

Unifying Policy Learning and State Prediction through Spatial Language Modeling

Minye Wu Zehao Wang Tinne Tuytelaars
KU Leuven

Abstract: Learning how actions change scene geometry can provide complementary supervision for goal-directed manipulation. We introduce Spatial Language Modeling, which represents scene contours, goals, action targets, and future states with a shared vocabulary of discrete coordinates and semantic tokens. A task-specific grammar organizes these elements into spatial sequences, allowing one autoregressive Transformer to learn action generation and action-conditioned state prediction through a common next-token objective. We train the model from scratch using random-play transition pretraining followed by joint action and state training on expert demonstrations. During pretraining, recorded action coordinates condition subsequent state predictions and are excluded from the prediction loss. During control, the model decodes only executable action targets and updates its history with newly observed states. We evaluate the approach on Push-T in simulation and on a real robot. The model achieves competitive simulation performance and higher task success and target coverage than the evaluated real-robot policy baselines. Training ablations show improved control with joint action and state sequences, with further gains from random-play pretraining. Given supplied action trajectories, the same model also predicts successive scene states, capturing the geometric effects of pushing.

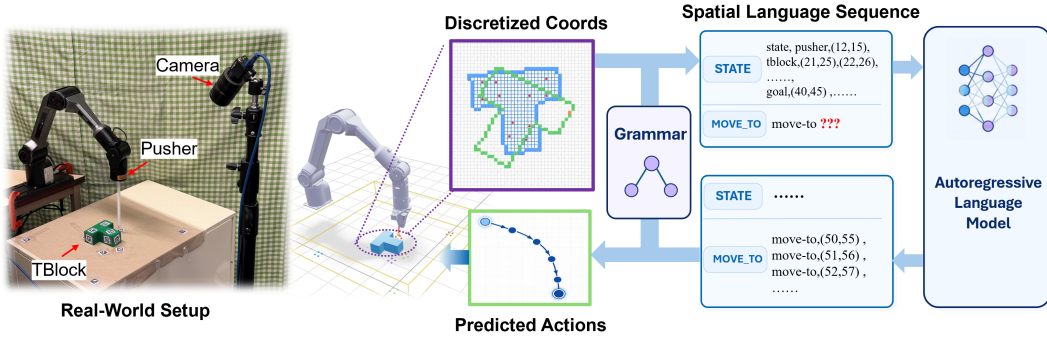


Figure 1: Spatial Language Modeling for Push-T. Scene geometry, goals, pusher actions, and optional future states are represented as structured coordinate-token sequences. The model generates actions for closed-loop execution and learns state transitions through future-state supervision.

1 Introduction

Robot manipulation requires selecting actions that transform a scene toward a goal. In the Push-T task [1], a planar pusher moves a T-shaped block to a target pose. The resulting motion depends on both object geometry and contact: the same pusher displacement can induce different combinations of translation and rotation depending on the contact location and the block’s orientation. Demonstrations of these interactions provide two complementary training signals: expert actions indicate how to act, while subsequent observations reveal the geometric changes that follow.

Policies such as Action Chunking Transformers, Diffusion Policy, and vision-language-action models learn actions from demonstrations [2, 1, 3, 4]. World models use future predictions for rollout

generation and planning [5, 6], while recent work also jointly models actions and future observations [7]. Predicting the geometric consequences of actions can provide complementary supervision for policy learning and enable the use of non-expert interaction data. A shared prediction interface allows this transition supervision to update the same model parameters used for action generation. We therefore seek a spatial representation that makes both targets accessible to the same autoregressive model.

Discrete workspace coordinates provide such an interface at the level of explicit geometry. We divide a bounded planar workspace into grid cells and assign each cell a coordinate token. Object contours, target regions, pusher positions, and action trajectories can then be represented in the same spatial vocabulary. A coordinate retains the same physical reference whether it appears in an observed contour, an executable action target, or a predicted future state. Semantic tokens identify whether coordinates belong to the pusher, the manipulated object, the goal, or a movement command. Geometry and actions thus retain their distinct roles within a shared spatial reference frame.

We develop this representation as **Spatial Language Modeling**. A task-specific grammar organizes scene observations, goals, action targets, and optional subsequent states into a *Spatial Language Sentence* (SLS). The grammar specifies how these elements are ordered and identified, while the model learns their coordinate content. A serializer converts observed geometry and recorded actions into spatial sentences, and a parser maps generated action blocks back to pusher targets. This formulation turns action generation and action-conditioned state prediction into sequence-completion tasks that share an output vocabulary and a next-token training objective.

We train an autoregressive Transformer from scratch using two sources of interaction data. Random-play pretraining first trains the model to predict scene transitions conditioned on recorded pusher motions. The action coordinates remain in the input sequence, but their prediction losses are masked; subsequent states supply targets describing the resulting geometry. Expert-demonstration training then supervises both action targets and selected subsequent states. These losses update the same parameters, making the geometric effects of interaction an explicit learning target alongside goal-directed action generation. This training scheme connects transition information from random interactions with action supervision from expert demonstrations.

During closed-loop control, an observed-history SLS prefix describes recent scene states and pusher actions. The model completes this prefix with consecutive action blocks, which are parsed into target coordinates and executed by the robot. Future-state blocks are excluded from control decoding, and new observations refresh the history for the next completion. The policy can therefore use a model trained with transition supervision while generating only actions at execution time. The same model also supports a separate prediction setting: given an initial scene and a supplied action trajectory, it generates successive scene states, using its previous state predictions as context.

We evaluate the formulation on Push-T in simulation and on a real robot. The model achieves competitive simulation performance and higher task success and target coverage than the evaluated real-robot policy baselines. Comparisons of training variants show improved control with interleaved action and state sequences, with further gains from random-play pretraining. In the state-prediction setting, predicted object states retain substantial geometric overlap with simulator reference states across the evaluated rollout horizon.

Our contributions are as follows:

- We introduce a grammar-structured spatial representation that expresses observed geometry, goals, action targets, and future states in a shared coordinate vocabulary, enabling action and state prediction through autoregressive sequence completion.
- We develop a training scheme that combines action-conditioned transition pretraining on random interactions with joint action and state supervision on expert demonstrations, and supports closed-loop execution through action-only decoding.

- We demonstrate closed-loop Push-T control in simulation and on a real robot, and report results and comparisons that support the proposed formulation.

2 Related Work

Policy Learning and World Modeling. Imitation-learning policies use demonstrations to supervise action distributions. Action Chunking Transformers predict temporally extended action sequences, while Diffusion Policy generates actions through trajectory denoising [2, 1]. Vision-language-action models connect action learning to vision-language pretraining, heterogeneous robot data, and fine-tuning [3, 8, 4]. Sequence modeling also provides a direct route to control: Decision Transformer predicts actions conditioned on returns and trajectory history, and Behavior Transformers combine action discretization with continuous corrections to model multimodal demonstrations [9, 10].

World models learn scene transitions and connect future predictions to control through several interfaces. DreamGen generates visual trajectories and obtains action labels through inverse dynamics or latent-action models for policy training [5]. LingBot-VA interleaves video prediction and action decoding in a causal sequence [7]. RoboDreamer synthesizes compositional video plans [11], Dreamitate tracks generated tool trajectories for robot execution [12], and LeWorldModel predicts future latent embeddings for planning [6]. Our formulation uses coordinate-level scene transitions and action targets within one next-token objective, combining random-play transition pretraining with joint action/state training on demonstrations.

Language Models in Spatial Computation. Language models have been applied to spatial intelligence through several representations of 3D scenes. PointLLM and 3D-LLM connect point-cloud or scene features to language models for object and scene understanding [13, 14]. SegPoint extends this connection to instruction-conditioned segmentation [15]. LSceneLLM adaptively selects task-relevant scene details, while Scene-LLM combines scene-level and egocentric features for 3D visual reasoning [16, 17]. Other approaches expose spatial structure through prompting: SayPlan uses hierarchical 3D scene graphs for robot task planning [18], and 3DAxisPrompt supplies visual coordinate axes and masks for 3D grounding and reasoning [19].

Geometric sequence models provide another route to spatial representation. PolyGen autoregressively generates mesh vertices and faces, and MeshGPT generates discrete codes from a learned geometric vocabulary [20, 21]. Geometry-aware patch tokenization also supports mesh classification and segmentation [22]. LLaMA-Mesh and MeshLLM connect mesh generation to language models through textual representations of geometry [23, 24]. In 2D vision, Pix2Seq represents bounding boxes and class labels as discrete output tokens, formulating detection as sequence prediction [25].

Scene Understanding and Generation. 3D scene understanding develops geometric representations for identifying objects and their relationships. PointNet, PointNet++, PointConv, and Point Transformer learn features directly from point sets [26, 27, 28, 29]. VoxFormer predicts volumetric geometry and semantics from images [30], while ScanRefer and context-aware grounding models connect language to objects in reconstructed scenes [31, 32]. Semantic scene graphs explicitly represent object relations [33], and PointCLIP models extend recognition using vision-language priors [34, 35]. Complementary generative methods synthesize voxel shapes [36], surfaces and textured meshes [37, 38], or point clouds [39]. DreamFusion, Magic3D, and Fantasia3D use text-to-image diffusion priors for text-conditioned 3D content creation [40, 41, 42]. GALA3D organizes scene generation through layout-guided Gaussian representations [43], while SmartSpatial uses depth and attention control to improve spatial arrangements in generated images [44].

Across these directions, spatial representations provide the interface between scene geometry and task-dependent predictions. Our work organizes object contours, goals, action targets, and future-state targets into coordinate and semantic token sequences, with their roles and temporal order specified by a task grammar. In Push-T, this interface supports joint action/state training and direct closed-loop execution through one autoregressive language model.

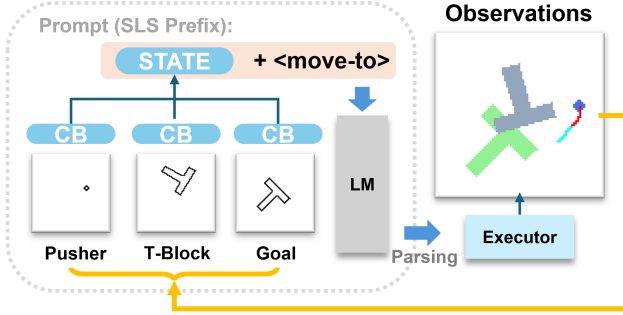


Figure 2: Push-T workflow. The model completes an observed-history SLS prefix into actions, and new observations close the control loop.

Table 1: Push-T sequence format. Each COORD is one token; control decoding omits optional state blocks.

Symbol	Definition
Context-Free Grammar	
SEQ	STATE MOVE_TO*
MOVE_TO	move-to CB [STATE]
STATE	state pusher CB* tblock CB* goal CB*
CB	COORD
Lexical Tokens	
COORD	One ID in $\mathcal{Q}_{coord}$, displayed as (p_1, p_2) with $0 \leq p_j < v_j$

3 Method

We formulate *Spatial Language Modeling* as autoregressive completion over grammar-structured token sequences for Push-T. Scene geometry, the goal, pusher actions, and optional future states are represented by coordinate and semantic tokens in a *Spatial Language Sentence* (SLS). Training sequences contain observed states and rollout targets. At inference, an observed-history prefix prompts the model to sample a completion, which is parsed into pusher actions. The grammar defines the sequence format and parser.

3.1 Spatial Language Modeling

Tokenization and Vocabularies. We define the planar workspace as an axis-aligned rectangle with minimum and maximum corners $\mathbf{b}_{min}, \mathbf{b}_{max} \in \mathbb{R}^2$. The workspace is discretized into a regular grid with cell size v_s and dimensions $\mathbf{v}_g = (v_1, v_2)$, where $\mathbf{v}_g = \lceil (\mathbf{b}_{max} - \mathbf{b}_{min}) / v_s \rceil$. All possible grid locations form the coordinate vocabulary $\mathcal{Q}_{coord}$.

We project scene geometry onto the grid and represent occupied contour cells as coordinate tokens. Multiple contour points in the same cell map to the same token. Each grid coordinate is represented by one model token with a unique integer ID under a fixed row-major indexing convention. The tuple $\mathbf{p} = (p_1, p_2)$ is a readable label for this single token. The grid resolution trades geometric precision against vocabulary size and the number of tokens needed to represent contours; it also quantizes action targets.

The semantic vocabulary $\mathcal{Q}_{semantic}$ identifies state blocks, entity types, and pusher motion commands. The Push-T symbols and their roles are specified in § 3.2 and the supplementary material.

Grammar as an Interface. An SLS s is an ordered sequence of tokens from the union of the two vocabularies:

$$s \in (\mathcal{Q}_{coord} \cup \mathcal{Q}_{semantic})^n, \quad (1)$$

where n is the sequence length. A context-free grammar (CFG; Tab. 1) specifies the ordering and combination of tokens in training sequences. The SEQ rule defines the rollout format. A compiler serializes observations and actions into this format, and a parser converts generated action blocks back into target coordinates. For control, a grammar state machine excludes future-state blocks; action coordinates are sampled autoregressively.

3.2 Application to Push-T

We instantiate Spatial Language Modeling for Push-T [1], where a planar pusher moves a T-shaped block to a specified target pose. The input describes the pusher, block, and goal geometry; outputs specify pusher target coordinates and optional subsequent scene states.

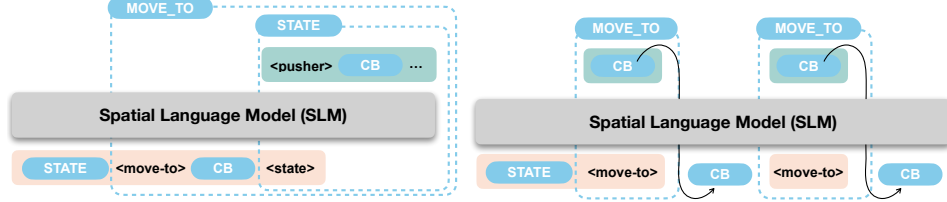


Figure 3: SLS completion for action-conditioned state prediction (left) and action-only control decoding (right). Generated action blocks are parsed into pusher target coordinates.

We define a coordinate system and a bounding rectangle on the tabletop, then discretize it into coordinate tokens. Tab. 1 summarizes the grammar and token definitions, and Fig. 2 illustrates the Push-T workflow.

Scene Description. We project the T-block, pusher, and target region onto the plane, keep their outer contours, and represent each entity using the coordinate sequence CB^+ of its occupied contour cells. These sequences form the CFG nonterminal STATE.

Action Definition and Future Prediction. A MOVE_TO block starts with the semantic token move-to, followed by one coordinate token specifying the target of the pusher center and, optionally, a STATE describing the subsequent scene. Observed subsequent states supply scene-transition targets during training. Action generation and state prediction use the same autoregressive model and next-token objective, so both targets update the shared parameters used to generate actions. During control, the model produces consecutive action blocks, reducing the output token count, and closes the loop through new observations.

Rollout Format. Each Push-T rollout starts with STATE followed by a temporally ordered sequence of MOVE_TO productions, matching the SEQ rule. Each step is one pusher movement and corresponds to one scene frame. A MOVE_TO block specifies the target coordinate for that step and optionally the resulting STATE. Rollout discretization makes consecutive pusher positions adjacent on the grid, yielding local displacements in the serialized trajectories.

Training and Inference. We compile rollouts into SLS sequences and optimize next-token cross-entropy:

$$\mathcal{L}(\theta) = - \sum_{i=1}^n m_i \log p_{\theta}(s_i \mid s_{<i}), \quad (2)$$

where $m_i \in \{0, 1\}$ indicates whether position i contributes to the loss. In expert-demonstration training, $m_i = 1$ for all sequence positions. During random-play pretraining, $m_i = 0$ only for the action-coordinate token immediately following move-to; all remaining tokens, including the command marker and state coordinates, contribute to the loss. Subsequent STATE blocks are independently retained with probability 30%; when omitted, their MOVE_TO blocks contain only the pusher target. Retained blocks provide scene-transition targets, while omitted blocks create consecutive-action subsequences within the same training format.

For closed-loop control, the prefix comprises three temporally consecutive STATE–MOVE_TO segments and ends with move-to for the next action coordinate. A grammar state machine excludes future STATE blocks while the model samples action coordinates autoregressively (Fig. 3). Parsed targets are executed, and new observations update the history for the next prefix (Fig. 2). The execution schedule is specified in Appendix B.

Pretraining with Random-Play. We collect rollouts in which the pusher moves randomly and may contact the T-block. Action-coordinate tokens supply the conditioning context, with their loss positions masked. Subsequent state tokens provide scene-transition targets conditioned on these random actions. Task-specific training then supervises demonstration sequences, including action coordinates and retained state blocks. Both stages update the same model parameters, combining transition data from random play with expert action supervision.

4 Experiments

We evaluate Spatial Language Modeling through closed-loop pushing, action-conditioned future-state prediction, and ablations of joint training and random-play pretraining. Simulation and real-robot experiments examine how the shared sequence model generates executable actions and predicts the geometric effects of pushing. Implementation details are provided in Appendix B and Appendix C.1.

4.1 Push-T Setup and Evaluation

We discretize the workspace into a 128×128 grid and train a 462M-parameter Transformer using the Qwen2 architecture [45] from scratch. Training takes 5 hours on four NVIDIA A100 GPUs.

Simulation Data. We use the released 200 expert demonstrations from Diffusion Policy [1]. Our model is trained for 200 epochs, with one checkpoint evaluated every 5 epochs. Our model receives rasterized contour coordinates, while the reported simulation baselines use state inputs.

Real-Robot System and Data. A stick mounted on the robot arm serves as the pusher, and an industrial camera observes the tabletop from above (Fig. 1). AprilTags [46] on the T-block and tabletop provide localization, from which the known block geometry is converted to contour coordinates in the calibrated tabletop frame. The arm, camera, and tabletop share this coordinate system.

We collect 40 successful expert demonstrations and 20 random-play episodes. The random-play data contains 5,229 frames in total and is used for action-conditioned state-prediction pretraining, with action-coordinate tokens excluded from the loss. Each collected episode contains no more than 300 steps. We compare against the state-based DP-Coord, DP-Tag, and ACT adaptations described in Appendix C.1. These policy baselines are trained on the expert demonstrations, while our full method also uses random-play data.

Coverage and Success. Let B_t be the set of grid cells occupied by the filled T-block region at step t , and let G be the corresponding set for the target region. Coverage is

$$C_t = \frac{|B_t \cap G|}{|G|}. \quad (3)$$

A test episode is successful and terminates when $C_t \geq 0.95$. Otherwise, evaluation ends after 300 steps (frames), each corresponding to one pusher movement. Coverage is computed from filled regions on the discrete grid, while model inputs represent their contours.

In the real-robot evaluation, each method is tested from 20 initial states, with starting poses manually matched across methods as closely as possible. Our model uses a three-frame observation-history window. The implemented DP baselines use two observation frames and ACT uses one; their input encodings and action-chunk execution settings are specified in Appendix C.1.

For real-robot results, Max is the mean of the maximum coverage reached across the test episodes:

$$Max = \frac{1}{N} \sum_{e=1}^N \max_{t \in \mathcal{T}_e} C_{e,t}, \quad (4)$$

where N is the number of test episodes and $\mathcal{T}_e$ contains the evaluated steps of episode e . Success rate is the fraction of these episodes that reach the coverage threshold within 300 steps.

4.2 Push-T Results

Simulation. Table 2 (left) reports the highest mean coverage across checkpoints (Max) and the mean coverage over the last ten checkpoints (Avg). Our model matches the best Max of 0.95 achieved by DP-C and DP-T, while reaching the highest Avg of 0.93, compared with 0.91 and 0.79, respectively. Its average over the last ten checkpoints is close to its best coverage, showing that strong performance extends beyond the single best checkpoint. These results demonstrate that language-model completion of spatial sequences can produce an effective closed-loop pushing policy.

Table 2: Push-T results. In simulation (left), *Max* is the highest average coverage across evaluated checkpoints and *Avg* is the average coverage over the last 10 checkpoints. Real-robot results (right) report success rate (*SR*) and the mean of episode-level maximum coverage (*Max*; Eq. 4) over 20 episodes per method. *Action-only* deletes future-state blocks and omits random-play pretraining; *w/o P.* omits pretraining but retains joint action/state training.

Methods	Max	Avg	Methods	SR	Max
LG [47]	0.67	0.61	DP-Coord	0.45	0.89
IBC [48]	0.90	0.84	DP-Tag	0.50	0.89
BET [10]	0.79	0.70	ACT	0.65	0.86
DP-C [1]	0.95	0.91	Ours	0.80	0.95
DP-T [1]	0.95	0.79	Action-only	0.05	0.67
Ours	0.95	0.93	Ours w/o P.	0.65	0.92

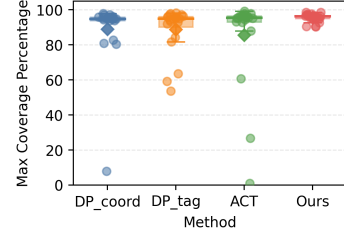


Figure 4: Distribution of episode-level maximum coverage in the real-robot Push-T evaluation.

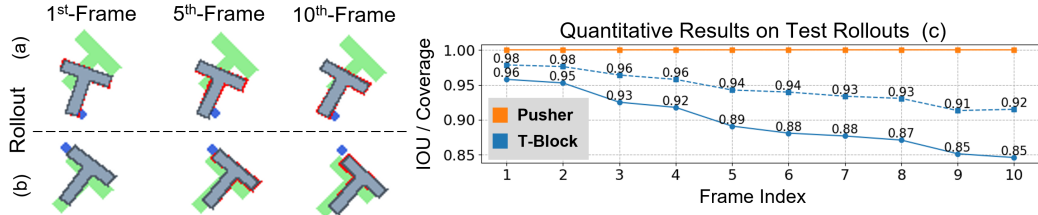


Figure 5: Future-state prediction on Push-T. Given a manually annotated action trajectory, the model predicts successive states using its own previous predictions as context. Reference states are obtained by executing the same actions in the simulator. Subfigures (a) and (b) show two example rollouts, where red pixels indicate the prediction-error regions of the T-block on the rasterized workspace. Subfigure (c) reports the average state prediction accuracy at each successive step (frame) over the test rollouts, where dashed lines denote coverage and solid lines denote IoU.

Future-State Prediction. Figure 5 shows the model predicting T-block translation and rotation along manually annotated action trajectories. Each predicted state supplies the context for the next prediction, and the simulator provides reference states under the same actions. The T-block’s coverage changes from 0.98 at the first frame to 0.92 at the tenth, while IoU changes from 0.96 to 0.85. Although error accumulates over the rollout, substantial geometric overlap remains after ten steps (frames). Together, the examples and frame-wise metrics show that the same spatial language model used for action generation also learns action-conditioned scene transitions.

Real Robot. Our full method achieves a success rate of 0.80, compared with 0.45 for DP-Coord, 0.50 for DP-Tag, and 0.65 for ACT (Tab. 2, right). It also reaches the highest mean episode-level maximum coverage of 0.95, compared with 0.89 for both DP variants and 0.86 for ACT. The language model’s generated coordinates therefore support successful physical execution as well as close geometric alignment with the target.

Figure 4 further shows that our episode-level peak coverage is concentrated around 90%–100%. The baselines include low-overlap episodes, while the full model brings the block close to the target across the tested initial states. Figure 6 provides a complementary view: averaging the maximum-coverage frames yields a clearer T-block outline and fewer ghosting artifacts for our method, indicating more concentrated block positions and orientations. The numerical and visual results together show improvements in task completion and target alignment.

Ablations. We compare two training variants with the full method. *Action-only* deletes the future STATE blocks from the demonstration sequences and uses no random-play pretraining. *Ours w/o P.* retains joint action and future-state training on the demonstrations, also without pretraining. Adding future-state blocks raises success from 0.05 to 0.65 and mean episode-level maximum coverage from 0.67 to 0.92. Joint training thus improves both task completion and geometric alignment. It gives the

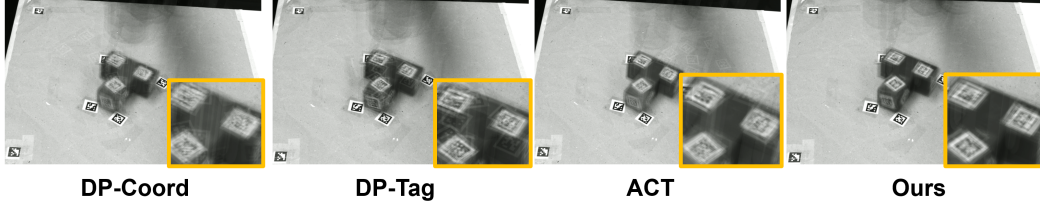


Figure 6: For each method, images are averaged over the test episodes at their respective maximum-coverage frames. This visualization summarizes alignment across episodes.

shared model targets for the actions to take and the scene changes that follow, making future-state prediction a useful component of policy training.

Random-play pretraining further raises success from 0.65 to 0.80 and maximum coverage from 0.92 to 0.95. This stage supplies additional state-transition supervision conditioned on recorded pusher actions, followed by demonstration training for goal-directed control. The two comparisons show complementary benefits from joint action/state training on demonstrations and transition pretraining on random interactions. Both stages use the common spatial vocabulary and update the same autoregressive model.

Inference Throughput. On a single A100 GPU, language-model throughput is approximately 7000 tokens/s during prefilling and 800 tokens/s during decoding. During control, the grammar state machine selects the action-only sequence format, in which each target requires one move-to token and one coordinate token. Generating 100 action targets therefore requires about 200 output tokens, corresponding to about 0.25 s of decoding at the measured throughput. The 20-step execution chunk requires about 40 tokens, or approximately 0.05 s of decoding. This compact output format supports rapid action-chunk generation while the same model is trained jointly on actions and future states.

5 Limitations

Our method has several limitations. First, the real-robot system relies on calibrated AprilTag localization and known object geometry; end-to-end learning from raw sensor observations has not been evaluated. Second, the grammar is manually designed, while grid resolution trades geometric and action precision against vocabulary size and sequence length. Adapting to new entities, action types, or task structures may require changes to the representation. Third, evaluation focuses on planar Push-T, so generalization to other object geometries, manipulation tasks, and three-dimensional scenes remains to be established. Finally, errors accumulate during autoregressive state rollouts, and prediction over longer horizons requires further evaluation. This limitation concerns state prediction; closed-loop control updates its history with newly observed states.

6 Conclusion

We presented Spatial Language Modeling, which expresses scene geometry, goals, action targets, and future states as grammar-structured coordinate sequences. One autoregressive model learns action generation and action-conditioned state prediction through a shared next-token objective. Training combines transition pretraining on random interactions with joint action and state supervision on expert demonstrations. During closed-loop execution, the model generates action targets directly and updates its context using new observations. Experiments demonstrate effective control in simulated and real-world Push-T, while rollouts conditioned on supplied actions capture successive geometric scene changes. Training comparisons show benefits from interleaved action and state sequences, with additional gains from random-play pretraining. These findings support shared spatial modeling as a practical way to connect action-conditioned transition supervision with goal-directed policy learning.

References

- [1] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.
- [2] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *Proceedings of Robotics: Science and Systems*, 2023. doi:10.15607/RSS.2023.XIX.016.
- [3] Physical Intelligence, K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.
- [4] J. Bjorck, F. Castañeda, N. Cherniadev, X. Da, R. Ding, L. Fan, Y. Fang, D. Fox, F. Hu, S. Huang, et al. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.
- [5] J. Jang, S. Ye, Z. Lin, J. Xiang, J. Bjorck, Y. Fang, F. Hu, S. Huang, K. Kundalia, Y.-C. Lin, et al. Dreamgen: Unlocking generalization in robot learning through video world models. In *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pages 5170–5194. PMLR, 2025.
- [6] L. Maes, Q. Le Lidec, D. Scieur, Y. LeCun, and R. Balestriero. Leworldmodel: Stable end-to-end joint-embedding predictive architecture from pixels. *arXiv preprint arXiv:2603.19312*, 2026.
- [7] L. Li, Q. Zhang, Y. Luo, S. Yang, R. Wang, F. Han, M. Yu, Z. Gao, N. Xue, X. Zhu, Y. Shen, and Y. Xu. Causal world modeling for robot control. *arXiv preprint arXiv:2601.21998*, 2026.
- [8] M. J. Kim, C. Finn, and P. Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.
- [9] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *arXiv preprint arXiv:2106.01345*, 2021.
- [10] N. M. Shafiullah, Z. Cui, A. A. Altanzaya, and L. Pinto. Behavior transformers: Cloning k modes with one stone. *Advances in neural information processing systems*, 35:22955–22968, 2022.
- [11] S. Zhou, Y. Du, J. Chen, Y. Li, D.-Y. Yeung, and C. Gan. Robodreamer: Learning compositional world models for robot imagination. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 61885–61896. PMLR, 2024.
- [12] J. Liang, R. Liu, E. Ozguroglu, S. Sudhakar, A. Dave, P. Tokmakov, S. Song, and C. Vondrick. Dreamitate: Real-world visuomotor policy learning via video generation. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pages 3943–3960. PMLR, 2025.
- [13] R. Xu, X. Wang, T. Wang, Y. Chen, J. Pang, and D. Lin. Pointllm: Empowering large language models to understand point clouds. In *European Conference on Computer Vision*, pages 131–147. Springer, 2024.
- [14] Y. Hong, H. Zhen, P. Chen, S. Zheng, Y. Du, Z. Chen, and C. Gan. 3d-llm: Injecting the 3d world into large language models. *Advances in Neural Information Processing Systems*, 36: 20482–20494, 2023.

- [15] S. He, H. Ding, X. Jiang, and B. Wen. Segpoint: Segment any point cloud via large language model. In *European Conference on Computer Vision*, pages 349–367. Springer, 2024.
- [16] H. Zhi, P. Chen, J. Li, S. Ma, X. Sun, T. Xiang, Y. Lei, M. Tan, and C. Gan. Lscenellm: Enhancing large 3d scene understanding using adaptive visual preferences. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 3761–3771, 2025.
- [17] R. Fu, J. Liu, X. Chen, Y. Nie, and W. Xiong. Scene-llm: Extending language model for 3d visual reasoning. In *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 2195–2206. IEEE, 2025.
- [18] K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. Reid, and N. Suenderhauf. Sayplan: Grounding large language models using 3d scene graphs for scalable robot task planning. *arXiv preprint arXiv:2307.06135*, 2023.
- [19] D. Liu, C. Wang, P. Gao, R. Zhang, X. Ma, Y. Meng, and Z. Wang. 3daxisprompt: Promoting the 3d grounding and reasoning in gpt-4o. *Neurocomputing*, 637:130072, 2025.
- [20] C. Nash, Y. Ganin, S. M. A. Eslami, and P. Battaglia. Polygen: An autoregressive generative model of 3d meshes. In *International conference on machine learning*, pages 7220–7229. PMLR, 2020.
- [21] Y. Siddiqui, A. Alliegro, A. Artemov, T. Tommasi, D. Sirigatti, V. Rosov, A. Dai, and M. Nießner. Meshgpt: Generating triangle meshes with decoder-only transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 19615–19625, 2024.
- [22] M. Farazi and Y. Wang. A recipe for geometry-aware 3d mesh transformers. In *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 3290–3300. IEEE, 2025.
- [23] Z. Wang, J. Lorraine, Y. Wang, H. Su, J. Zhu, S. Fidler, and X. Zeng. Llama-mesh: Unifying 3d mesh generation with language models. *arXiv preprint arXiv:2411.09595*, 2024.
- [24] S. Fang, I.-C. Shen, Y. Wang, Y.-H. Tsai, Y. Yang, S. Zhou, W. Ding, T. Igarashi, and M.-H. Yang. Meshllm: Empowering large language models to progressively understand and generate 3d mesh. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14061–14072, 2025.
- [25] T. Chen, S. Saxena, L. Li, D. J. Fleet, and G. Hinton. Pix2seq: A language modeling framework for object detection. *arXiv preprint arXiv:2109.10852*, 2021.
- [26] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.
- [27] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.
- [28] W. Wu, Z. Qi, and L. Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pages 9621–9630, 2019.
- [29] H. Zhao, L. Jiang, J. Jia, P. H. Torr, and V. Koltun. Point transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 16259–16268, 2021.
- [30] Y. Li, Z. Yu, C. Choy, C. Xiao, J. M. Alvarez, S. Fidler, C. Feng, and A. Anandkumar. Voxformer: Sparse voxel transformer for camera-based 3d semantic scene completion. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9087–9098, 2023.

- [31] D. Z. Chen, A. X. Chang, and M. Nießner. Scanrefer: 3d object localization in rgb-d scans using natural language. In *European conference on computer vision*, pages 202–221. Springer, 2020.
- [32] L. Yang, C. Yuan, Z. Zhang, Z. Qi, Y. Xu, W. Liu, Y. Shan, B. Li, W. Yang, P. Li, Y. Wang, and W. Hu. Exploiting contextual objects and relations for 3d visual grounding. *Advances in Neural Information Processing Systems*, 36:49542–49554, 2023.
- [33] J. Wald, H. Dhano, N. Navab, and F. Tombari. Learning 3d semantic scene graphs from 3d indoor reconstructions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3961–3970, 2020.
- [34] R. Zhang, Z. Guo, W. Zhang, K. Li, X. Miao, B. Cui, Y. Qiao, P. Gao, and H. Li. Pointclip: Point cloud understanding by clip. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8552–8562, 2022.
- [35] X. Zhu, R. Zhang, B. He, Z. Guo, Z. Zeng, Z. Qin, S. Zhang, and P. Gao. Pointclip v2: Prompting clip and gpt for powerful 3d open-world learning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 2639–2650, 2023.
- [36] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. *Advances in neural information processing systems*, 29, 2016.
- [37] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry. A papier-mâché approach to learning 3d surface generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 216–224, 2018.
- [38] J. Gao, T. Shen, Z. Wang, W. Chen, K. Yin, D. Li, O. Litany, Z. Gojcic, and S. Fidler. Get3d: A generative model of high quality 3d textured shapes learned from images. *Advances in neural information processing systems*, 35:31841–31854, 2022.
- [39] G. Yang, X. Huang, Z. Hao, M.-Y. Liu, S. Belongie, and B. Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4541–4550, 2019.
- [40] B. Poole, A. Jain, J. T. Barron, and B. Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv preprint arXiv:2209.14988*, 2022.
- [41] C.-H. Lin, J. Gao, L. Tang, T. Takikawa, X. Zeng, X. Huang, K. Kreis, S. Fidler, M.-Y. Liu, and T.-Y. Lin. Magic3d: High-resolution text-to-3d content creation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 300–309, 2023.
- [42] R. Chen, Y. Chen, N. Jiao, and K. Jia. Fantasia3d: Disentangling geometry and appearance for high-quality text-to-3d content creation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 22246–22256, 2023.
- [43] X. Zhou, X. Ran, Y. Xiong, J. He, Z. Lin, Y. Wang, D. Sun, and M.-H. Yang. GALA3D: Towards text-to-3d complex scene generation via layout-guided generative gaussian splatting. *arXiv preprint arXiv:2402.07207*, 2024.
- [44] M. X. Huang, B. J. Chan, and H.-H. Huang. Smartspatial: Enhancing the 3d spatial arrangement capabilities of stable diffusion models and introducing a novel 3d spatial evaluation framework. *arXiv preprint arXiv:2501.01998*, 2025.
- [45] A. Yang, B. Yang, B. Hui, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.
- [46] E. Olson. Apriltag: A robust and flexible visual fiducial system. In *2011 IEEE international conference on robotics and automation*, pages 3400–3407. IEEE, 2011.

- [47] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. *arXiv preprint arXiv:2108.03298*, 2021.
- [48] P. Florence, C. Lynch, A. Zeng, O. A. Ramirez, A. Wahid, L. Downs, A. Wong, J. Lee, I. Mordatch, and J. Tompson. Implicit behavioral cloning. In *Conference on robot learning*, pages 158–168. PMLR, 2022.

A Model Implementation

We employ a 462M-parameter Qwen2 model [45] and train it from scratch. Specifically, the model consists of 7 transformer layers with a hidden dimension of 1536 and 24 attention heads. Rotary Position Embeddings (RoPE) are used for positional encoding. Push-T uses 16,384 coordinate tokens for its 128×128 grid, together with the semantic tokens defining its sequence format. We pack multiple spatial language sentences into training samples with a maximum context length of 13,000.

B Push-T Representation and Execution Details

We provide the grammar definition for the Push-T task [1]. In Push-T, a pusher moves a T-shaped block to a goal region. The observation and action are both represented on a discretized 2D grid and serialized as a Spatial Language Sentence (SLS).

2D Representation and Vocabularies. We discretize the workspace in 2D. Each grid location is encoded as one coordinate token $CB \in \mathcal{Q}_{coord}$. In the simulator, we render the pusher, the T-block, and the goal region, extract their edge pixels, and represent each entity by its contour coordinate tokens.

State (Scene) Description. A Push-T state is composed of three components: the pusher geometry, the T-block geometry, and the goal-region geometry. We define the nonterminal STATE to serialize these components as

$$STATE \rightarrow \text{state pusher } CB^+ \text{ tblock } CB^+ \text{ goal } CB^+.$$

Here, the terminal tokens *state*, *pusher*, *tblock*, and *goal* belong to $\mathcal{Q}_{semantic}$, and each CB^+ is a (sorted) sequence of coordinate tokens that describes the corresponding entity.

Action Description. We represent a pushing action as moving the pusher to a target 2D coordinate. Each such movement is one step and corresponds to one scene frame. Accordingly, the nonterminal MOVE_TO is defined as

$$MOVE_TO \rightarrow \text{move-to } CB \text{ [STATE]}.$$

The optional bracket [STATE] indicates that we may append the next observation state after executing the action, enabling multiple actions to be represented within a single sequence.

The Push-T task is then defined by the 2D production rule:

$$SEQ_{2D} \rightarrow STATE MOVE_TO^*.$$

The sequence format consists of an initial STATE and temporally ordered action blocks, optionally interleaved with resulting states. Training trajectories follow this format. During control, a grammar state machine excludes future STATE blocks; action coordinates are sampled autoregressively and parsed into pusher targets.

Closed-loop execution at test time. At test time, the prefix contains three temporally consecutive STATE-MOVE_TO segments and ends with *move-to* for the next target coordinate. We execute the first 20 predicted *move-to* targets from the action-only continuation, one per step (frame), then update the history with new observations for the next prompt.

Training objectives. Task demonstration training uses next-token prediction over the serialized trajectories. Random-play pretraining masks only the target-coordinate token immediately following each *move-to* marker; the marker and all other tokens remain supervised. Target coordinates remain in the context to condition subsequent state predictions. Action generation and state prediction share model parameters.

Success criterion. Let B_t be the set of filled grid cells occupied by the T-block at time t , and let G be the set of filled grid cells in the target region. Coverage is

$$\text{Coverage}(t) = \frac{|B_t \cap G|}{|G|}.$$

Evaluation episodes terminate successfully at 0.95 coverage or unsuccessfully after 300 steps (frames), corresponding to at most 300 pusher movements. Coverage uses filled grid regions; model inputs use their contours.

For the real-robot results, *Max* averages each episode’s maximum coverage over the 20 test episodes, as defined in Eq. 4. *SR* is the fraction of successful episodes. The simulation table instead reports the highest mean coverage across evaluated checkpoints and the mean over the last 10 checkpoints.

C Full Symbol and Token Definitions

This section complements the brief grammar description in the main paper by explaining the symbols used for Push-T.

Vocabularies. All SLS tokens come from the union of the coordinate vocabulary $\mathcal{Q}_{coord}$ and the semantic vocabulary $\mathcal{Q}_{semantic}$. Coordinate tokens represent discretized locations on the planar grid.

The Push-T semantic tokens are:

$$\mathcal{Q}_{semantic} = \{\text{state, pusher, tblock, goal, move-to}\}.$$

Lexical token: COORD. Each COORD is one vocabulary token identifying one discrete grid cell. The tuple (x, y) is readable notation for that single token ID.

Coordinate block symbol: CB. In the CFG, a coordinate block CB expands to a single COORD token. A repetition CB^+ denotes a non-empty list of coordinates representing the discretized geometry of an entity.

2D task symbols (Push-T). The 2D production SEQ_{2D} defines a Push-T rollout:

$$SEQ_{2D} \rightarrow \text{STATE MOVE_TO}^*.$$

State block: STATE. $\text{STATE} \rightarrow \text{state pusher } CB^+ \text{ tblock } CB^+ \text{ goal } CB^+$ describes one observation, where the coordinate sequences correspond to edge pixels extracted from the rendered pusher, T-block, and goal region.

Action block: MOVE_TO. $\text{MOVE_TO} \rightarrow \text{move-to } CB \text{ [STATE]}$ represents one action that moves the pusher to the target coordinate CB. The optional [STATE] allows appending the next observation after execution.

Canonicalization (coordinate serialization). We sort each entity-specific geometry list CB^+ by a fixed grid order $(x, \text{ then } y)$. Sorting is confined to each entity block.

C.1 Baseline Implementation Details

All three baselines (DP-Coord, DP-Tag, and ACT) operate in the state-based real-world Push-T setting. Their per-frame inputs describe the manipulated T-block together with the pusher-center coordinate $\mathbf{p}_t \in \mathbb{R}^2$. Our SLS input serializes contour coordinates for the pusher, T-block, and goal (§3.2). The baselines use two T-block representations:

- **Object contour coordinates** (used by DP-Coord and ACT): the T-block contour coordinates $X_t = \{\mathbf{x}_{t,i}\}_{i=1}^K$, $\mathbf{x}_{t,i} \in \mathbb{R}^2$, taken from the discretized outer contour used in our

STATE block. The number of occupied cells varies per frame, so each frame is sentinel-padded to a dataset-wide fixed count $K=100$ with $\bar{\mathbf{x}} = (-1, -1)$ marking empty slots.

- **AprilTag corner coordinates** (used by DP-Tag): the $S = 4|\mathcal{T}| = 12$ corner coordinates from the $|\mathcal{T}| = 3$ AprilTags on the top surface of the T-block (four corners each), placed in a fixed slot order with no padding.

For all baselines, the action is the absolute target pusher coordinate $\mathbf{a}_t \in \mathbb{R}^2$, i.e. the same quantity our MOVE_TO production emits. Relative to their default implementations, the observation encoding (and, for ACT, the token construction) is modified; the action heads and training objectives are inherited unchanged. Diffusion Policy uses two observation frames, ACT uses one frame, and our model uses three frames.

DP-Coord / DP-Tag (Diffusion Policy). Both Diffusion Policy baselines use low-dimensional Diffusion Policy [1]: a conditional 1-D U-Net ϵ_θ trained with DDPM and conditioned on the observation as a *global conditioning* vector. We adapt the original Push-T low-dimensional input, which contains nine ground-truth keypoints, to a flattened geometry vector concatenated with the pusher coordinate,

$$\mathbf{o}_t = [\text{vec}(X_t) \parallel \mathbf{p}_t] \in \begin{cases} \mathbb{R}^{2K+2}, & \text{DP-Coord (contour coords)}, \\ \mathbb{R}^{2S+2}, & \text{DP-Tag (tag coords)}, \end{cases} \quad (5)$$

where $\text{vec}(\cdot)$ stacks the coordinates in their fixed order. Each dimension is normalized to $[-1, 1]$ (limits normalizer). DP-Coord feeds the sentinel-padded contour through the fixed-width global-conditioning path with no explicit padding mask. Stacking $n_o=2$ observation steps yields the conditioning $\mathbf{c}_t = [\mathbf{o}_{t-1}; \mathbf{o}_t]$ passed to ϵ_θ . The output is the default Diffusion Policy receding-horizon scheme: denoise an action trajectory $\hat{\mathbf{a}}_{t:t+H} \in \mathbb{R}^{H \times 2}$ of horizon $H=16$ and execute the first $n_a=8$ steps before re-planning.

ACT. The default ACT [2] (DETR-VAE) feeds a ResNet image feature map ($H' \times W'$ tokens) as the cross-attention memory of its transformer decoder, plus a proprioception token from the agent state and a CVAE latent token. For our state-based task, we replace the CNN feature tokens with *per-coordinate* tokens over the object contour coordinates. Each coordinate is lifted to the hidden dimension d by a single shared learnable linear map $\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^d$ and combined with a learnable per-slot positional embedding $\mathbf{e}_i$,

$$\mathbf{m}_{t,i} = \phi(\mathbf{x}_{t,i}) + \mathbf{e}_i, \quad i = 1, \dots, K, \quad (6)$$

giving K memory tokens. The pusher coordinate is encoded by the *same* kind of learnable linear that default ACT applies to the agent state, $\mathbf{m}_t^{\text{prop}} = \psi(\mathbf{p}_t)$, $\psi : \mathbb{R}^2 \rightarrow \mathbb{R}^d$, and prepended together with the CVAE latent token. Padded contour slots are excluded from attention through a key-padding mask $\mu_{t,i} = \mathbf{1}[\mathbf{x}_{t,i} = \bar{\mathbf{x}}]$, so a single model serves the variable-length contour. The CVAE encoder, the latent prior, and the $L_1 + \text{KL}$ training objective are identical to default ACT. The decoder uses $H=16$ learnable query tokens and a linear action head to emit the chunk $\hat{\mathbf{a}}_{1:H} \in \mathbb{R}^{H \times 2}$; at inference the latent is fixed to its prior mean ($\mathbf{0}$). ACT conditions on a single observation step ($n_o=1$) and executes the first $n_a=8$ of the 16 predicted actions per re-plan.