal{V}_{\mathrm{bpe}}, \tag{18}$$

where O'_p is the contracted node set after replaying the R retained merges. The goal encoder again uses the empty-state specialization $\psi_{\mathrm{bpe}}(g_p, p) = \phi_{\mathrm{bpe}}(\varnothing, g_p, p)$.

Random tokenization. The random reference tokenizer has no learned parameters beyond the fixed global seed. For any token string ω, it defines a deterministic random vector $\eta(\omega) \in \mathbb{R}^{128}$ by seeding a standard-normal generator with the SHA256 digest of ω XORed with the global seed. The state encoder discards object identities and sums predicate-name vectors only,

$$\phi_{\mathrm{rand}}(s, g_p, p) = \mathrm{norm}\left(\sum_{a \in \mathrm{atoms}(s)} \eta\big(\texttt{state_pred:}\,\mathrm{pred}(a)\big)\right), \tag{19}$$

while the goal encoder preserves grounded goal atoms,

$$\psi_{\mathrm{rand}}(g_p, p) = \mathrm{norm}\left(\sum_{a \in \mathrm{atoms}_g(p)} \eta\big(\texttt{goal:}\,a\big)\right). \tag{20}$$

Table 4: Tokenizer implementation summary. "Fitted object" denotes the data structure produced by `fit`; "goal path" denotes how the goal vector is constructed at inference time.

Tokenizer	Fitted object	State path	Goal path	Output type
WL	`wlplan` feature generator and collected color vocabulary	Build an object-fact ILG, run two WL iterations, and histogram the resulting colors	Apply the same ILG construction to goal atoms only and embed the resulting pseudo-state	Sparse nonnegative count vector
SimHash	Sparse feature-key dictionary plus seeded Gaussian projection matrix	Extract symbolic count features, then apply random projection and thresholding	Reuse the same feature extractor with the empty state	Binary 128-vector
Shortest-path	Sorted feature-key dictionary for predicate and path features	Build state and goal object graphs, then emit path and count histograms	Reuse the same transform with the empty state	Sparse nonnegative count vector
GraphBPE	Learned merge list plus final token vocabulary	Build an object graph, replay greedy graph merges, and histogram the final labels	Reuse the same transform with the empty state	Sparse nonnegative count vector
Random	Fixed seed only	Sum deterministic random vectors over state predicate-name tokens and normalize	Sum deterministic random vectors over grounded goal-atom tokens and normalize	Dense real-valued 128-vector

Table 5: Tokenizer representation dimensions derived from the fitted vocabularies and feature spaces.

Tokenizer	DD dim range	DD mean dim	AD dim	AD construction
WL	138-552	291.0	1164	union
GraphBPE	58-104	87.0	123	pooled
Shortest path	27-32	29.2	55	pooled
SimHash	128-128	128.0	128	pooled
Random	128-128	128.0	128	pooled

Here $\mathrm{norm}(v) = v/\|v\|_2$ when $v \neq 0$ and 0 otherwise. As with WL, $\phi_{\mathrm{rand}}(s, g_p, p)$ ignores g_p in the state channel and uses $\psi_{\mathrm{rand}}(g_p, p)$ for explicit goal conditioning. This construction preserves explicit goal conditioning while removing nearly all object-level structure from the state channel. Because the state encoder sums one vector per predicate occurrence (indexed by predicate name, not by grounded atom), the resulting embedding is a random linear projection of the state's predicate-count vector. In domains where task progress correlates tightly with a single predicate count (for example, the accumulation of `visited` atoms in `visitall-from-everywhere` or the reduction of `clear` atoms in `blocks`), this encoding can serve as a coarse macroscopic progress indicator despite its lack of relational structure, which partially explains the strong extrapolation performance of the random reference tokenizer on `visitall-from-everywhere`.

Table 4 summarizes the main implementation differences among the tokenizer families. Tables 5 and 6 record the representation dimensions and construction settings used in the study. Table 5 shows that the DD setting yields domain-specific fitted tokenizers for every tokenizer, whereas the AD setting uses a pooled feature space for all tokenizers except WL, whose pooled representation is the union construction. Table 6 lists the exact settings used in the reported study, including WL refinement depth, GraphBPE merge budget and vocabulary size, shortest-path radius, and the 128-dimensional SimHash and random constructions. Figure 6 compares pooled representation dimension with held-out performance.

A.3 Predictor Implementation and Optimization

The two predictor families consume the same tokenized trajectories but differ in optimization behavior and functional form.

LSTM transition predictor. The recurrent predictor is trained on variable-length tokenized trajectories from the `train` split, with `validation` used only for model selection. Each batch is padded to the longest trajectory in the batch, packed before entering the recurrent core, and masked so that padding positions do not contribute to the loss. Trajectories of length one are discarded because they do not form a transition pair. When projection is enabled, both z_t and z_g are linearly projected to 32 dimensions and passed through ReLU before concatenation. The recurrent core is a two-layer LSTM with hidden size 256.

Table 6: Tokenizer construction settings and pooled-representation descriptions used in the reported study.

Tokenizer	Key settings	DD dim range	AD dim	Pooled representation
WL	WL color refinement, iterations=2	138-552	1164	Union block representation
GraphBPE	GraphBPE merges=100, vocab_size=1000	58-104	123	Shared pooled vocabulary
Shortest path	Shortest-path histograms, max_path_length=5	27-32	55	Shared pooled histogram
SimHash	Gaussian SimHash, hash_dim=128, seed=42	128-128	128	Shared pooled projection
Random	Deterministic random vectors, dim=128, seed=42	128-128	128	Shared pooled reference space

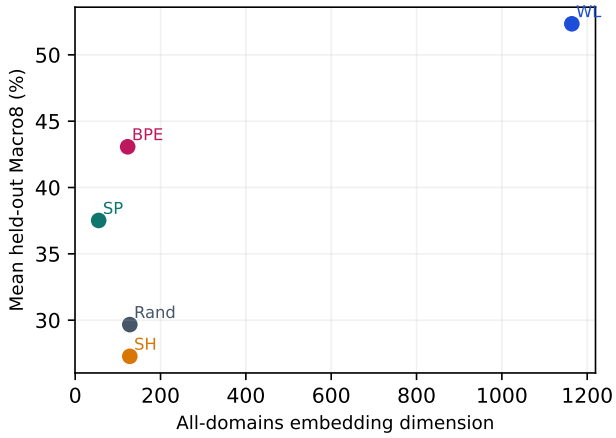

Figure 6: All-domains representation dimension versus mean held-out Macro_8. The figure shows how pooled dimensionality varies across tokenizers and protocols.

The state formulation uses a two-layer output head with LayerNorm and ReLU and minimizes

$$\mathcal{L}_{\text{state}} = 1 - \frac{1}{|\Omega|} \sum_{(b,t)\in\Omega} \cos\big(\hat{z}_{t+1}^{(b)}, z_{t+1}^{(b)}\big), \tag{21}$$

where Ω denotes the valid masked timesteps. The delta formulation replaces the nonlinear head with a linear map back to the tokenizer dimension and minimizes

$$\mathcal{L}_{\text{delta}} = \frac{1}{|\Omega|} \sum_{(b,t)\in\Omega} \left\| \widehat{\Delta z}_t^{(b)} - \big(z_{t+1}^{(b)} - z_t^{(b)}\big) \right\|_2^2. \tag{22}$$

All reported LSTM runs use Adam with learning rate 10^{-2}, batch size 32, and 500 epochs. Automatic mixed precision and pinned-memory data loading are enabled on CUDA, and the checkpoint with the lowest validation loss is used for inference.

XGBoost transition predictor. The tree-based predictor flattens each tokenized trajectory into independent transition pairs. For each timestep, the input row is $[z_t; z_g]$ and the target is either z_{t+1} or $z_{t+1} - z_t$, matching the two planning formulations. The implementation uses `XGBRegressor` with squared-error objective, histogram-based tree construction, CUDA execution, and multi-output regression over the tokenizer dimension. All reported runs use 1000 boosting rounds, maximum depth 8, learning rate 0.1, 8 training threads, and early stopping with patience 10 on validation transitions. For the state formulation, the XGBoost predictor is trained with a squared-error objective but successors are ranked at inference time using cosine distance (Section 3.3). Because WL, shortest-path, and GraphBPE produce unnormalized count histograms, minimizing squared error during training does not strictly align with minimizing cosine distance during inference. This training-inference objective mismatch is a pragmatic consequence of standard multi-output tree regressors not supporting a cosine objective; the delta formulation avoids this issue since both training (squared error on $z_{t+1} - z_t$) and inference (ℓ_2 distance on the predicted update) use the same ℓ_2 geometry.

At inference time, both predictors are used within the same symbolic beam search. The beam width is fixed to 3, successor ranking is deterministic up to lexical tie-breaking over plans, and the search horizon scales as $H(p) = \max(100, 10|O_p|)$.

Table 7: Predictor hyperparameters used in all reported runs.

Predictor	Formulation	Hidden / Depth	Projection	Epochs / Trees	Batch	Learning rate
LSTM	state	2 layers, 256 hidden	32-dim	500 epochs	32	10^{-2}
LSTM	delta	2 layers, 256 hidden	32-dim	500 epochs	32	10^{-2}
XGBoost	state	depth 8	n/a	1000 trees	n/a	10^{-1}
XGBoost	delta	depth 8	n/a	1000 trees	n/a	10^{-1}

A.4 Planning and Validation Procedure

All reported runs use the same symbolic search procedure once the tokenizer and predictor are fixed. For a problem instance p, the planner starts from the initial symbolic state s_0, embeds the current state and goal, predicts the next latent state or latent update, scores every grounded symbolic successor generated by `pyperplan`, and keeps the top three partial plans under cumulative score. This loop repeats until a goal state is reached or the budget $H(p)$ is exhausted.

More explicitly, at search step t the planner computes (z_t, z_g), applies the selected predictor to obtain either $\hat{z}_{t+1}$ or $\widehat{\Delta z}_t$, converts the delta output to $\hat{z}_{t+1}$ when needed, and evaluates each successor $s' \in \mathcal{N}(s_t)$ with the scoring rule from Section 3.3. Search expands only applicable ground actions from the `pyperplan` grounding, so every successor is precondition-checked, and the goal test is performed internally during search. The cumulative score of a partial plan is the sum of step scores along that plan. Ties between equal-score partial plans are broken lexically so that inference is deterministic.

For the recurrent predictor, a separate LSTM hidden state is maintained for every beam branch: each partial plan is a tuple of cumulative score, hidden state, current state embedding, current symbolic state, action sequence, and a visited-state set. When a branch is expanded, the LSTM consumes that branch's current state embedding and the goal embedding under its own hidden state, and the updated hidden state is inherited by the branch's children, so the recurrent state encodes exactly the state sequence along that path. Cycles are prevented per branch: any successor whose symbolic state already appears in the branch's visited set is skipped. Branches reaching the same symbolic state through different paths are kept as distinct beam entries with their own scores and hidden states.

The `pyperplan` successor model and goal test are sound, so validation is in principle redundant; VAL serves as an independent end-to-end audit of reported solved rates, and across all 42,960 problem evaluations there are zero disagreements between search outcomes and validator verdicts (Table 19).

A.5 Extended Benchmark Statistics

Table 8 summarizes the benchmark composition. The benchmark is heterogeneous in problem counts, transition structure, action vocabulary, and the way size increases from interpolation to extrapolation.

Blocks. The `blocks` domain contains 5 predicates and 4 operators. Every transition changes only a few local relations, but support constraints make the global configuration space strongly coupled: changing one stack can create or remove `clear`, `holding`, and `on` relations that constrain many future moves.

Gripper. The `gripper` domain contains 7 predicates and 3 operators. Its action vocabulary is small and regular, but plans are often long because each object must be moved through a narrow transport bottleneck with two grippers and two rooms.

Logistics. The `logistics` domain contains 9 predicates and 6 operators spanning packages, trucks, airplanes, locations, and cities. Successful plans require coordinating intra-city truck motion with inter-city air transport while preserving containment relations.

Visitall-from-everywhere. The `visitall-from-everywhere` domain contains 4 predicates and a single move operator over a connectivity graph. Its operator algebra is simple, but the extrapolation split scales to much larger grids and therefore much larger search horizons than the other domains.

Table 8: Full benchmark composition.

Domain	Train	Validation	Interp.	Extrap.
`blocks`	9	3	3	20
`gripper`	4	2	3	16
`logistics`	12	4	9	18
`visitall-from-everywhere`	207	24	37	219

The splits are inherited unchanged from the benchmark of Fritzsche et al. [2026], including the train/validation/interpolation/extrapolation assignment. A size variable parsed from instance identifiers would overlap numerically across splits, because the interpolation and extrapolation instances follow different naming schemes (for `gripper`, interpolation instances are named by ball count while extrapolation instances carry historical indices 5–20 that correspond to 12–42 balls), even though the actual instances do not overlap. Table 9 therefore reports actual object counts: extrapolation instances are strictly larger than every training instance in all four domains.

Table 9: PDDL characteristics and within-domain instance scale, measured in object counts.

Domain	Pred.	Act.	Structural profile	Train obj.	Interp. obj.	Extrap. obj.
`blocks`	5	4	Stack and unstack manipulation with support constraints and long dependency chains.	4–7	5	9–17
`gripper`	7	3	Two-room transport with repeated pick–move–drop motifs and a narrow carrying bottleneck.	6–12	7–11	16–46
`logistics`	9	6	Multi-city multimodal package transport with truck and airplane coordination.	13–15	13–15	22–37
`visitall-from-everywhere`	4	1	Grid-coverage traversal on a connectivity graph with local moves only.	1–16	2–15	21–121

Table 9 helps interpret the domain reversals in the main paper. Shortest-path tokenization is particularly well aligned with `visitall-from-everywhere`, where the transition structure is close to the connectivity graph itself. WL is more favorable on `blocks` and `logistics`, where higher-order relational structure is not reducible to local path statistics alone. This provides a benchmark-grounded interpretation of the observed tokenizer specialization.

The benchmark statistics also show large differences in trajectory length and scale. Mean extrapolation plan length is 117.8 for `blocks`, 109.0 for `gripper`, 293.3 for `logistics`, and 460.6 for `visitall-from-everywhere`. The per-size composition is uneven: `gripper` has exactly one extrapolation instance per size value (16 instances over 16 sizes), and `blocks` has one to three instances per size, which matters when interpreting the size-resolved figures below.

Figure 7, together with Tables 8 and 9, shows that `visitall-from-everywhere` dominates the benchmark numerically while also contributing the widest scale range and the largest extrapolation horizons. For this reason, the empirical analysis resolves behavior by domain and split instead of relying on one pooled benchmark average.

A.6 Published SymT Comparison

The main paper compares the tokenizer study against SymT, the strongest action-centric planner considered in the companion benchmark summary. Table 10 records the SymT extrapolation values used in the main-paper comparison. The domain rows report the SymT encoder and encoder–decoder variants, and "Best SymT" selects the highest extrapolation score for that domain. The aggregate row reports the strongest overall SymT extrapolation summary on $\mathrm{Macro}_{\mathrm{ext}}$. These numbers are used only as contextual comparison; SymT is not rerun in the present work.

Table 11 records the corresponding predictor-scale context. For SymT, we retain the architecture-based parameter estimate derived from the published model specification. For the state-centric pre-

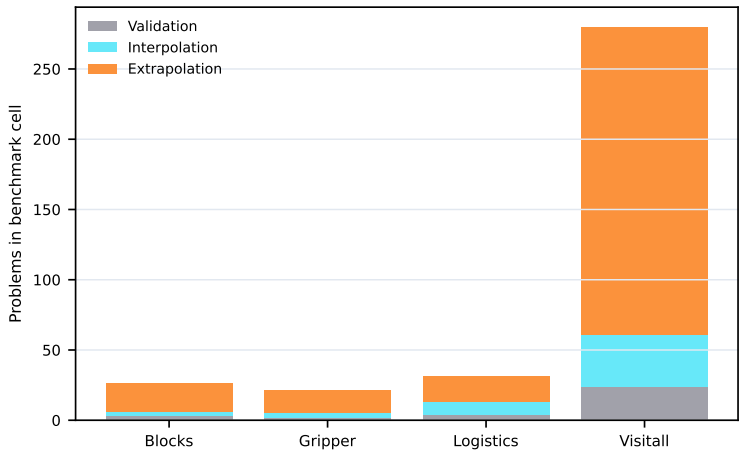

Figure 7: Benchmark composition by domain and split.

Table 10: SymT extrapolation values used in the main-paper comparison. Domain rows report the strongest SymT mean among the published encoder and encoder–decoder variants summarized in the action-centric comparison source. The aggregate row reports the strongest published SymT value on $\mathrm{Macro}_{\mathrm{ext}}$.

Domain / metric	Enc.	Enc–dec G	Enc–dec A	Enc–dec R	Best SymT
`blocks`	5.0	7.0	13.0	0.0	13.0
`gripper`	2.0	0.0	15.0	79.0	79.0
`visitall`	42.0	0.0	15.0	64.0	64.0
`logistics`	0.0	0.0	0.0	0.0	0.0
$\mathrm{Macro}_{\mathrm{ext}}{}^{\dagger}$	12.2	1.8	10.8	35.8	35.8

† The aggregate $\mathrm{Macro}_{\mathrm{ext}}$ row reports the single strongest SymT model variant's $\mathrm{Macro}_{\mathrm{ext}}$ score (Enc–dec R at 35.8%), not the average of the per-domain "Best SymT" column entries (which would be $(13.0 + 79.0 + 64.0 + 0.0)/4 = 39.0\%$). The 35.8% figure is used throughout the paper for aggregate comparison because it corresponds to a single model evaluated jointly across all domains.

dictors in the present paper, the LSTM counts come directly from the training logs, and the XGBoost complexity is reported as total tree nodes, which is the quantity logged by the training code and used throughout this work as a predictor-size proxy. The WL-based recurrent predictors used in the paper fall between 972,108 and 1,015,624 trainable parameters across the verified runs. For the strongest DOMAIN-DEPENDENT + WL + XGBoost + delta configuration, the verified per-domain node counts are 29,302 on `gripper`, 47,810 on `blocks`, and 139,046 on `logistics`; these values remain far below the scale of SymT.

Table 11: Representative predictor scales for SymT and the strongest state-centric configurations in this paper. SymT is an architecture-based estimate; LSTM counts and XGBoost node counts are taken from the reported training logs.

System	Scale	Type
SymT (enc–dec)	∼25–35M params	Transformer
WL + LSTM	0.97–1.02M params	2-layer recurrent predictor
WL + XGBoost + delta	29K–139K nodes	Gradient-boosted trees

A.7 Statistical Procedures

The main paper reports domain-balanced rankings and paired statistical comparisons. The rankings use domain-split means. The pairwise comparisons use held-out per-problem outcomes.

Table 12: Held-out winners under Macro8 for the eight protocol-predictor-formulation combinations, together with the Macro8 margin over the runner-up tokenizer. Appendix figures report paired confidence intervals and Holm-corrected p-values.

Family	Winner	Macro_8	Runner-up	Macro_8 margin
DD / LSTM-S	WL	56.5	GraphBPE	15.3
DD / LSTM-D	WL	49.9	Shortest path	10.0
DD / XGB-S	WL	39.7	Shortest path	0.1
DD / XGB-D	WL	74.0	GraphBPE	35.6
AD / LSTM-S	WL	56.3	GraphBPE	9.5
AD / LSTM-D	WL	52.0	GraphBPE	9.8
AD / XGB-S	GraphBPE	48.2	Shortest path	4.7
AD / XGB-D	WL	51.9	GraphBPE	1.0

Table 13: Complete held-out Macro8 matrix for all 40 tokenizer-predictor-formulation-protocol configurations.

Family	WL	BPE	SP	SH	Rand
DD / LSTM-S	56.5	41.2	37.5	39.4	29.2
DD / LSTM-D	49.9	38.4	39.9	22.2	28.7
DD / XGB-S	39.7	38.4	39.6	37.1	35.3
DD / XGB-D	74.0	38.4	26.4	23.2	37.5
AD / LSTM-S	56.3	46.8	38.9	35.7	16.8
AD / LSTM-D	52.0	42.2	36.7	17.9	23.1
AD / XGB-S	38.4	48.2	43.5	24.3	43.3
AD / XGB-D	51.9	50.9	37.5	18.4	23.3

For a pair of methods A and B, let $\bar{y}_i^A$ and $\bar{y}_i^B$ denote the seed-averaged solved indicators for held-out problem i, and define the paired difference

$$\delta_i = \bar{y}_i^A - \bar{y}_i^B. \tag{23}$$

We evaluate paired differences with a two-sided sign-flip test on $\{\delta_i\}_i$ and apply Holm correction within each comparison set. We report nonparametric bootstrap 95% confidence intervals from held-out problems for the corresponding paired mean differences.

Error bars and ribbons in the appendix show one standard deviation over seeds unless noted otherwise. Seed-induced variance is sharply predictor-dependent: across the 80 regime-domain-tokenizer-formulation cells per predictor family, the seed standard deviation of the extrapolation solved rate is exactly zero in every XGBoost cell, whereas 65% of LSTM cells have zero seed deviation with a maximum of 38.3 points concentrated in the hardest settings (for example, the pooled `gripper` WL delta cell has per-seed extrapolation solved rates of 0, 81.3, and 0). Confidence-interval plots and significance tables are computed from held-out problems rather than from domain-balanced cell means. This per-problem (micro-averaged) computation means that the `visitall-from-everywhere` domain, which contributes 219 of the 273 extrapolation problems and 37 of the 52 interpolation problems, has a proportionally large influence on significance outcomes and transfer-effect magnitudes. Domain-balanced Macro_8 and $\text{Macro}_{\text{ext}}$ rankings are therefore the primary comparison metrics in the paper; per-problem statistical tests complement those rankings and should be interpreted alongside the domain-resolved figures.

A.8 Held-Out Statistics and Complete Result Matrices

This section collects the held-out statistics used throughout the paper. Validation-inclusive summaries are reported only as auxiliary audits. The tables below include the complete held-out ranking matrices, the additive effect summary, the split-wise significance counts, the DD-to-AD transfer effects, the search-effort diagnostics, and the validation audit.

Table 12 lists the eight held-out family winners referenced in the main paper. Table 13 provides the complete 40-configuration held-out ranking matrix, and Table 14 isolates extrapolation. Table 15 gives the additive effect summary discussed in the main text. Table 16 shows that extrapolation exposes more Holm-significant tokenizer contrasts than interpolation in every combination. Table 17

Table 14: Complete extrapolation Macroext matrix for all 40 tokenizer-predictor-formulation-protocol configurations.

Family	WL	BPE	SP	SH	Rand
DD / LSTM-S	26.1	27.4	25.0	20.8	6.4
DD / LSTM-D	30.3	27.4	28.0	23.1	25.3
DD / XGB-S	9.9	27.4	23.7	18.5	25.2
DD / XGB-D	67.5	27.4	25.0	23.3	25.0
AD / LSTM-S	26.4	27.4	25.0	29.5	1.1
AD / LSTM-D	32.7	23.8	25.3	8.0	12.9
AD / XGB-S	12.9	27.4	39.8	11.7	42.2
AD / XGB-D	40.0	27.4	25.0	11.0	19.4

Table 15: Exploratory additive effect summary on held-out Macro8 and extrapolation Macroext using seed-level configuration means. Because the design reuses benchmark cells and only three seeds are available, we treat this table as descriptive rather than as a full inferential decomposition.

Metric	Factor	F	p	η^2
Macro8	tokenizer	39.44	1.408e-20	0.575
Macro8	predictor	0.47	0.4936	0.002
Macro8	formulation	3.38	0.06867	0.012
Macro8	training protocol	0.84	0.3618	0.003
Macroext	tokenizer	6.41	0.0001116	0.174
Macroext	predictor	4.25	0.04165	0.029
Macroext	formulation	3.84	0.05267	0.026
Macroext	training protocol	1.35	0.2478	0.009
Macro8 interaction	tokenizer:formulation	8.00	1.089e-05	0.093

reports the DD-to-AD transfer effects at matched tokenizer-predictor-formulation settings. Table 18 reports search-effort diagnostics for the strongest state-centric configuration, and Table 19 confirms that solved plans coincide with the validator verdicts.

Figure 8 reports paired winner-versus-runner-up intervals for the eight protocol-predictor-formulation combinations. Figure 9 reports tokenizer rank changes from interpolation to extrapolation. Figure 10 reports DD-to-AD transfer at matched tokenizer-predictor-formulation settings. Figure 11 separates unsolved extrapolation cases into budget-limited and non-budget-limited failures.

A.9 Per-Domain Success and Failure Analysis

This section attributes search outcomes to measurable representation properties. All experiments in Appendices A.9–A.15 use the paper's strongest family (DOMAIN-DEPENDENT + XGBoost + delta, identical search and budget) at seed 13, which is sufficient because every XGBoost cell is seed-invariant (Appendix A.7); the runs reproduce the reported results before any modification.

One-step successor ranking. For each held-out expert transition (s_t, s_{t+1}), we embed s_t, predict the next latent state, embed every applicable successor of s_t, and record the rank of the expert successor under the search scoring rule. Table 20 reports extrapolation top-1 accuracy against the chance rate $\mathbb{E}[1/|\mathcal{N}(s_t)|]$.

Joining these one-step measurements to search outcomes gives a mechanistic account per domain. On `blocks`, WL is the only tokenizer above chance (top-3 rate 0.94 versus 0.73 chance; all others at chance) and correspondingly the only one that solves any extrapolation instance (14/20 versus 0 for all others). On `gripper`, WL, SimHash, and GraphBPE have top-1 near 0.48 with top-3 between 0.57 and 0.79, and beam width 3 converts that into 81–100% solved while greedy selection collapses to zero for every tokenizer (Appendix A.11); beam search is the error-recovery mechanism here. One caveat: top-1 against a single reference plan underestimates quality when several successors are plan-equivalent, as with `gripper`'s symmetric balls. On `visitall-from-everywhere`, the three tokenizers above chance (WL, shortest-path, and Random at 0.37–0.38 versus 0.29) solve 219/219

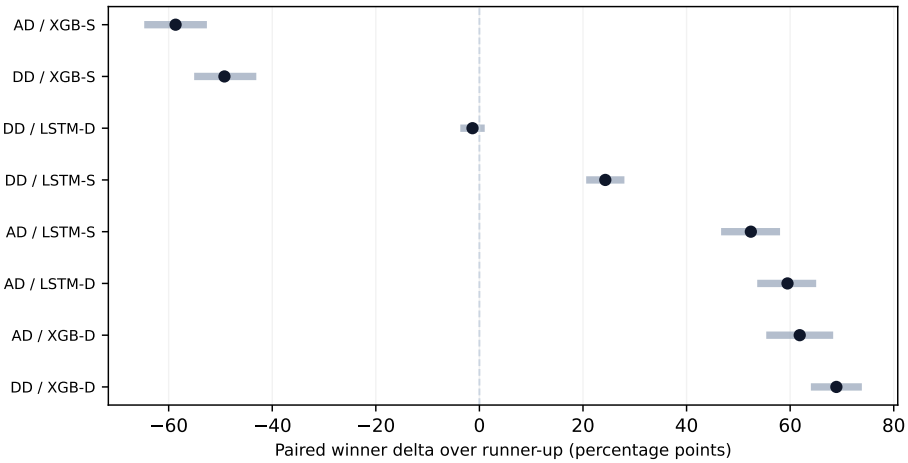

Figure 8: Winner-versus-runner-up paired confidence intervals for the eight protocol-predictor-formulation combinations. These intervals are computed from held-out per-problem seed means and complement the domain-balanced ranking table in the main text.

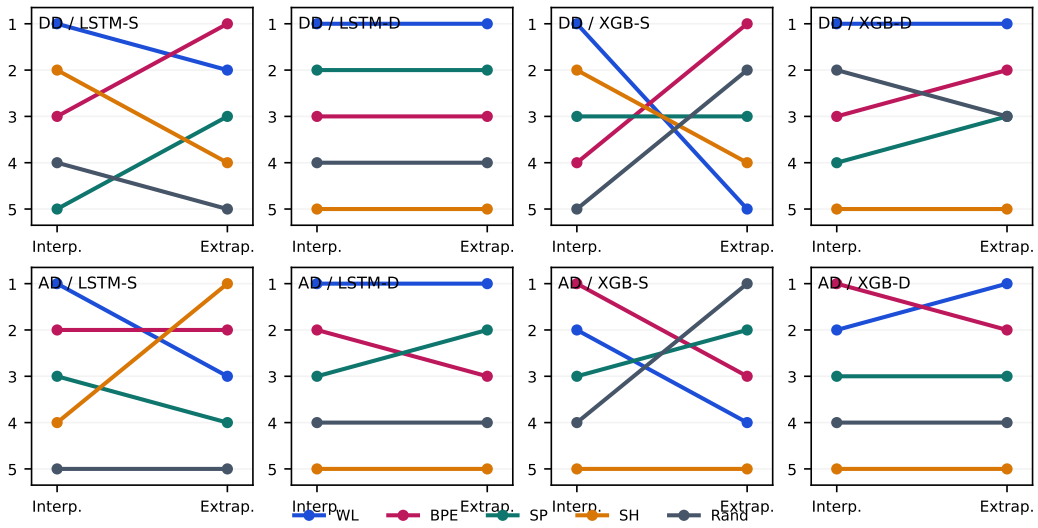

Figure 9: Interpolation-to-extrapolation rank changes for every tokenizer within each of the eight protocol-predictor-formulation combinations. The figure makes the 5-of-8 leader reversals in the main text visually explicit.

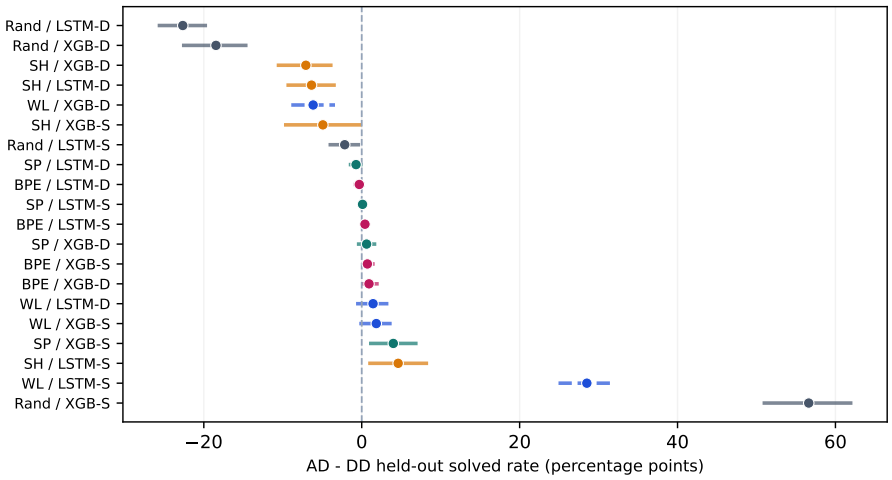

Figure 10: Held-out DD-versus-AD transfer effects for matched tokenizer-predictor-formulation settings. Dashed intervals correspond to WL, whose all-domain setting uses the union representation rather than a shared vocabulary.

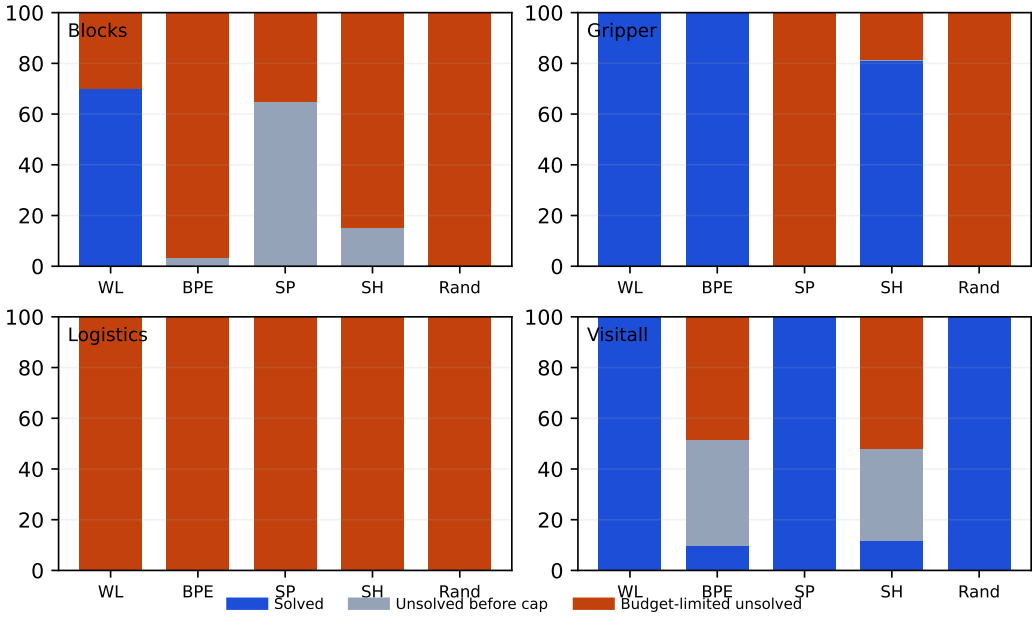

Figure 11: Extrapolation failure decomposition for DD / XGB-D across all tokenizers. Each stacked bar partitions held-out extrapolation problems into solved instances, unsolved instances that terminate before the search cap, and budget-limited unsolved instances.

Table 16: Number of Holm-significant tokenizer pairwise contrasts per family under held-out interpolation and extrapolation.

Family	Interp. sig pairs	Extrap. sig pairs
AD / LSTM-D	5	8
AD / LSTM-S	5	10
AD / XGB-D	4	10
AD / XGB-S	2	8
DD / LSTM-D	6	9
DD / LSTM-S	5	10
DD / XGB-D	2	8
DD / XGB-S	2	9

extrapolation instances, and still 197–203 of 219 under greedy selection, confirming the monotone visited-count progress signal needs almost no search; SimHash (0.26 one-step, 26/219 solved) loses the count magnitudes to sign-bit quantization, and GraphBPE (0.28, 21/219) coarsens them through merged labels. On `logistics`, the one-step signal is at chance for all five families, so failure is uniform and search-independent.

Embedding collisions on `logistics`. To attribute the `logistics` failure to the representations rather than the predictor, we measure the fraction of decision steps at which the expert successor's embedding is exactly identical to at least one alternative successor's embedding, so that no predictor over that representation could prefer the expert successor even in principle. On `logistics` extrapolation the collision rate is 82.6% for WL, 97.9% for SimHash, 68.9% for shortest-path, 100% for GraphBPE, and 96.1% for Random, with 1.8 to 19 indistinguishable alternatives per step on average. These ties are not benign symmetries: `logistics` packages have distinct goal destinations, so successors that load different packages lead to genuinely different subproblems. On `blocks`, by contrast, WL's collision rate is 0.0% (interpolation) and 1.2% (extrapolation), which is exactly why WL succeeds there while all tokenizers fail on `logistics`. This is representational indistinguishability in the sense of the expressiveness analyses of Ståhlberg et al. [2022], Martín and Geffner [2004], and Fern et al. [2006]; the two-iteration WL features used here sit inside the expressiveness classes those works show are insufficient for `logistics`.

Search-budget sensitivity on `logistics`. Raising the search budget tenfold (steps per object from 10 to 100) leaves `logistics` extrapolation at 0/18 for all five tokenizers, while WL interpolation rises from 22.2% to 66.7%. `Logistics` extrapolation is therefore representation-limited (budget cannot help because the one-step signal is at chance), whereas its interpolation split is partially budget-limited.

Why Random can beat SimHash. The two 128-dimensional references differ in one mechanism: SimHash keeps only the sign bit of each Gaussian projection (Appendix A.2), discarding count magnitudes, while Random keeps a normalized real-valued projection of predicate counts. In count-driven domains, magnitude is the progress signal, so the binary quantization is exactly what hurts SimHash, and Random solves zero `blocks` and `logistics` extrapolation instances, consistent with those domains requiring relational structure.

A.10 Spanner Stress-Test Domain

`Spanner` is a standard delete-relaxation trap with irreversible decisions: the agent traverses a one-way corridor, spanners are single-use, and walking past a spanner is an unrecoverable mistake. We generated 19 instances (train: 2–5 locations and 1–4 nuts; extrapolation: 8–20 locations and 5–12 nuts, plans up to 45 steps), each with a machine-verified reference plan, and ran the full pipeline for all five tokenizers. Extrapolation solved at beam width $k = 3$: GraphBPE 8/8, WL 1/8, shortest-path 0/8, SimHash 0/8, Random 0/8. This is a further leader reversal: WL, the overall winner on the original benchmark, is beaten decisively by GraphBPE, whose merged local structures preserve the spanner-location coupling that matters at the few irreversible decision points. `Spanner` also exposes a third failure mode distinct from `logistics` (at-chance prediction) and `gripper` (recoverable noise): a single early ranking mistake is unrecoverable by beam or budget.

Table 17: Held-out protocol-transfer comparison for matched tokenizer-predictor-formulation settings.

Config	AD-DD	95% CI	p
Rand / XGB-S	56.6	[50.8, 62.2]	0.0000
WL / LSTM-S	28.5	[24.9, 32.0]	0.0000
SH / LSTM-S	4.6	[0.8, 8.4]	0.0214
SP / XGB-S	4.0	[0.9, 7.1]	0.0194
WL / XGB-S	1.8	[-0.3, 4.3]	0.1827
WL / LSTM-D	1.4	[-0.7, 3.5]	0.2159
BPE / XGB-D	0.9	[0.0, 2.2]	0.2531
BPE / XGB-S	0.7	[0.0, 1.6]	0.2492
SP / XGB-D	0.6	[-0.6, 1.8]	0.6201
BPE / LSTM-S	0.4	[0.0, 0.9]	0.2522
SP / LSTM-S	0.1	[0.0, 0.3]	1.0000
BPE / LSTM-D	-0.3	[-1.0, 0.4]	0.2751
SP / LSTM-D	-0.7	[-1.6, 0.1]	0.1318
Rand / LSTM-S	-2.2	[-4.2, -0.2]	0.0289
SH / XGB-S	-4.9	[-9.8, 0.0]	0.0713
WL / XGB-D	-6.2	[-8.9, -3.4]	0.0001
SH / LSTM-D	-6.4	[-9.5, -3.3]	0.0001
SH / XGB-D	-7.1	[-10.8, -3.7]	0.0001
Rand / XGB-D	-18.5	[-22.8, -14.5]	0.0000
Rand / LSTM-D	-22.7	[-25.8, -19.6]	0.0000

Under greedy selection ($k = 1$), Random flips from 0/8 to 8/8 while GraphBPE stays at 8/8. The verified mechanism: greedy Random reproduces the reference plans exactly (its plan lengths match the reference lengths), consistent with its 0.80 one-step top-1; at $k = 3$, the searches terminate well before the budget because the extra branches compete on noisy cumulative scores, the correct pickup branch is pruned, and the surviving branches walk past spanners and dead-end at the gate. With a noisy but locally adequate representation, hedging across beam branches is what loses the irreversible decision, so beam width itself interacts with the representation.

A.11 Beam-Width Sensitivity and Greedy Selection

Table 21 reports the full beam-width grid $k \in \{1, 3, 5\}$ across all five tokenizers and all five domains. Two conclusions follow. First, widening the beam from 3 to 5 changes no ranking and moves no cell by more than three problems, so the reported comparisons are not sensitive to the beam-width choice upward. Second, all the interesting sensitivity lies at the greedy boundary, and it is tokenizer- and domain-dependent: `gripper` collapses to zero for every tokenizer under greedy selection (beam search is the error-recovery mechanism where top-1 accuracy is near 0.48), `blocks` WL degrades mildly and stays the only solver, `visitall-from-everywhere` barely changes for the strong tok-

Table 18: Search-effort summary for the best learned configuration, DD / WL / XGB-D, over held-out interpolation and extrapolation slices.

Slice	Solved	Cap hits	Mean budget use	Solved plan length
blocks / interp.	100.0	0.0	12.7	12.7
blocks / extrap.	70.0	30.0	57.2	49.0
gripper / interp.	100.0	0.0	27.9	29.0
gripper / extrap.	100.0	0.0	37.5	117.0
logistics / interp.	22.2	77.8	89.7	76.5
logistics / extrap.	0.0	100.0	100.0	–
visitall-from-everywhere / interp.	100.0	0.0	7.8	8.8
visitall-from-everywhere / extrap.	100.0	0.0	11.4	61.2

Table 19: Validation audit computed directly from the evaluation records. Every solved plan agrees with the validator verdict, and no solved plan is marked non-executable.

Metric	Value
Problem evaluations	42,960
Solved evaluations	20,402
Solved / VAL disagreement count	0
Solved but non-executable count	0

enizers, and `spanner` shows the beam-representation interaction discussed in Appendix A.10, where greedy Random solves all eight instances while beam search solves none.

A.12 Goal-Blind and Dimension-Matched Controls

The tokenizer families differ not only in encoding strategy but also in dimensionality and in how goal information enters the state channel: WL's state channel is strictly goal-blind, whereas SimHash, shortest-path, and GraphBPE place goal-derived features directly in the state representation. Two controls isolate these factors (Table 22).

Goal conditioning is not a driver. The goal-blind variants strip every goal-derived key (overlap, goal-missing, goal counts, goal-graph histograms) from the state channel while keeping the separate goal vector, giving all tokenizers exactly WL's state-only interface. The results are unchanged up to one or two problems per cell. Combined with the fact that WL, the only originally goal-blind tokenizer, is the overall winner, the unequal goal interface does not explain any part of the reported ranking.

Capacity is not a driver; feature geometry is. At a matched 128 dimensions, projected WL remains the strongest 128-dimensional tokenizer on `blocks` (2/20 versus 0/20 for both 128-dimensional references) and stays perfect on `gripper`, so dimension count alone reproduces neither WL's advantage nor the ordering among the equal-dimension families. The projection does cost WL performance on `blocks` and `visitall-from-everywhere` relative to its native form, and this is itself informative: a seeded Gaussian projection preserves the information in the histogram but destroys its sparse, axis-aligned count structure, which is exactly what a tree predictor splits on. The driver the data supports is therefore the structure of the feature space, not its size.

A.13 Tokenizer Hyperparameter Sensitivity

Each family in the main study uses the published defaults of its construction. This section sweeps the two construction hyperparameters with the clearest a priori influence on the feature space, WL refinement depth and shortest-path radius, holding everything else fixed (DOMAIN-DEPENDENT + XGBoost + delta, seed 13).

WL refinement depth. Table 23 sweeps the number of WL refinement iterations over $\{1, 2, 3\}$. The canonical depth 2 is a genuine sweet spot on `blocks`, and the failures on either side have dif-

Table 20: One-step expert-successor top-1 accuracy on held-out extrapolation transitions, with the chance rate in the last column. `spanner` is the stress-test domain of Appendix A.10.

Domain	WL	SimHash	Shortest-path	GraphBPE	Random	Chance
`blocks`	0.66	0.29	0.30	0.34	0.24	0.26
`gripper`	0.48	0.48	0.39	0.48	0.10	0.10
`logistics`	0.04	0.06	0.04	0.05	0.04	0.05
`visitall`	0.38	0.26	0.38	0.28	0.37	0.29
`spanner`	0.77	0.54	0.69	0.80	0.80	0.67

Table 21: Extrapolation instances solved, shown as greedy $k = 1$ / beam $k = 3$ / beam $k = 5$ (DOMAIN-DEPENDENT + XGBoost + delta, seed 13). Rankings are stable from $k = 3$ to $k = 5$; the behavioral changes concentrate at the greedy boundary.

Domain (problems)	WL	SimHash	Shortest-path	GraphBPE	Random
`blocks` (20)	11 / 14 / 13	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
`gripper` (16)	0 / 16 / 16	0 / 13 / 13	0 / 0 / 0	0 / 16 / 16	0 / 0 / 0
`logistics` (18)	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
`visitall` (219)	203 / 219 / 219	8 / 26 / 23	203 / 219 / 219	8 / 21 / 21	197 / 219 / 219
`spanner` (8)	1 / 1 / 2	0 / 0 / 0	1 / 0 / 0	8 / 8 / 8	8 / 0 / 0

ferent, measurable causes. At depth 1 the representation under-distinguishes: the expert-successor collision rate on `blocks` extrapolation rises to 14.4% (versus 1.2% at depth 2), and solved instances collapse from 14/20 to 1/20; `gripper` collapses entirely, including its interpolation split. At depth 3 the collision rate falls to 0.1%, so states are almost perfectly distinguished, yet `blocks` still collapses to 2/20: failure without indistinguishability, consistent with over-specialized colors that fail to recur on larger held-out instances and leave the fitted vocabulary. `Visitall-from-everywhere` is depth-insensitive, and `logistics` remains unsolved at every depth, consistent with its expressiveness limits (Appendix A.9). These results reinforce the central finding at a finer grain: what matters is whether the feature construction distinguishes exactly the distinctions that recur across instance sizes, not the volume of features produced.

Shortest-path radius. Sweeping the maximum path length over $\{3, 5, 7\}$ produces identical results in every held-out cell (0/20 on `blocks`, 0/16 on `gripper`, 0/18 on `logistics` extrapolation with 1/9 interpolation, and 219/219 on `visitall-from-everywhere`). The shortest-path family's successes and failures are therefore entirely insensitive to its radius within this range: its `visitall-from-everywhere` strength comes from short-range connectivity structure already captured at radius 3, and its failures elsewhere are representational rather than a matter of histogram range.

Additional seeds for the strongest configuration. Beyond the three seeds of the factorial study, two additional seeds (47 and 59) for DOMAIN-DEPENDENT + WL + XGBoost + delta reproduce every domain-split cell exactly (14/20, 16/16, 0/18 with 2/9 interpolation, 219/219). The headline configuration is therefore verified seed-invariant over five seeds, complementing the exact three-seed invariance of all 80 XGBoost cells (Appendix A.7).

A.14 Sensitivity to Removing the Largest Domain

Because `visitall-from-everywhere` contributes the majority of benchmark problems, we recompute the held-out macro over the remaining three domains and two held-out splits from the recorded per-seed results (Table 24; the full-benchmark column reproduces the values reported in the main text). The variance decomposition on the same seed-level configuration means gives, with `visitall-from-everywhere` removed, $\eta^2 = 0.641$ for tokenizer (versus 0.575 on the full benchmark), with predictor at 0.000, formulation at 0.033, and training protocol at 0.005. Removing the largest domain therefore makes tokenizer choice more dominant, not less. The tokenizer spread remains 26 points, and the only ranking change (GraphBPE edging WL by 1.8 points) is consistent with the domain-specialization finding, since `visitall-from-everywhere` is where shortest-path and Random are strongest.

Table 22: Isolation controls: extrapolation instances solved (DOMAIN-DEPENDENT + XGBoost + delta, seed 13).

Configuration	blocks	gripper	logistics	visitall
Shortest-path, published form	0/20	0/16	0/18	219/219
Shortest-path, goal-blind state channel	0/20	0/16	0/18	219/219
SimHash, published form	0/20	13/16	0/18	26/219
SimHash, goal-blind state channel	0/20	14/16	0/18	24/219
WL, native dimension (138–552)	14/20	16/16	0/18	219/219
WL, random-projected to 128	2/20	16/16	0/18	91/219

Table 23: WL refinement-depth sweep: extrapolation instances solved (DOMAIN-DEPENDENT + XGBoost + delta, seed 13). Depth 2 is the paper's canonical setting.

WL iterations	blocks	gripper	logistics	visitall
1	1/20	0/16	0/18	219/219
2 (canonical)	14/20	16/16	0/18	219/219
3	2/20	16/16	0/18	219/219

A.15 Training Composition under Pooled Training

Pooled ALL-DOMAINS training concatenates all training trajectories with no domain-balanced sampling and no loss weighting; the LSTM sees all trajectories in one pool and XGBoost receives all flattened transition pairs equally weighted. Table 25 reports the resulting composition. The protocol deliberately evaluates naive pooling as commonly practiced, so the imbalance interaction is part of the measured effect; the domain-balanced evaluation metrics correct the reported averages but not the training distribution, and the per-domain transfer analyses (Figure 15) resolve the resulting effects by domain.

A.16 Additional Visual Diagnostics

This section reports additional diagnostics that complement the main-paper figures. The figures cover four themes: compact global summaries, size-scaling diagnostics, domain-resolved transfer views, and optimization and budget diagnostics.

Figure 12 provides two compact views of the held-out results. The split-bar panel averages within domains before averaging across domains, so the extrapolation drop is a balanced split effect rather than a count-weighted benchmark mean. The radar panel highlights domain specialization: GraphBPE is nearly perfect on `gripper`, shortest-path is strongest on `visitall-from-everywhere`, and WL remains strongest on `blocks` and `logistics`.

Figure 13 gives protocol-controlled size curves for the tree predictor, regenerated from per-problem records of the DOMAIN-DEPENDENT and ALL-DOMAINS XGBoost delta runs; the reproduction matches the reported results exactly except that, for the high-dimensional WL union representation under ALL-DOMAINS, CPU histogram training flips three borderline `blocks` searches relative to the reported GPU runs (9/20 versus 12/20), affecting no ranking. Unlike the size plot in the main paper, these panels separate the two training protocols, so they show whether a tokenizer fails abruptly at a specific scale or degrades smoothly as problems grow, and whether pooled training changes that shape. Per-size curves for the LSTM predictor are not shown: with one to three benchmark instances per size bin, a per-size per-seed value is 0 or 100 by construction, so a seed ribbon at this granularity spans most of the axis and is uninformative; LSTM seed behavior is instead quantified at the domain-split level, where 65% of LSTM cells have zero seed deviation and the maximum is 38.3 points (Appendix A.7), with optimization diagnostics in Figure 16.

Figure 14 strips away protocol-specific variation to expose domain-wise size ordering for the tree predictor. Tokenizers remain sharply separated on `visitall-from-everywhere`, `logistics` remains uniformly unsolved, and `blocks` separates WL from all other families across the entire size range.

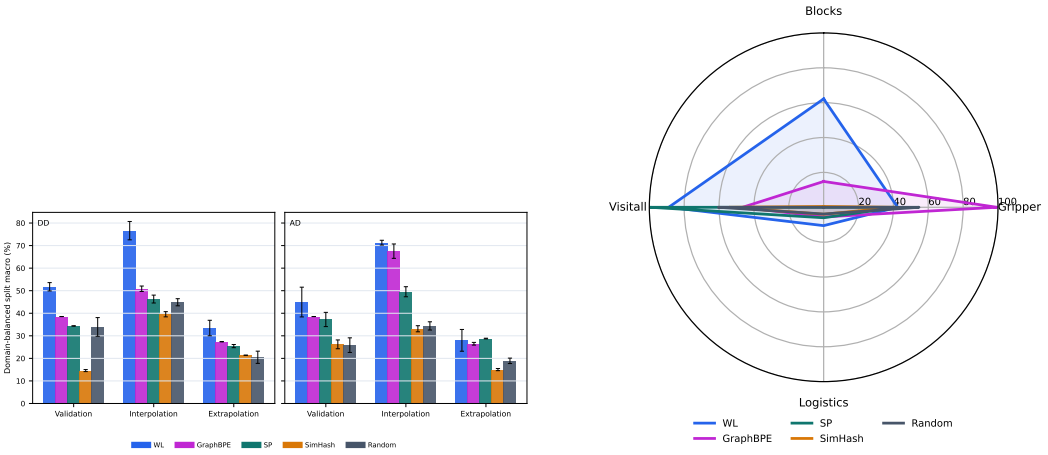

Figure 12: Compact global summaries of the held-out study. Left: domain-balanced split averages by tokenizer and training protocol, with error bars over seeds. Right: radar chart of tokenizer means over the four held-out domains after averaging over both training protocols, both predictor families, and both formulations.

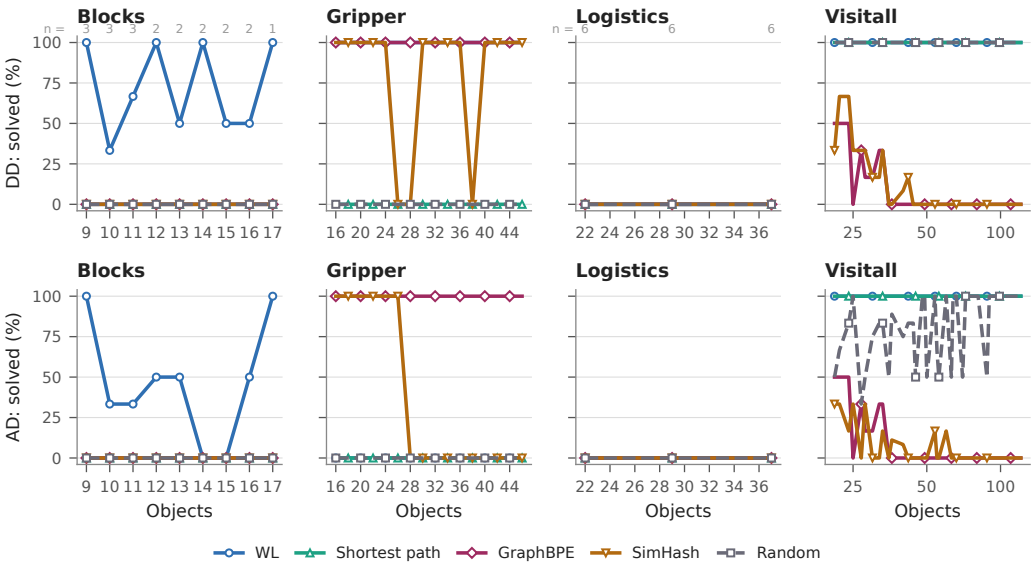

Figure 13: XGBoost delta size curves under the two training protocols (top row: DOMAIN-DEPENDENT; bottom row: ALL-DOMAINS). The horizontal axis is the actual object count of each held-out extrapolation instance, and the muted row above each top-row panel gives the number of benchmark instances at that size (`gripper` has exactly one per size). No seed bands are drawn because the seed standard deviation is exactly zero in every XGBoost cell (Appendix A.7). Random is drawn dashed as the neutral reference tokenizer.

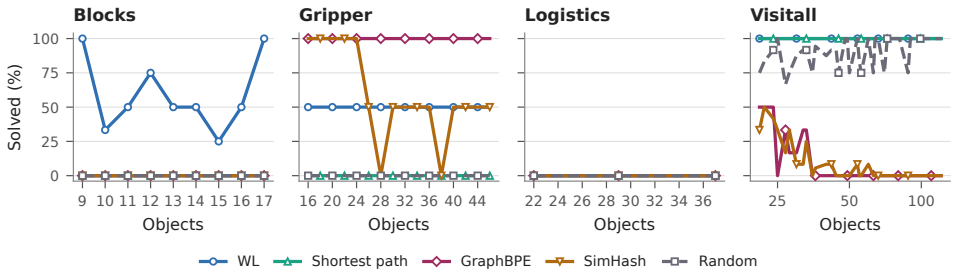

Figure 14: Average extrapolation size curves by domain for the XGBoost delta formulation, averaged over the two training protocols. The horizontal axis is the actual object count; the `visitall-from-everywhere` panel uses a log scale because the extrapolation range is much wider than in the other domains. Random is drawn dashed as the neutral reference tokenizer.

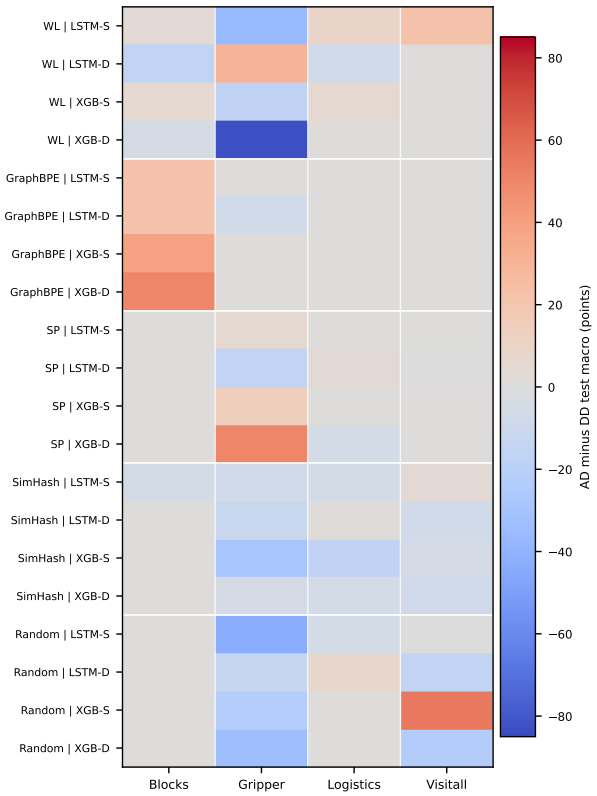

Figure 15: Domain-resolved DD-versus-AD transfer effects for every tokenizer-predictor-formulation configuration. Positive values favor pooled all-domain training and negative values favor domain-dependent training.

Table 24: Domain-balanced held-out macro per tokenizer, averaged over the eight protocol-predictor-formulation families, with and without `visitall-from-everywhere`.

Tokenizer	Macro, all 4 domains	Macro, `visitall` removed
WL	52.3	41.4
GraphBPE	43.1	43.2
Shortest-path	37.5	16.9
SimHash	27.3	19.0
Random	29.7	19.2

Table 25: Per-domain training composition under pooled ALL-DOMAINS training.

Domain	Train problems	Transitions	Share
`blocks`	9	126	5.2%
`gripper`	4	56	2.3%
`logistics`	12	148	6.1%
`visitall-from-everywhere`	207	2085	86.3%

Figure 15 complements the transfer lollipop by resolving the sign and magnitude of DD-versus-AD effects at the domain level. The heatmap shows that transfer is not even sign-consistent across domains: several configurations gain on `blocks` while losing on `gripper`, and the strongest negative cells are concentrated on `gripper` for WL and random-feature configurations.

Figure 16 reports seed variance and optimization behavior. Most XGBoost cells are exactly invariant across seeds, while the LSTM runs retain a heavier right tail of unstable cells on the hardest settings. The learning-curve panel shows that, on the difficult domains, tokenizers can converge at different rates and to different validation-loss plateaus even when trained with the same optimizer and horizon.

Figure 17 summarizes two boundary conditions. The cap-rate panel shows that `logistics` is budget-limited on almost the entire validation and extrapolation evaluation, whereas `visitall-from-everywhere` extrapolation remains mostly below the cap and therefore leaves more room for tokenizer differences. The compactness panel shows that lower-dimensional representations do not monotonically imply stronger extrapolation.

A.17 Implementation Details and Reproducibility

The reported study uses the tokenizer set {`wl`, `simhash`, `shortest_path`, `graphbpe`, `random`}, the predictor set {`lstm`, `xgboost`}, the two formulations {`state`, `delta`}, the two training protocols {DOMAIN-DEPENDENT, ALL-DOMAINS}, and seeds {13, 23, 37}. All reported planning outcomes are validated with VAL, and all searches use the same inference budget formula.

For reproducibility, the appendix specifies the tokenizer constructions, predictor hyperparameters, held-out evaluation protocol, statistical tests, and compute environment used for the study. The experiments were executed with CUDA-enabled PyTorch 2.10.0, XGBoost 3.2.0, and `wlplan` 2.1.0 on a single NVIDIA GeForce RTX 4080 SUPER GPU with 16376 MiB memory, together with an AMD Ryzen 7 7800X3D CPU and eight CPU workers for validation and XGBoost jobs. The full factorial study required approximately 31.3 wall-clock hours.

Table 26 gives approximate per-command inference runtimes reconstructed from the recorded experiment schedule. The protocol/predictor medians are all under 65 seconds, but the distribution is highly skewed because `visitall-from-everywhere` evaluations are much slower than the other domains: the median per-command wall time is 18.7 seconds on `blocks`, 58.0 seconds on `gripper`, 56.7 seconds on `logistics`, and 384.5 seconds on `visitall-from-everywhere`. The study does not record node expansions or pure search runtime per problem, so the search-effort analyses use quantities that are available consistently across all runs: solved rate, plan length, effective step budget, budget-use ratio, and cap-hit rate.

Table 26: Approximate per-command inference wall-clock reconstructed from sequential experiment timestamps. Each duration is the difference between adjacent command-start times, so the values summarize end-to-end inference-plus-validation command time rather than pure search time.

Protocol / predictor	Median (s)	90th pct. (s)	Max (s)
DD / LSTM	61.1	525.3	841.4
DD / XGBoost	47.4	413.7	6821.4
AD / LSTM	64.2	490.4	860.3
AD / XGBoost	52.3	446.2	10995.8

A.18 Code and Data Availability

The code-and-data repository for this paper is available at `https://github.com/ai4society/planfm-state-centric-tokenization`. Its README provides setup and reproduction instructions for the implementation, generated data products, tables, and figures used in the reported study.

A.19 Broader Impacts and Responsible Use

This paper studies evaluation methodology for learned planning rather than a deployed decision system. A positive impact of the work is clearer empirical accounting of representation choice in symbolic planning, which can improve reproducibility and reduce misleading conclusions drawn from pooled benchmark averages. A potential negative impact is that stronger learned planning systems may be over-interpreted as evidence of deployment readiness in robotics, scheduling, or other automation settings where benchmark success does not guarantee robustness. For that reason, the paper emphasizes held-out evaluation, validator-checked plans, and domain-resolved reporting rather than broad claims of general planning competence.

A.20 Limitations

The factorial study uses three seeds, although seed variance is exactly zero in all XGBoost cells, small in most LSTM cells (Appendix A.7), and the headline configuration is verified invariant over five seeds (Appendix A.13). The comparison between the state and delta formulations changes the target, loss, and retrieval metric simultaneously. For WL, DD-to-AD comparisons also combine pooled training with the union representation used in the all-domain setting. SimHash and the deterministic random tokenizer serve as reference tokenizers, but the study does not include a learned feature-pool family in the concept-language tradition, which we consider the most valuable extension. The tokenizer families differ in dimensionality and construction rules; the controls in Appendix A.12 exclude goal conditioning and dimension count as drivers, and the sweeps in Appendices A.11 and A.13 cover WL refinement depth, shortest-path radius, and beam width, leaving the GraphBPE merge budget and vocabulary size and the SimHash hash width at their single published settings. The analyses in Appendices A.9–A.13 use the strongest configuration family, justified by the XGBoost seed invariance.

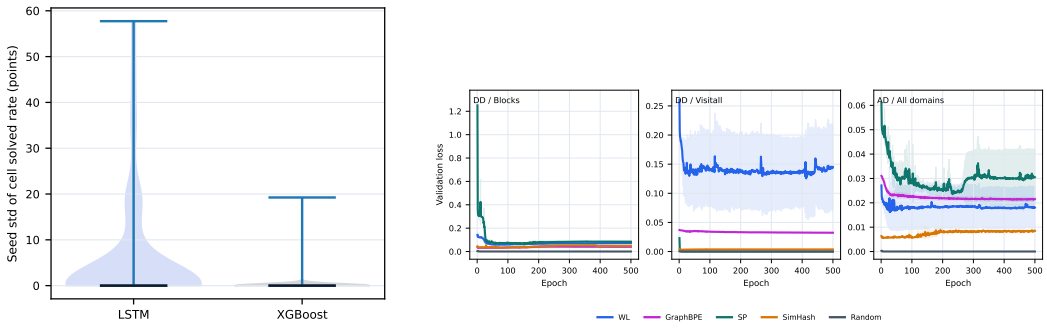

Figure 16: Variance and optimization diagnostics. Left: distribution of seed standard deviations over the domain-split setup cells. Right: representative LSTM validation-loss trajectories with one-standard-deviation ribbons over seeds.

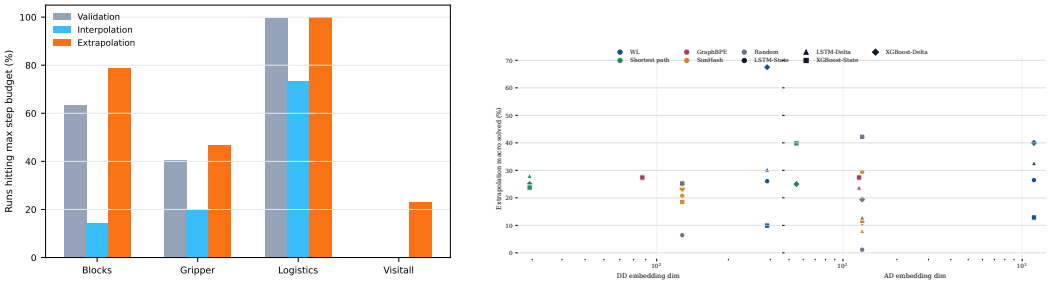

Figure 17: Boundary conditions and secondary diagnostics. Left: fraction of runs that terminate at the search-step budget in each domain and split. Right: compactness versus extrapolation quality.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction present the paper as a benchmark-specific study of representation sensitivity inside one fixed learned planning stack, and Sections 4–5 plus the appendix provide the corresponding evidence and scope-limiting caveats.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Appendix A.20 discusses the missing tokenization-free control, the coupled state/delta formulation choices, the WL pooled-representation confound, the three-seed variance limit, and the benchmark regions that are search-budget dominated.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper provides a formal problem formulation and explicit equations, but it does not claim any theorem-level theoretical result requiring a formal proof.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Sections 3–4 and the appendix specify the tokenizer constructions, transition predictors, domains, splits, metrics, seeds, inference budget, validation protocol, software environment, and computational details needed to reproduce the reported results.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Appendix A.18 provides a repository URL with the implementation, and README instructions for reproducing the reported study.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Sections 3–4 and Appendices A.2, A.3, A.5, and A.17 provide split counts, tokenizer settings, predictor hyperparameters, optimizers, validation/model-selection rules, search budget, and run-configuration details.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Section 4.3 defines the paired sign-flip tests, Holm correction, and bootstrap 95% confidence intervals used in the paper; Section 5 reports representative corrected comparisons, and Appendix A.8 includes the full held-out significance summaries. The additive ANOVA-style table is explicitly labeled descriptive rather than inferential.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 4.2 and Appendix A.17 report the RTX 4080 SUPER GPU (16376 MiB), CPU/worker setup, the approximately 31.3-hour wall-clock span of the full study, and an approximate per-command runtime table reconstructed from the recorded experiment schedule.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work uses standard benchmark domains, no human subjects, and no high-risk model release; the manuscript also discusses responsible interpretation and limitations in the appendix.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The appendix subsection "Broader impacts and responsible use" discusses both the positive impact of clearer learned-planning evaluation and the potential risks of brittle automation or misleading aggregate conclusions.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release high-risk generative models, scraped human data, or other assets that require special access controls.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [No]

Justification: Appendix A.18 and the repository credit the main software and benchmark assets used by the study, but the submission does not claim complete license coverage for every upstream benchmark asset.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: Appendix A.18 links a repository containing the implementation, generated data products, tables, figures, and documentation for the reported experiments. The benchmark domains and external software components are cited separately rather than presented as newly authored assets.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The work does not involve crowdsourcing or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The work does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs were not part of the core methodology, experiments, or scientific claims of the paper.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.