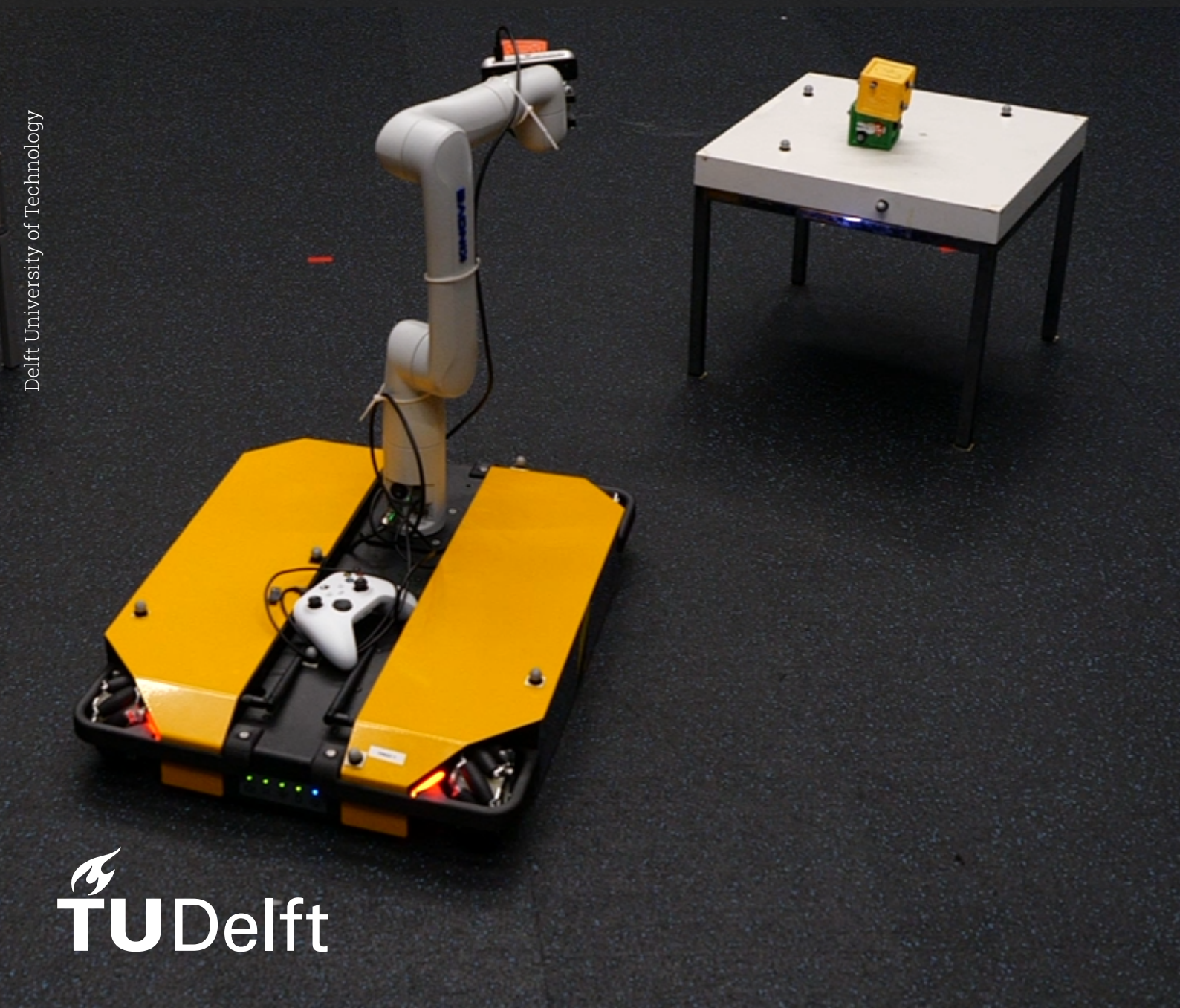
Task and Motion Planning with Flow Matching for Coupled Mobile Manipulation

Zuleika M. Redondo García



Delft University of Technology

TU Delft

Task and Motion Planning with Flow Matching for Coupled Mobile Manipulation

by

Zuleika M. Redondo García

Daily Supervisor: Andreu Matoses Giménez
Main Supervisor: Javier Alonso-Mora
Faculty: Faculty of Mechanical Engineering, Delft

TU Delft

Acknowledgements

I would first like to thank my main supervisor, Javier Alonso-Mora, who has become an inspiration to me, not only through his classes but also through the opportunity to work with him as a TA. It has been a great pleasure to learn from him.

I would also like to thank my daily supervisor, Andreu Matoses-Gimenez, who has been there through every step of this thesis: encouraging me, listening to my ideas, and helping me shape them into something we could build together, culminating in a work that I feel incredibly proud of. This could not have happened without you.

A special thank you also goes to Saray Bakker, who guided me through working with the robot and helped me during those difficult moments when I could no longer figure out what else to try to make the hardware work.

To my friends back in Spain, Iván, Alba, Paco, Isabel, and Laura: thank you for always being there and for making me feel like the most supported person, even from so many kilometres away. Thank you for making every return home feel as if I had never left.

During these two years, I have met so many beautiful people, and there are memories that will stay with me for the rest of my life. For the nights at RoboHouse, the kayaking, the burger days, and everything in between.

Thank you, Robert, for the weirdest and deepest conversations about the most random facts and your English corrections. Thank you, Kaj, for making every moment at CoR more fun, for our runs filled with deep conversations, and for your caffeine-and-anxiety-fuelled zoomies. Thank you, Juan, for the whisky talks, the bonding walks, and the most amazing barbecues — my Colombian brother.

And Harini, who over the past year has become one of those people I would do anything with. Thank you for being there and for becoming such an important part of this chapter of my life.

Last but certainly not least, I want to thank my family: my mom, my dad, and my sister, Elara. Thank you for always supporting me, both academically and personally; for listening to me, being patient with me, celebrating my successes, and being there when things were difficult. Thank you for continuing to teach me things about life that I still did not know, and I hope that never changes.

For all these reasons, I finish this chapter happy, proud, and deeply grateful.

Zuleika M. Redondo García
Delft, October 2026

Abstract

Task and motion planning for mobile manipulation requires jointly reasoning about high-level action sequences and the continuous motions needed to execute them. Classical TAMP methods typically rely on hand-designed task models and search over candidate symbolic and continuous solutions, which can become computationally expensive and limit rapid replanning when the environment changes. Generative approaches offer an alternative by learning planning behaviour directly from demonstrations, but jointly learning symbolic and continuous plans is particularly challenging when demonstrations must be collected through real-world teleoperation and are therefore limited in number.

This thesis introduces a Hybrid Flow Matching planner that jointly generates a discrete symbolic task plan and continuous trajectories for the mobile base and end-effector of a mobile manipulator. The method builds on a hybrid discrete–continuous generative planning by replacing the diffusion denoising and training elements with flow matching and introducing the following additional components. Modality embeddings are introduced to distinguish the two prediction modalities; plan-suffix augmentation increases the effective number of training samples by extracting several intermediate demonstrations from the original ones and exposes the model to those intermediate task states; and remasking allows low-confidence discrete predictions to be reconsidered during inference. The symbolic plan additionally defines action boundaries at which updated observations can be incorporated for closed-loop replanning.

The method is trained on 109 human-teleoperated demonstrations of a three-cube stacking task and evaluated on 12 validation demonstrations. Hybrid Flow Matching achieves a symbolic-plan validity of 55%, with task-completion rates of 30% in open-loop and 41% in closed-loop execution, and a median end-effector RMSE of $0.255\,\mathrm{m}$. Hybrid Diffusion trained on the same suffix-augmented data achieves higher symbolic validity at 62%, but substantially lower task-completion rates of 6% and 9%, respectively, with an RMSE of $0.710\,\mathrm{m}$. A motion-only Flow Matching baseline reaches 39% and 52% task completion, but does not provide an explicit symbolic plan.

Real-system experiments further provide evidence that the jointly generated symbolic and continuous plans can be executed on the real mobile manipulator and replanned after execution failures. Performance nevertheless degrades for under-represented long-horizon configurations, indicating that limited generalisation remains the main challenge. Overall, the results demonstrate the feasibility of joint symbolic and continuous generation with flow matching in a data-limited mobile-manipulation setting, while highlighting the trade-off between explicit task-level planning and continuous execution performance.

The source code for this work is available at:
`https://github.com/AndreuMatoses/hybrid-flow-planning`

Contents

Acknowledgements i

Abstract ii

Nomenclature iv

1 Introduction 1

2 Related Work 3
- 2.1 Mobile manipulation planning . . . 3
- 2.2 Long-horizon planning in robotics . . . 3
- 2.3 Generative models for task and motion planning . . . 3
- 2.4 Hybrid Diffusion Planner . . . 4
- 2.5 Data efficiency and closed-loop replanning in robot learning . . . 6

3 Problem Definition 7
- 3.1 Goal . . . 7
- 3.2 Formulation . . . 7
- 3.3 Requirements and Challenges . . . 8

4 Hybrid Flow Matching Method 9
- 4.1 Construction of corrupted inputs . . . 9
- 4.2 Input representation and conditioning . . . 10
  - 4.2.1 Positional and Modality embeddings . . . 11
  - 4.2.2 Generative-time embeddings . . . 11
  - 4.2.3 Observation embeddings . . . 11
- 4.3 Training Procedure . . . 12
  - 4.3.1 Padding . . . 12
  - 4.3.2 Loss calculation . . . 13
- 4.4 Prediction Procedure . . . 14
- 4.5 Transformer pipeline . . . 15
- 4.6 Data augmentation . . . 16

5 Results 19
- 5.1 Environment . . . 19
- 5.2 Dataset Generation . . . 20
  - 5.2.1 Arm Joint-State Collection . . . 22
- 5.3 Comparisons . . . 23
  - 5.3.1 Discrete loss weight ablation . . . 24
  - 5.3.2 Model Architecture Ablations . . . 24
- 5.4 Real-System Deployment . . . 27
- 5.5 Real-System Evaluation . . . 28

6 Conclusions 31
- 6.1 Conclusion . . . 31
- 6.2 Future work . . . 32

References 34

A Training and Inference Pseudocode 37

Nomenclature

Symbols

$\mathbf{o}_0$	Observation of the robot and environment at the start of the current planning query.
$\mathbf{x}^c$	Continuous state used during the generative process.
$\mathbf{x}^d$	Discrete state used during the generative process.
$\mathbf{z}^c$	Clean continuous trajectory.
$\mathbf{z}^d$	Clean discrete symbolic sequence.
$\mathbf{x}^c_{t_c}$	Continuous trajectory at generative time t_c.
$\mathbf{x}^d_{t_d}$	Discrete sequence at generative time t_d.
t^c	Continuous generative time.
t^d	Discrete generative time.
$\boldsymbol{\epsilon}$	Gaussian source sample, $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
$\widehat{\mathbf{v}}$	Continuous velocity field predicted by the model.
$\boldsymbol{\ell}$	Predicted categorical logits for the discrete sequence.
$\mathcal{V}$	Vocabulary of discrete symbolic tokens.
H	Continuous trajectory horizon.
F_c	Number of continuous features per waypoint.
N_d	Discrete sequence horizon or number of token positions.
N_o	Number of parameter groups in the observation data.
M	Number of symbolic actions in a plan.
K	Number of inference steps.
λ_d	Weight of the discrete loss in the combined training objective.
$\mathcal{L}^c$	Continuous Flow Matching loss.
$\mathcal{L}^d$	Discrete categorical reconstruction loss.
ε	Small positive constant used for numerical stability and to avoid evaluating the continuous generative process exactly at its endpoint.
$\mathbf{a}$	Sequence of high-level symbolic actions.
a_k	k-th high-level symbolic action.
b	Demonstration index.
M_b	Number of symbolic actions in demonstration b.
θ	Learnable model parameters.

Abbreviations

TAMP	Task and Motion Planning.
FM	Flow Matching.
HDP	Hybrid Diffusion Planner.
Hybrid FM	Hybrid Flow Matching planner proposed in this thesis.
EE	End-effector.
EOS	End-of-sequence token.
MASK	Token representing a hidden discrete symbol.
MLP	Multi-layer perceptron.
ROS	Robot Operating System.
VR	Virtual reality.
RMSE	Root mean square error.
DoF	Degrees of freedom.

1

Introduction

Mobile manipulation tasks such as tidying a room or assembling a structure require a robot to decide what to do and how to move. To stack three cubes on a target table (Fig. 1.1), a robot has to choose which cube to pick first, from which side of which table, in which position to place it. It also has to produce the base and arm motion for each choice. The two levels depend on each other: a cube can only be reached from certain sides of a table, and a covered cube cannot be picked. Task and motion planning (TAMP) produces both the action sequence and the motion that performs it while keeping them consistent [17].

Classical TAMP methods search over symbolic action sequences and call a motion planner to check each candidate [20, 35, 37, 16]. They need a hand-written model of the task: predicates, action operators, and samplers for continuous parameters such as grasps and base placements. The search is also expensive, which limits replanning when the scene changes. Learning can guide the search [8, 11] or distill planner solutions into policies [9], but both still depend on the planner and its model.

Generative models trained on demonstrations take a different route. The task knowledge enters through the demonstrations, so no planning domain is written by hand. A plan then costs one generation pass with a fixed number of network evaluations instead of a search over candidate plans. Diffusion and flow matching policies capture the multimodal distribution of demonstrated motion and generate whole trajectories [19, 7, 24, 5]. On long tasks with several distinct decisions, however, the model blends different behaviours and the trajectory commits to none of them. Høeg et al. [18] address this with a discrete and a continuous diffusion process in one network that generate a symbolic plan together with the trajectory, and show that the joint generation outperforms trajectory-only generation.

The space of task and motion plans is combinatorial, as each possible action sequence must be grounded in the geometry of the scene. Prior joint symbolic and continuous generative planners have therefore relied on hundreds to thousands of demonstrations per task, often generated by scripted experts on fixed-base manipulators and executed open-loop [18]. For real mobile manipulators, however, demonstrations are typically collected through teleoperation and cannot be generated at that scale. This limited coverage raises three challenges: generalisation to unseen scenes and plans, generation of variable-length symbolic plans, and error accumulation during open-loop execution.

We address these challenges with a hybrid flow matching planner for mobile manipulation trained on about one hundred real demonstrations. Given an observation of the scene, a single network simultaneously generates a discrete symbolic plan and the continuous trajectories of both the mobile base and the end-effector. To improve data efficiency, we augment each demonstration with its suffixes, the remaining plan and trajectory after each of its actions, which also enables the planner to replan after every action.

We evaluate the method on a mobile manipulator building a three-cube tower from 109 teleoperated demonstrations. Generated plans are validated with a symbolic model, executed in kinematic simulation, and evaluated on the real robot. We compare against motion-only flow matching and the hybrid

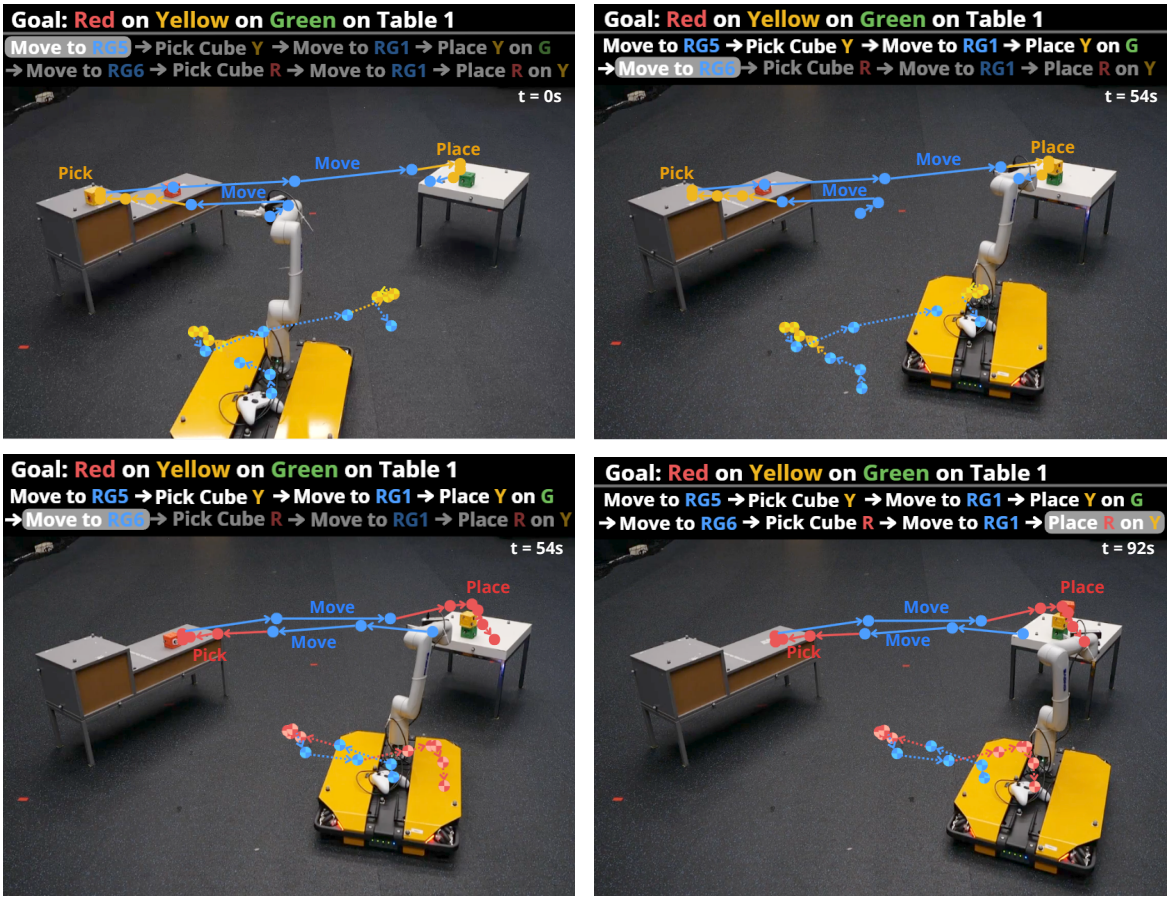


Figure 1.1: The planner on the real robot building the tower R on Y on G. Four frames of one run: the start and end of the first half of the task (top) and of the second half (bottom). Each frame shows the plan the planner generated from that scene, with the action in progress highlighted, and the generated waypoints of the base and end-effector. The planner generates the waypoints of the whole task; for readability only those of the half shown are drawn.

diffusion planner of Høeg et al. [18], and ablate the proposed components.

The contributions of this thesis are:

- A Hybrid Flow Matching planner for coupled mobile manipulation that jointly generates a discrete symbolic plan and continuous robot motion, incorporating modality embeddings and remasking during discrete generation.
- A plan-suffix augmentation method that extracts additional training samples from intermediate stages of task and motion demonstrations, increasing the available data and providing more information about the environment at symbolic action boundaries without collecting additional demonstrations (Section 4.6).
- An evaluation on human-generated demonstrations against motion-only flow matching and hybrid diffusion [18], with ablations and a deployment on the real robot (Section 5.3).

2

Related Work

2.1. Mobile manipulation planning

Motion planning for mobile manipulators commonly follows either a decoupled or a coupled approach. Decoupled methods plan separately for the mobile base and manipulator, introducing a sequential structure in which each subsystem contributes to task completion. This decomposition divides the planning problem into smaller subproblems that can be addressed using specialised and efficient techniques, and has demonstrated reliability in complex tasks [29]. However, requiring the base to reach a stable configuration before arm motion, or vice versa, limits simultaneous coordination. This can reduce flexibility and negatively affect operational speed and energy efficiency [33, 41].

Coupled methods instead treat the mobile manipulator as a single kinematic system, planning jointly over the base and arm configurations [29, 21, 6]. Considering the complete configuration space enables more coordinated behaviour, but also increases the complexity of the planning problem. In particular, the larger number of degrees of freedom and kinematic redundancy make joint planning computationally demanding [33, 29].

2.2. Long-horizon planning in robotics

Long-horizon tasks require reasoning over extended action sequences, during which errors in state estimation, dynamics prediction, and execution can accumulate and lead to task failure. Hierarchical planning addresses this challenge by decomposing complex objectives into smaller subgoals or tasks [3, 32]. This decomposition allows planning to operate at different levels of abstraction while retaining the need to connect high-level decisions with executable actions.

For example, [13] uses two large language models (LLMs), with one acting as a planner and the other as an executor. The planner generates a high-level task sequence, while the executor decomposes it into actions that account for the geometry of the surroundings. Although this separation improves performance over the single-model approaches considered in that work, repeated computation after each action increases planning time.

A skill-based Task and Motion Planning (TAMP) framework is presented in [32]. Each predefined skill describes a mode of interaction with the environment and its constituted objects. Planning involves selecting a sequence of skills and optimising their continuous parameters, such as contact locations and end-effector poses. Linking task selection with motion-level parameterisation helps address incompatibilities between intermediate decisions and the final goal. These approaches illustrate the importance of coordinating symbolic decisions and continuous motions over the full task horizon.

2.3. Generative models for task and motion planning

Generative models, particularly diffusion models, have been explored for behaviour generation and long-horizon planning. They offer an alternative to policy-learning approaches with limited long-horizon

reasoning and classical TAMP methods that depend heavily on symbolic representations [25]. However, generalisation remains a challenge, motivating combinations of diffusion models with other planning and learning frameworks [23, 14].

Janner et al. [19] formulate planning as the generation of complete state–action trajectories through iterative denoising. Rather than learning a one-step dynamics model and subsequently optimizing over it, the method learns a distribution over trajectories. A separate return model guides sampling towards high-reward behaviour, while conditioning incorporates the current state and, optionally, a goal. AdaptDiffuser extends this approach by generating synthetic trajectories to increase the training dataset [22]. Candidate trajectories of unseen scenarios are generated using reward guidance and filtered before being used as additional training data. Repeating this process improves generalisation, although difficulties remain in high-dimensional settings.

Diffusion models have also been used as expressive policy representations in offline reinforcement learning. Wang et al. [40] parameterise the policy as a conditional denoising diffusion model capable of representing multimodal action distributions. Their objective balances consistency with the fixed training dataset and preference for out-of-dataset high-value actions. This combines policy regularization with policy improvement, although increasing the number of denoising steps introduces a trade-off between policy quality and computational cost.

For visuomotor control, Diffusion Policy [7] generates action sequences conditioned on visual observations and robot state information. Visual features and additional state inputs condition the denoising process, which progressively transforms noisy actions into a predicted sequence. A receding-horizon strategy regenerates future actions from updated observations, supporting multimodal behaviour and recovery from deviations. Nevertheless, iterative inference introduces computational overhead, while longer prediction horizons may reduce responsiveness, therefore, balancing these two parameters is key. However, as a behaviour-cloning approach, the method also remains dependent on the behaviours represented in the training demonstrations.

Another direction combines LLM reasoning with diffusion-based generation. PlanDiffuser [30] uses scene information and task prompts to generate a high-level action sequence through an LLM. Diffusion-based inpainting then produces intermediate images representing the planned changes in the scene, and a goal-conditioned policy translates these visual goals into control actions. The process is applied iteratively until the goal is reached. Although the approach improves long-horizon performance over the evaluated end-to-end and open-loop LLM planners, errors in perception, reasoning, or image generation may propagate through the pipeline. Furthermore, generated subgoals may not satisfy the robot's physical or kinematic constraints.

Diffusion models can also support spatial reasoning. Ji et al. [42] use LLM-derived knowledge about object–room relationships to guide semantic map generation for object navigation. Starting from partial observations, the method estimates likely object locations in both observed and unexplored regions. The resulting belief map guides exploration towards promising areas. Although diffusion is triggered selectively to limit delays, the additional computation remains a limitation.

Closer to manipulation planning, Generative Skill Chaining [26] combines separate diffusion models for individual skills. Each model represents a distribution over input states, action parameters, and output states. Given a predefined skill sequence, the method refines intermediate states and parameters to promote both local feasibility and consistency with the final goal. In particular, the output of one skill must be compatible with the next skill in the sequence. The framework also allows geometric constraints to be incorporated at inference time without retraining. However, it requires the symbolic sequence to be supplied in advance and therefore addresses skill parameterisation without autonomously selecting the skills or their order.

2.4. Hybrid Diffusion Planner

The Hybrid Diffusion Planner (HDP), introduced by Høeg et al. [18], addresses this separation by jointly generating discrete symbolic plans and continuous trajectories. Its shared denoising network infers both representations, allowing the symbolic task sequence and its associated motion parameters to be generated together. This joint formulation provides the methodological basis and baseline for this

thesis.

The continuous component uses a Denoising Diffusion Probabilistic Model (DDPM) to generate trajectory parameters. During training, isotropic Gaussian noise corrupts the continuous plans, and the network learns to reverse this corruption. Iterative denoising then transforms noise into a trajectory that follows the learned continuous plan distribution.

The discrete component uses the MD4 masked diffusion framework [31] to model symbolic sequences. Instead of perturbing tokens with Gaussian noise, the forward process replaces them with an absorbing `MASK` token. The network is trained to predict the original symbols through a cross-entropy-based objective. Its predictions define categorical distributions over the task vocabulary, from which tokens are sampled during reconstruction. This provides a discrete counterpart to the denoising process used for continuous trajectories.

HDP is trained on paired continuous and discrete plans, with separate noise schedules controlling corruption in the two modalities. A shared denoising network processes both corrupted representations, conditioned on an initial observation that may consist of images or lower-level state information. Different corruption levels allow the modalities to complement one another during reconstruction. The training objective combines the continuous DDPM loss with a weighted discrete masked diffusion loss, balancing the optimisation of the two components.

At inference time, the continuous trajectory is initialised with noise and the symbolic sequence is fully masked. Conditioned on a new observation and specified planning horizons, the model iteratively refines both representations through reverse diffusion. The resulting output consists of a symbolic plan and its corresponding continuous trajectory, as illustrated in Figure 2.1. Symbolic planning and motion generation therefore occur within the same denoising process, rather than in successive planning stages.

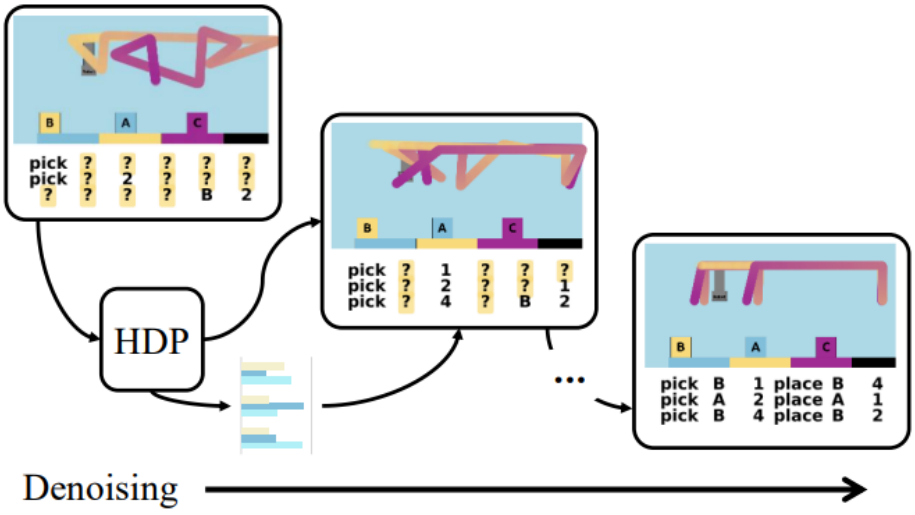


Figure 2.1: Joint denoising of continuous trajectories and discrete symbolic plans in the Hybrid Diffusion Planner [18].

HDP reports strong performance relative to the evaluated diffusion-based and motion-only planning baselines in simulation and on real robotic platforms [18]. It also supports conditional sampling and can operate on image observations, although these introduce additional representational complexity and a moderate performance decrease. Its relevance to this thesis lies particularly in the simultaneous treatment of symbolic and continuous planning: the high-level sequence is inferred alongside the motion trajectory, without requiring a predefined skill sequence.

Our work builds on this hybrid formulation, replacing diffusion with flow matching and studying the more data-limited setting of teleoperated mobile-manipulation demonstrations.

2.5. Data efficiency and closed-loop replanning in robot learning

Several approaches improve data efficiency by extracting additional learning signal from existing experience or by incorporating prior structure into the policy. Hindsight Experience Replay (HER) relabels previously collected transitions with alternative goals that were actually achieved during execution, allowing unsuccessful attempts with respect to the original goal to still provide useful learning signals for goal-conditioned policies [2]. Different relabeling strategies select achieved goals from the final state, future states, or other states observed during training, with goals achieved later in the same trajectory providing particularly useful supervision. In contrast, equivariant policies exploit known geometric structure rather than relabeling experience. Equivariant Diffusion Policy incorporates planar $SO(2)$ symmetry into the denoising model while supporting full 6-DoF control, improving sample efficiency and generalisation by enforcing transformations that are known a priori to preserve the underlying manipulation problem [39]. These approaches illustrate two complementary mechanisms for improving learning from limited data: generating additional supervision from existing experience and embedding task symmetries directly into the model.

A complementary direction addresses the accumulation of execution errors over long horizons through action chunking and closed-loop replanning. Rather than predicting a single action independently at every timestep, Action Chunking with Transformers (ACT) predicts a fixed-length sequence of k future actions, reducing the effective decision horizon and maintaining temporal consistency [43]. To retain responsiveness to new observations, overlapping action chunks can additionally be combined through temporal ensembling, allowing predictions made at successive timesteps to contribute to the executed action. Diffusion Policy similarly predicts sequences of future actions, but explicitly adopts receding-horizon control: only an action horizon from the predicted sequence is executed before the environment is observed again and a new sequence is generated [7]. This provides a trade-off between temporal consistency, obtained by predicting coherent action sequences, and responsiveness to deviations that arise during execution. Such closed-loop execution is particularly relevant for long-horizon manipulation, where small prediction or control errors can accumulate if a complete trajectory is executed open-loop.

The benefits of action chunking and additional co-training have also been demonstrated in real-world mobile manipulation. Mobile ALOHA applies action-chunking policies to long-horizon bimanual mobile manipulation tasks and supplements relatively small sets of task-specific mobile demonstrations with co-training data collected on the related static ALOHA platform [15]. This combination improves data efficiency and helps the learned policies recover from errors arising during extended mobile manipulation tasks, where inaccuracies in both manipulation and base motion can accumulate over time.

Our work draws on both data-efficient training and closed-loop replanning. To increase the amount of supervision available from the limited demonstration set, we use plan-suffix augmentation, which transforms each demonstration into multiple training samples corresponding to different intermediate stages of the task. During execution, the planner repeatedly conditions on the updated state of the environment rather than committing to the complete initially generated trajectory. In contrast to approaches that replan after a fixed temporal action horizon, our method performs replanning at symbolic action boundaries.

3

Problem Definition

3.1. Goal

The goal is considered successfully accomplished once the robot has finished executing the predicted plan and the cubes form a tower with the green cube at the base, the yellow cube in the middle, and the red cube on top. This tower must be located on a specific table, *Table 1*.

3.2. Formulation

A task is defined by an observation of the initial state of the robot and environment, denoted by $\mathbf{o}_0$. In this work, the task goal is fixed across all demonstrations and evaluation tasks: the red cube must be placed on the yellow cube, the yellow cube on the green cube, and the resulting tower must be located on Table 1. Therefore, the goal is not provided to the model as an additional conditioning variable.

A solution consists of two aligned components: a discrete symbolic plan and a continuous robot trajectory. Let the symbolic plan contain M high-level actions,

$$\mathbf{a} = [a_1, a_2, \ldots, a_M], \tag{3.1}$$

where each a_k represents a high-level action together with its corresponding arguments. In this work, the action types are $\mathrm{Move\_to}(\mathrm{region})$, $\mathrm{Pick}(\mathrm{cube})$, and $\mathrm{Place}(\mathrm{cube}, \mathrm{support})$, where the support may be either a cube or a table. Each action is associated with a segment of the continuous trajectory containing the waypoints required to execute it.

For learning, the symbolic plan is represented as a clean sequence of discrete tokens,

$$\mathbf{z}^d = \left(z_1^d, \ldots, z_{N_d}^d\right), \qquad z_j^d \in \mathcal{V}, \tag{3.2}$$

where $\mathcal{V}$ denotes the discrete vocabulary and N_d is the sequence length. The vocabulary contains action types, cube identifiers, table identifiers, approach-region identifiers, and the End-Of-Sequence token *EOS*.

The corresponding clean continuous trajectory is represented as

$$\mathbf{z}^c = [\mathbf{z}_1^c, \ldots, \mathbf{z}_H^c] \in \mathbb{R}^{H \times F_c}, \tag{3.3}$$

where H denotes the trajectory horizon and F_c is the number of continuous features at each waypoint. Each waypoint contains the robot state required for execution, including the mobile-base pose, end-effector pose, and gripper state.

The discrete actions and continuous trajectory are aligned through action boundaries. For each action a_k, a corresponding subset of consecutive waypoints in $\mathbf{z}^c$ describes the continuous motion executed

while that action is active. This alignment is later used to construct the plan-suffix training samples described in Section 4.6.

The planning problem is therefore formulated as learning the conditional joint distribution

$$p_\theta \left(\mathbf{z}^d, \mathbf{z}^c \mid \mathbf{o}_0 \right), \tag{3.4}$$

from which the symbolic plan and its corresponding continuous trajectory are generated jointly.

A generated symbolic plan is considered valid if its action sequence can be executed from the initial symbolic state and results in the desired task goal. For evaluation, an abstraction ϕ maps the initial observation to a symbolic state, and a transition function f maps an applicable action and the current symbolic state to the next state:

$$\mathbf{s}_1 = \phi(\mathbf{o}_0), \qquad \mathbf{s}_{k+1} = f(\mathbf{s}_k, a_k). \tag{3.5}$$

The plan is valid if the state obtained after the final action satisfies the fixed task goal. A generated solution is considered successful if execution of the continuous trajectory results in the desired physical cube configuration.

During closed-loop execution, only the continuous trajectory segment associated with the current symbolic action is executed before a new observation is acquired and the model is queried again. Consequently, the same planning problem can be solved from intermediate task states during replanning.

3.3. Requirements and Challenges

Several assumptions and challenges are made about the problem setting.

- The environment is fully observable, meaning that geometric constraints and key element poses are assumed to be known. In a real-world deployment, these element poses would typically be obtained through a perception pipeline; however, due to time constraints, they are assumed to be perfectly defined in this work using a Motion Capture System.
- All objects in the environment are assumed to be rigid, i.e., they do not deform under contact with the robot. However, only some of them are movable: the cubes, whereas the tables remain in the same position from start to end.
- The robot is assumed to be able to hold only one object at a time, i.e., one cube at a time.

Some requirements for the project are:

- The planning must contain two main parts: a task plan with a sequence of actions and a motion plan consisting of a series of parameters that define the movement of the robot throughout execution.
- The continuous motion component must precisely define trajectories for the mobile manipulator's base and arm, using either the end-effector pose or the robot joint configuration.
- The mobile manipulator will move around its environment, accounting for the different geometric constraints defined, avoiding known obstacles, and accomplishing the goal defined at the beginning of this section.
- Rather than treating base navigation and arm manipulation as a sequence of independent tasks, the system should generate simultaneous, coupled motions through a whole-body execution framework.

4

Hybrid Flow Matching Method

4.1. Construction of corrupted inputs

Training the hybrid flow-matching model requires the construction of corrupted versions of both the discrete task sequence and the continuous trajectory. Each modality is associated with a generative time that determines its corruption level, with (t=0) corresponding to the noisy distribution and (t=1) to the clean data.

Following [18], the discrete and continuous times, t_d and t_c, are initially sampled independently for each demonstration in a batch. Both are drawn uniformly from $[0, 1-\varepsilon]$, where $\varepsilon > 0$ avoids evaluating the continuous target vector field at its singular endpoint. To emphasize examples in which both modalities are heavily corrupted, each demonstration has a probability of (0.25) of having both times replaced by a shared value sampled uniformly from $[0, 0.2]$.

The corrupted inputs are then constructed as follows:

- **Discrete sampling:** Each symbol in the task vocabulary is represented by an integer token, and an additional `MASK` token represents a hidden symbol. Given a clean sequence z^d and time $t_d \in [0, 1]$, each position is independently retained or masked using Bernoulli sampling [34]. Specifically, a retention indicator $r_i \sim \mathrm{Bernoulli}(t_d)$ determines whether the token at position i is retained ($r_i = 1$) or masked ($r_i = 0$), yielding

$$x^d_{t_d,i} = \begin{cases} z^d_i, & \text{with probability } t_d, \\ \mathrm{MASK}, & \text{with probability } 1 - t_d. \end{cases} \tag{4.1}$$

  where i denotes a position in the sequence. Thus, the path transitions from a fully masked sequence at $t_d = 0$ to the clean sequence at $t_d = 1$.

- **Continuous sampling:** Following the flow-matching framework of [24], we use a straight-line conditional path between Gaussian noise and a clean trajectory:

$$x^c_{t_c} = t_c z^c + (1 - t_c)\epsilon, \qquad \epsilon \sim \mathcal{N}(0, I), \tag{4.2}$$

  where z_c is the clean continuous trajectory and ϵ is an independent standard isotropic Gaussian sample of the same dimensionality. At $t_c = 0$, the input consists entirely of noise; at $t_c = 1$, it equals the clean trajectory.

The model receives the corrupted continuous trajectory and discrete sequence, their corresponding generative times, and the initial observations of the demonstration. It predicts the continuous velocity at the current trajectory sample and a probability distribution over clean token values at each discrete sequence position. These predictions are used to compute the continuous and discrete training losses, respectively.

4.2. Input representation and conditioning

The model operates on heterogeneous inputs consisting of continuous trajectory parameters, discrete action sequences, observations, and generative-time information. Before being processed by the transformer, these inputs are projected into a common embedding space of dimension e_d. This allows the model to jointly process information with different structures and dimensionalities.

Continuous and discrete input embeddings

The continuous and discrete components of a trajectory are initially embedded separately. Let the continuous trajectory be represented as

$$\mathbf{X}^c \in \mathbb{R}^{H \times F_c}, \tag{4.3}$$

where H denotes the trajectory horizon and F_c is the number of continuous features at each trajectory step. The continuous trajectory can therefore be interpreted as a two-dimensional array whose axes correspond to trajectory steps and features.

Following a patch-based representation similar to that used in Vision Transformers [10], the continuous input is divided into patches and projected into the embedding space using a convolutional layer. In our implementation, patches have size $1 \times F_c$, where F_c is the dimensionality of a continuous waypoint. Each patch, therefore, contains one complete waypoint, resulting in one continuous token per temporal step. More generally, for a patch size $P_H \times P_F$, the number of continuous tokens is

$$N_c = \left\lfloor \frac{H - P_H}{S_H} + 1 \right\rfloor \left\lfloor \frac{F_c - P_F}{S_F} + 1 \right\rfloor, \tag{4.4}$$

where S_H and S_F denote the convolution strides along the trajectory and feature dimensions, respectively. When the stride is equal to the patch size and the dimensions are exactly divisible, this expression simplifies to

$$N_c = \frac{H}{P_H} \frac{F_c}{P_F}. \tag{4.5}$$

In the configuration used in this work, $P_H = 1$ and $P_F = F_c$, such that each token corresponds to one full waypoint. Therefore,

$$N_c = H \tag{4.6}$$

Finally, the patch projection produces the continuous token representation as

$$\mathbf{E}^c_{\text{patch}} = \text{Flatten}\left(\text{Conv}\left(\mathbf{X}^c\right)\right) \in \mathbb{R}^{N_c \times e_d}. \tag{4.7}$$

On the other hand, the discrete trajectory is represented as a sequence of categorical variables,

$$\mathbf{x}^d = \left[x^d_1, \ldots, x^d_{N_d}\right], \tag{4.8}$$

where N_d is the length of the discrete sequence. Each categorical value is mapped to a learned embedding vector through an embedding lookup table:

$$\mathbf{E}^d_{\text{token}} = \left[\text{Emb}_d(x^d_1); \ldots; \text{Emb}_d(x^d_{N_d})\right] \in \mathbb{R}^{N_d \times e_d}. \tag{4.9}$$

Thus, despite having different original representations, both the continuous and discrete inputs are transformed into sequences of vectors with the same embedding dimension e_d.

4.2.1. Positional and Modality embeddings

Following [18], positional embeddings are added to preserve information about the position of each token within its corresponding sequence. Separate learned positional embeddings are used for the continuous and discrete components:

$$\mathbf{P}^c \in \mathbb{R}^{N_c \times e_d}, \qquad \mathbf{P}^d \in \mathbb{R}^{N_d \times e_d}. \tag{4.10}$$

These parameters are initialised using samples from a normal distribution.

In addition, one of the implementations that was included is modality embeddings. Modality embeddings allow the model to distinguish between tokens originating from the continuous and discrete modalities. Two learned vectors are introduced:

$$\boldsymbol{\tau}^c, \boldsymbol{\tau}^d \in \mathbb{R}^{e_d}, \tag{4.11}$$

where $\boldsymbol{\tau}^c$ identifies continuous tokens and $\boldsymbol{\tau}^d$ identifies discrete tokens. Each vector is broadcast across all tokens of its corresponding modality.

The complete embeddings for the continuous and discrete inputs are consequently given by

$$\mathbf{Z}^c = \mathbf{E}^c_{\text{patch}} + \mathbf{P}^c + \mathbf{1}_{N_c}(\boldsymbol{\tau}^c)^\top, \tag{4.12}$$

$$\mathbf{Z}^d = \mathbf{E}^d_{\text{token}} + \mathbf{P}^d + \mathbf{1}_{N_d}(\boldsymbol{\tau}^d)^\top, \tag{4.13}$$

where $\mathbf{1}_N$ is a column vector of ones used to broadcast a modality embedding across N number of tokens.

Finally, the continuous and discrete embeddings are concatenated along the token dimension:

$$\mathbf{Z} = \text{Concat}\left(\mathbf{Z}^c, \mathbf{Z}^d\right) \in \mathbb{R}^{(N_c+N_d)\times e_d}. \tag{4.14}$$

This combined token sequence is used as the input to the transformer layers.

4.2.2. Generative-time embeddings

As will be described in Section 4.5, the continuous and discrete trajectory components may be evaluated at different generative times, denoted by t_c and t_d, respectively. Both generative-time values are encoded using sinusoidal embeddings and concatenated.

$$\mathbf{e}_t = \text{Concat}\left[\text{PE}(t_c), \text{PE}(t_d)\right] \in \mathbb{R}^{2e_t}, \tag{4.15}$$

where e_t is the dimension of each individual sinusoidal time encoding.

A modality embedding is also incorporated into the time representation, allowing the model to distinguish the generative time associated with each modality. The resulting time embedding is used to condition the transformer during the generative process.

4.2.3. Observation embeddings

The model is additionally conditioned on the (initial) available observation data. Since the observation vector contains heterogeneous quantities, such as robot states, object poses, and scene parameters, its components are divided according to their semantic type. Let

$$\mathbf{o} = \left[\mathbf{o}^{(1)}, \mathbf{o}^{(2)}, \ldots, \mathbf{o}^{(N_o)}\right] \tag{4.16}$$

denote an observation composed of N_o parameter groups. Each group is independently projected into the common embedding space:

$$\mathbf{e}_{\text{obs}}^{(k)} = f_{\text{obs}}^{(k)}\left(\mathbf{o}^{(k)}\right) + \boldsymbol{\tau}_{\text{obs}}^{(k)}, \qquad k = 1, \ldots, N_o \tag{4.17}$$

where $f_{\text{obs}}^{(k)}$ is the projection associated with observation type k, and $\boldsymbol{\tau}_{\text{obs}}^{(k)}$ is its learned observation-modality embedding.

The complete observation representation is obtained by combining the embedded parameter groups:

$$\mathbf{E}_{\text{obs}} = \text{Concat}\left(\mathbf{e}_{\text{obs}}^{(1)}, \ldots, \mathbf{e}_{\text{obs}}^{(N_o)}\right). \tag{4.18}$$

By explicitly encoding the semantic type of each observation component, the model can distinguish between heterogeneous sources of information and learn the relationships between the scene state, the discrete task sequence, and the continuous trajectory.

Finally, the observation and generative-time embeddings are combined to form the conditioning information provided to the model, also called *memory*, as indicated in Figure 4.1. This allows the generative process to be conditioned on both the initial state of the environment and the generative times of the continuous and discrete components, as further explained in Section 4.5.

4.3. Training Procedure

Planning with the proposed model involves two main processes: training and inference. During training, the model learns to reconstruct continuous trajectories and discrete task sequences from corrupted samples while being conditioned on observations of the initial state of the environment.

4.3.1. Padding

The demonstrations collected using the real robot contain different numbers of trajectory steps. This variability arises because the demonstrations may contain different task sequences or require different numbers of waypoints to complete the task.

Also, unlike in the baseline method [18], the demonstrations were gathered by a human and not by a scripted expert, making it impractical to create motion trajectories of the same horizon.

To process multiple demonstrations within the same batch, the continuous trajectories must share a common temporal dimension. Repeating the final waypoint, rather than introducing an arbitrary constant value, maintains a physically consistent stationary configuration throughout the padded section of the trajectory.

A binary validity mask is also created to distinguish the original trajectory from the padded waypoints.

The discrete task sequences are padded in a similar manner. Each valid sequence terminates with an *End-of-Sequence* (*EOS*) token, which indicates that the task plan has been completed. Sequences shorter than the maximum sequence length among all demonstrations are extended using a designated *fill* token:

Thus, the padded discrete sequence has the structure

$$\widetilde{\mathbf{x}}_b^d = \left[x_{b,1}^d, \ldots, x_{\text{EOS}}, x_{\text{fill}}, \ldots, x_{\text{fill}}\right]. \tag{4.19}$$

The *EOS* token is part of the valid task sequence and must therefore be learned by the model. In contrast, the *fill* tokens only occupy the unused positions following the *EOS* token. A discrete validity mask identifies this distinction as well.

Padding is introduced only to construct tensors with consistent dimensions. Consequently, repeated continuous waypoints and discrete *fill* tokens are excluded from the corresponding loss calculations through their validity masks. The *EOS* token, however, remains a valid element and contributes to the discrete training objective.

4.3.2. Loss calculation

After padding, separate generative times are sampled for the continuous and discrete components. The continuous trajectory is corrupted using noise sampled from an isotropic Gaussian distribution (Equation 4.2), with the sampled continuous generative time determining its corruption level. For the discrete sequence, independent Bernoulli retention indicators are sampled over the valid token positions, with retention probability t_d [34]. Tokens whose indicators equal zero are replaced by the `MASK` token, corresponding to a masking probability of $1 - t_d$ (Equation 4.1). The padded positions following the *EOS* token are also replaced by `MASK` tokens before the sequence is provided to the model, preventing an explicit padding boundary from revealing the original sequence length. These padded positions remain excluded from the loss calculation through the discrete validity mask.

Given the sampled generative times, the partially corrupted demonstration, and the observation of the initial state of the environment, the model jointly predicts the continuous velocity and the categorical logits for the discrete sequence. Since these outputs have different mathematical representations, a separate loss function is calculated for each component.

Discrete loss The discrete branch predicts a categorical distribution over the possible tokens at each sequence position. Its output can be represented by a set of logits

$$\boldsymbol{\ell}_{b,j} \in \mathbb{R}^C, \tag{4.20}$$

where C is the number of discrete token classes.

Cross-entropy loss is calculated only at positions that are both valid and masked during the corruption process. Let $r_{b,j} = 0$ indicate that token j was masked and must be reconstructed and $m^d_{b,j}$ the discrete validity mask. The discrete loss for demonstration b is then

$$\mathcal{L}^d_b = \frac{\sum_{j=1}^{N_d} m^d_{b,j}(1 - r_{b,j})\,\mathrm{CE}\left(\boldsymbol{\ell}_{b,j}, z^d_{b,j}\right)}{\sum_{j=1}^{N_d} m^d_{b,j}(1 - r_{b,j}) + \varepsilon}, \tag{4.21}$$

where $z^d_{b,j}$ is the ground-truth token and ε is a small constant that prevents division by zero. Normalising by the number of valid masked tokens prevents demonstrations with longer discrete sequences from having a disproportionately large contribution to the loss.

Continuous loss For the continuous component, the model predicts the velocity associated with the flow-matching process. Let

$$\widehat{\mathbf{v}}_{b,h} = \mathbf{v}_\theta\left(\mathbf{x}^c_{t,b}, \mathbf{x}^d_{t,b}, t_c, t_d, \mathbf{o}_{b,0}\right)_h \tag{4.22}$$

denote the predicted continuous velocity at trajectory step h, where $\mathbf{o}_{b,0}$ represents the observation of the initial state of the environment. Given the target velocity for each demonstration $\mathbf{u}_b$, which is obtained following Equation 4.24 [24], the continuous loss for demonstration b is calculated as

$$\mathcal{L}^c_b = \frac{\sum_{h=1}^{H_{\max}} \sum_{f=1}^{F_c} m^c_{b,h}\left(\widehat{v}_{b,h,f} - u_{b,h,f}\right)^2}{F_c \sum_{h=1}^{H_{\max}} m^c_{b,h} + \varepsilon}, \tag{4.23}$$

where F_c is the number of continuous features per trajectory step and $m^c_{b,h}$ is the continuous validity mask. By excluding padded trajectory steps and normalising by the number of valid continuous values, longer demonstrations do not receive greater weight solely because they contain more waypoints.

$$\mathbf{u}_b = z^c_b - \epsilon_b = \frac{z^c_b - x^c_{t_c,b}}{1 - t_c} \tag{4.24}$$

where $\mathbf{z}^c_{b,h}$ is the clean continuous state at trajectory step h and for demonstration b, $\boldsymbol{\epsilon}_{b,h}$ is the corresponding sampled Gaussian noise vector, and t_c is the continuous generative time.

The losses are averaged over the demonstrations of the selected batch and finally, the continuous and discrete objectives are combined as

$$\mathcal{L} = \mathcal{L}^c + \lambda_d \mathcal{L}^d, \tag{4.25}$$

where λ_d is a scaling factor that controls the contribution of the discrete loss into the combined loss [18] and, $\mathcal{L}^c$ and $\mathcal{L}^d$ are the averaged losses of the continuous and discrete parts, respectively. The resulting combined loss is used to optimise the model parameters.

The complete training procedure is summarised in Algorithm 1.

4.4. Prediction Procedure

After training, the model can generate a hybrid plan from an observation of the initial state of the environment. The generation process starts from a fully corrupted continuous trajectory and a fully masked discrete sequence. Both components are then progressively refined over a fixed number of inference steps, similarly to [18].

Let K denote the number of inference steps. The continuous and discrete time intervals are discretised as

$$0 = t^c_0 < t^c_1 < \cdots < t^c_K = 1, \qquad 0 = t^d_0 < t^d_1 < \cdots < t^d_K = 1. \tag{4.26}$$

where $t_0 = 0$ corresponds to the fully corrupted state and $t_K = 1$ corresponds to the generated plan. For the continuous component, the integration step at iteration k is

$$\Delta t^c_k = \frac{1}{K}. \tag{4.27}$$

The prediction procedure consists of the following steps:

1. The continuous trajectory is initialised by sampling from a Gaussian distribution:

$$\mathbf{x}^c_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{4.28}$$

   The valid positions of the discrete sequence are initialised using the mask token:

$$x^d_{0,j} = x_{\text{mask}}, \qquad j = 1, \ldots, N_d. \tag{4.29}$$

2. At every inference step k, the current continuous and discrete states are passed through the model together with their generative times and the initial-state observation:

$$(\widehat{\mathbf{v}}_k, \boldsymbol{\ell}_k) = f_\theta\left(\mathbf{x}^c_k, \mathbf{x}^d_k, t^c_k, t^d_k, \mathbf{o}_0\right), \tag{4.30}$$

   where $\widehat{\mathbf{v}}_k$ is the predicted continuous velocity and $\boldsymbol{\ell}_k$ contains the predicted logits for the discrete tokens.

3. The continuous trajectory is updated using an Euler integration step [12]:

$$\mathbf{x}_{k+1}^{c} = \mathbf{x}_{k}^{c} + \Delta t_{k}^{c} \hat{\mathbf{v}}_{k}. \tag{4.31}$$

4. For the discrete component, the logits are converted into categorical probabilities:

$$\widehat{\mathbf{p}}_{k,j} = \mathrm{softmax}\left(\boldsymbol{\ell}_{k,j}\right). \tag{4.32}$$

   At each inference step, the discrete sampling schedule determines which of the positions that remain masked are selected for unmasking. For each selected position, the token with the highest predicted probability among the tokens is revealed:

$$x_{k+1,j}^{d} = \arg\max_{c} \widehat{p}_{k,j}(c), \tag{4.33}$$

   where c ranges over all token classes in the discrete vocabulary. Positions that are not selected for unmasking retain the *MASK* token, while previously revealed tokens remain unchanged unless they are later selected by the remasking procedure.

   Another of our additional implementations was remasking. Remasking is performed at three evenly spaced iterations during inference. At each remasking step, the confidence of every previously revealed token is obtained from the categorical probability assigned to its current value. The lowest-confidence 10% of these tokens are replaced by the *MASK* token, allowing uncertain early predictions to be reconsidered using the additional context available at later iterations. The discrete generative time is adjusted accordingly before being passed to the model at the next iteration.

5. Steps 2–4 are repeated for K inference iterations. The resulting continuous trajectory $\mathbf{x}_{K}^{c}$ and discrete sequence $\mathbf{x}_{K}^{d}$ constitute the generated hybrid plan.

Throughout this procedure, the initial-state observation $\mathbf{o}_0$ remains fixed and conditions every inference iteration. Therefore, although generation begins from random continuous values and masked discrete tokens, the resulting plan is guided towards a solution that is consistent with the initial configuration of the robot and its environment.

After the final inference step, the generated discrete sequence is truncated at the first *End-Of-Sequence* token, (*EOS*). The *EOS* token marks the end of the predicted symbolic plan, and all tokens appearing after it are discarded.

The complete prediction procedure is summarised in Algorithm 2.

4.5. Transformer pipeline

The hybrid flow-matching model is parameterised using a sequence of Transformer blocks. Each block contains three principal components: self-attention, cross-attention, and a position-wise feed-forward network. Together, these components allow the model to capture dependencies within the hybrid plan, condition its predictions on the initial environment state and generative times, and transform the resulting token representations non-linearly. The position of this structure within the architecture is visible in Figure 4.1.

1. **Self-attention**

   The continuous trajectory tokens and discrete symbolic tokens are combined into a single sequence and processed using self-attention. This mechanism allows every token to attend to all the other tokens in the hybrid sequence, independently of their relative distance [38]. Given query, key, and value matrices $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$, respectively, scaled dot-product attention is calculated as

$$\mathrm{Attention}\left(\mathbf{Q}, \mathbf{K}, \mathbf{V}\right) = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\mathsf{T}}}{\sqrt{d_k}}\right)\mathbf{V}, \tag{4.34}$$

where d_k is the dimensionality of the key vectors. In the self-attention component, the queries, keys, and values are obtained from the same hybrid token sequence. The multi-head formulation performs this operation using multiple learned projections, enabling the model to capture different types of relationships simultaneously.

This component is particularly important in the proposed architecture because it enables information exchange between the continuous and discrete modalities. For example, discrete symbolic tokens can attend to the continuous trajectory segments associated with their execution, while continuous tokens can incorporate information about the task-level action sequence.

The position and modality embeddings introduced in Section 4.2.1 provide additional information before self-attention is applied. Position embeddings preserve the ordering of tokens, whereas modality embeddings allow the model to distinguish between continuous and discrete representations.

2. **Cross-attention**

   The output of the self-attention component is subsequently processed using cross-attention. Cross-attention allows one sequence to retrieve and integrate relevant information from a separate conditioning sequence [36].

   In this architecture, the hybrid plan tokens provide the queries, while the keys and values are obtained from a conditioning representation that combines the observation of the initial state with the continuous and discrete generative-time embeddings. This combination is called *memory*, as shown in Figure 4.1. Consequently, the cross-attention operation can be expressed as

$$\mathbf{Z}_{\text{cross}} = \text{Attention}\left(\mathbf{Z}_{\text{plan}}\mathbf{W}^{Q}, \mathbf{C}\mathbf{W}^{K}, \mathbf{C}\mathbf{W}^{V}\right), \tag{4.35}$$

   where $\mathbf{Z}_{\text{plan}}$ denotes the hybrid plan representation, $\mathbf{C}$ denotes the conditioning representation, and $\mathbf{W}^{Q}$, $\mathbf{W}^{K}$, and $\mathbf{W}^{V}$ are learned projection matrices.

   This conditioning mechanism enables the model to relate the discrete task sequence and continuous trajectory to the initial robot and environment configuration. The generative-time embeddings additionally inform the model of the current corruption level of each modality, allowing it to produce an appropriate generative prediction at every stage of the flow-matching process.

3. **Position-wise feed-forward network**

   The attention components are followed by a position-wise feed-forward network, also referred to as a multi-layer perceptron (MLP). It applies the same non-linear transformation independently to the representation of every token [1].

   While the attention components exchange information between tokens and conditioning inputs, the feed-forward network transforms each resulting token representation independently. This enables the model to learn more complex non-linear features after the relevant relationships have been identified through attention.

After the final Transformer block, separate output heads are used for the two modalities. The continuous output head predicts the velocity field used to update the continuous trajectory, whereas the discrete output head produces categorical logits for the task tokens (Figure 4.1). Thus, the shared Transformer representation enables joint reasoning over both components of the plan while preserving the modality-specific outputs required by their respective generative processes.

4.6. Data augmentation

To improve the generalisation capability of the model, the original demonstrations are augmented by extracting valid subtrajectories. This procedure increases the effective number of training samples and exposes the model to initial environment states that occur at intermediate stages of the original demonstrations.

Each demonstration is divided into suffixes according to its discrete action boundaries. Let a demonstration contain the ordered action sequence

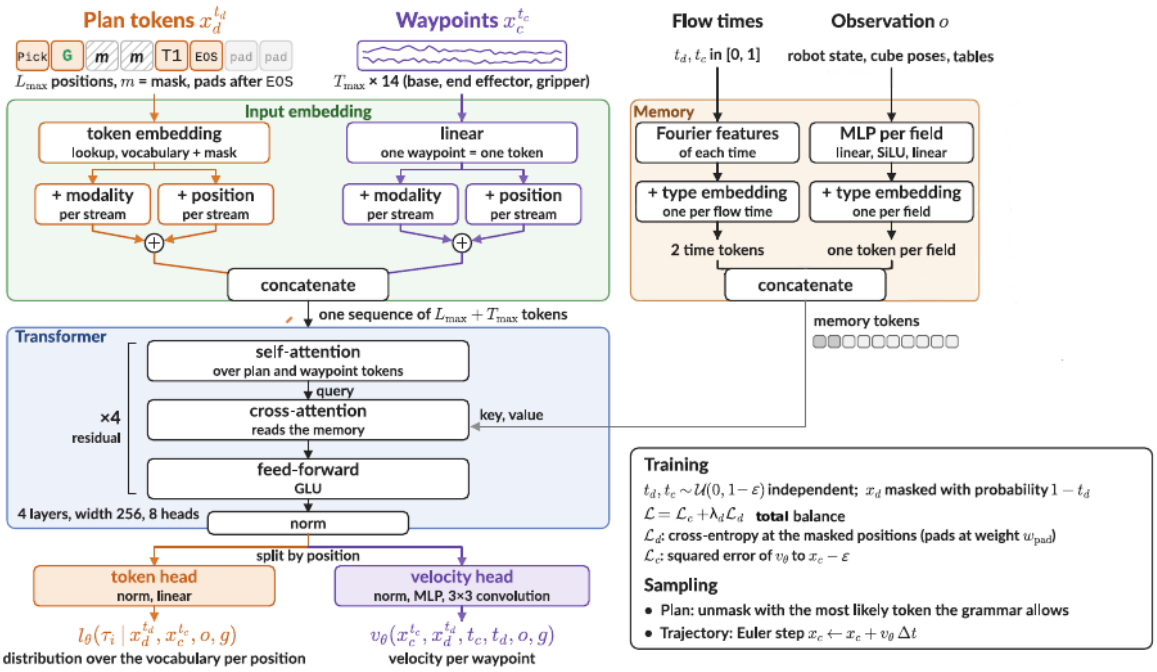


Figure 4.1: Architecture diagram. The symbolic plan tokens and the waypoints get a token embedding or a linear layer, a learned modality and positional embedding, and form one sequence for the transformer. Its cross-attention reads the memory: one token per flow time and one per observation component (an MLP), each with a modality embedding. The token head gives l_θ at every symbolic position and the velocity head v_θ at every waypoint.

$$\mathbf{a}^{(b)} = \left[a_1^{(b)}, a_2^{(b)}, \ldots, a_{M_b}^{(b)}\right], \tag{4.36}$$

where M_b is the number of actions in demonstration b. Each action $a_k^{(b)}$ is associated with the continuous waypoints executed while that action is active. This association is obtained from an additional annotation that assigns an action index to every waypoint in the demonstration.

Starting from the complete demonstration, new training samples are generated by progressively removing actions from the beginning of the sequence, together with their associated continuous waypoints. An example of this is shown in Figure 4.2.

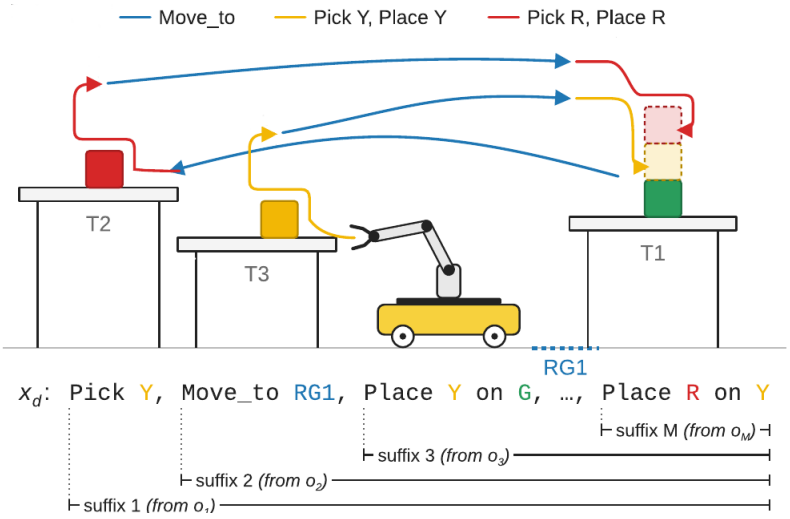


Figure 4.2: Example of plan suffixes generated out of one single demonstration [28].

This way, the observation recorded at the beginning of each action becomes the initial-state observation of the resulting subtrajectory.

For example, a demonstration containing the action sequence $[a_1, a_2, a_3]$ produces the following training samples:

$$[a_1, a_2, a_3], \qquad [a_2, a_3], \qquad [a_3], \tag{4.37}$$

each paired with its corresponding continuous trajectory and initial-state observation.

Each action is represented by an action token followed by its corresponding arguments. A *Move_to* action is followed by an approach-region token, a *Pick* action by a cube token, and a *Place* action by a cube token and a support token, where the support may be either another cube or a table. An example of a possible discrete sequence is given in Equation 5.1.

This augmentation allows the model to learn how to complete the task not only from the original initial configuration, but also from the intermediate environment states reached after executing each action. Given M_b actions in demonstration b, the procedure produces M_b valid training samples, including the original complete demonstration. The effect of training with and without this subtrajectory augmentation is evaluated in Section 5.3.

5 Results

5.1. Environment

The environment used to evaluate the accuracy and performance of the method is shown in Figure 5.1 as a simplified version of the actual setup. The most important elements are listed below.

- 3 Cubes: The setup includes three cubes that have a side of 6.5 cm and are the only elements that the robot can manipulate.

  These can be positioned on any of the tables and on top of each other. Towers can be formed by two or three cubes. If the tower consists of three blocks, it can only be positioned on *Table 3* or *Table 1*; if it consists of two blocks, it can be positioned on any of the tables.

- 3 Tables: Two of the three tables are positioned to the left of the setup and are next to each other, allowing the robot to approach them from only three sides. This constraint influences how the robot handles manipulation tasks. Moreover, the tables have different heights, adding complexity to the task.

  Table 1 is located to the right and determines the final position of the goal tower.

- Robot: To execute the predicted planning tasks, the *Dinova* robot [4], developed by the AMR lab at Delft University of Technology, is used. This robot combines a holonomic mobile base (Clearpath Dingo-O) with a six-joint arm (Kinova Gen3 Lite), allowing it to move around the environment and manipulate the cubes from one table to another to achieve the final goal.

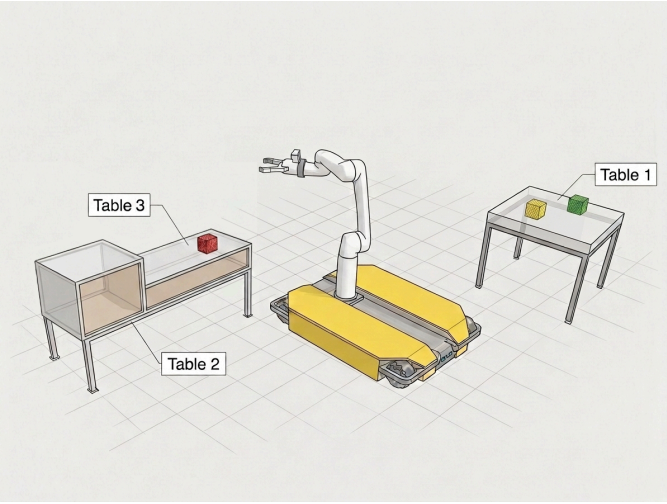


Figure 5.1: Schematic of the experimental environment. It shows the 3 tables available, 2 on the left next to each other and 1 on the right, which indicates the goal table; 3 cubes with different colours: red, yellow, and green, and the mobile manipulator.

5.2. Dataset Generation

The dataset was collected using the real-world experimental setup shown in Figure 5.2. The positions of the three tables remained fixed throughout the data-collection process, while the robot and cubes were moved between demonstrations. The initial position and orientation of the robot were also varied around the environment.

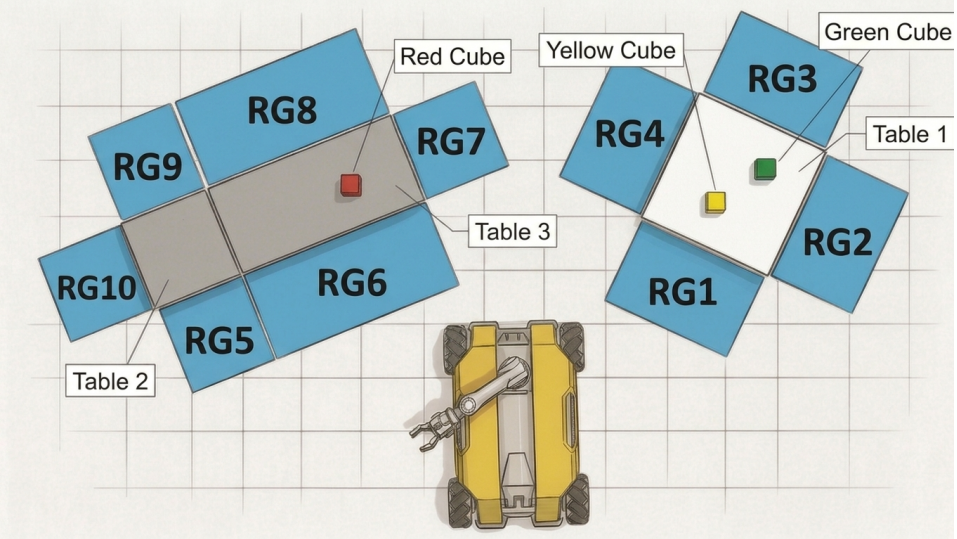


Figure 5.2: Top view of the experimental setup used for dataset collection with approach-region labels, table names and cube names indicated.

In total, 121 demonstrations were recorded, of which 109 were used for training and 12 for validation (Figure 5.4). In contrast to the baseline method in [18], which relies on demonstrations generated by a scripted expert, our dataset was collected through human teleoperation. This introduces additional variability into the demonstrations, since the executed trajectories are not perfectly repeatable and may contain small inconsistencies or execution errors. At the same time, these demonstrations more closely reflect the variability that can be expected when collecting data directly from a real robotic system.

To introduce further variability into the dataset, the demonstrations were divided into three categories according to the initial cube configuration (Figure 5.3):

- **Demonstration type 1: Green cube on Table 1.**

  In this category, the green cube, which forms the base of the target tower as described in Section 3.1, is initially located on *Table 1*, its required final table. At least one of the other two cubes is located on either *Table 2* or *Table 3*. The red and yellow cubes may be placed separately or stacked on one of these tables. Alternatively, one of them may initially be placed on top of the green cube on *Table 1*, while the other remains on *Table 2* or *Table 3*.

- **Demonstration type 2: Table 1 unoccupied.**

  For this category, *Table 1* is initially unoccupied. All three cubes are distributed between *Table 2* and *Table 3*. The cubes may be placed individually or stacked on top of one another, resulting in configurations from which the complete target arrangement must be constructed on *Table 1*.

- **Demonstration type 3: Red or yellow cube on Table 1.**

  In the final category, either the red or the yellow cube is initially located on *Table 1*. The remaining cubes may be placed separately or stacked on *Table 2* or *Table 3*. They may also be stacked on top of the cube already positioned on *Table 1*. These configurations require the robot to reorganise cubes that have already been placed on the target table before constructing the correct final arrangement.

The demonstrations were performed through teleoperation. A *PS4* controller was used to control the holonomic mobile base, while a virtual-reality (*VR*) system was used to control the six degrees of

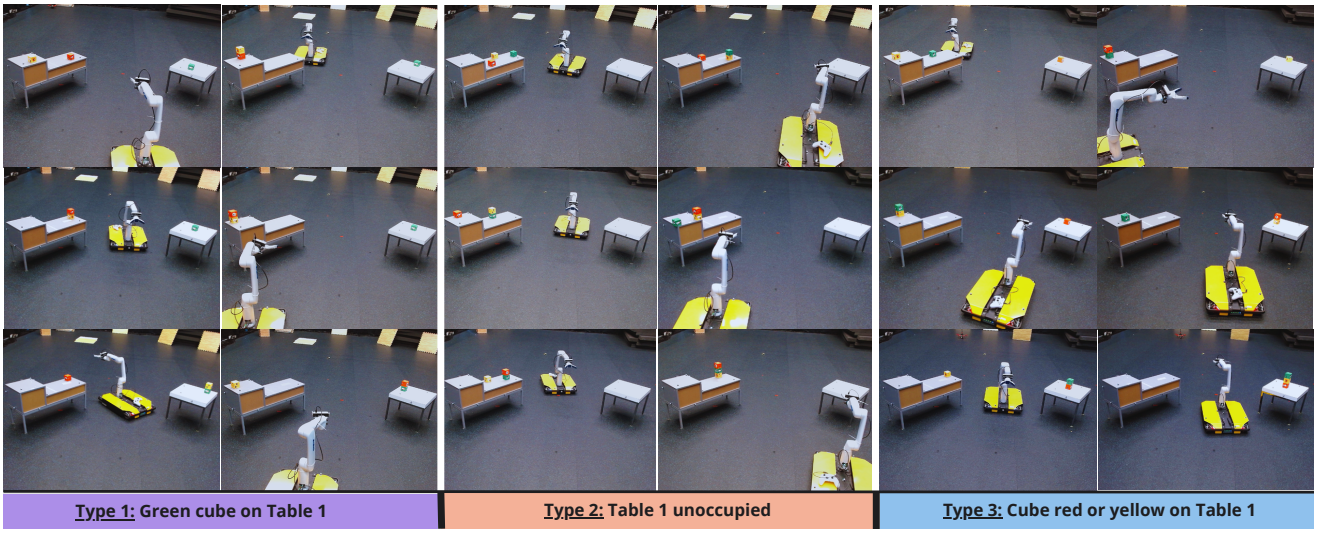


Figure 5.3: Examples of initial demonstration configurations for the three dataset categories.

freedom of the robot arm, as further explained in Section 5.2.1

Each demonstration consists of a sequence of manually selected and recorded waypoints. At every waypoint, the following information was stored:

- **Robot base pose.** The position of the mobile base along the x- and y-axes and its orientation θ around the vertical axis, expressed with respect to the center of the experimental environment.
- **Robot arm joint angles.** The angular position of each of the six joints of the robot arm.
- **Gripper state.** A binary value indicating whether the gripper is closed or open. A value of 1 represents a closed gripper, while a value of 0 represents an open gripper.
- **Cube poses.** The three-dimensional position and orientation of each cube.
- **Table positions and dimensions.** The position and dimensions of each table. Although the tables remain fixed throughout the demonstrations, this information is included because, unlike the cubes, they have different sizes.

Although the robot arm joint angles are recorded, they are not provided directly to the final model. Instead, forward kinematics is used to calculate the position and orientation of the end-effector with respect to the Motion Capture System coordinate system. The resulting end-effector pose is used as the representation of the robot arm in place of the individual joint angles.

The continuous trajectory contains only the variables describing the robot state, as these are the values that the model must predict. These variables comprise the robot base pose, the end-effector pose, and the gripper state. By contrast, the observation also describes the surrounding environment and therefore includes the robot state, the poses of the three cubes in 3D, and the positions and dimensions of the tables.

Each demonstration is associated with two additional forms of discrete information: a task sequence and a set of suffix annotations. The task sequence specifies the ordered actions that the robot must execute to solve the task from the corresponding initial configuration. It is constructed using a vocabulary comprising four categories of tokens together with *EOS*: action tokens, such as *Move_to*, *Pick*, and *Place*; cube identifiers, where *R*, *Y*, and *G* represent the red, yellow, and green cubes, respectively; table identifiers *T1*, *T2*, and *T3*; and approach-region identifiers ranging from *RG1* to *RG10*. The locations of these approach regions are shown in Figure 5.2.

For example, a task sequence for picking the green cube from Table 3 and placing it on *Table 1* could be represented as

$$[\texttt{Move\_to}, \texttt{RG6}, \texttt{Pick}, \texttt{G}, \texttt{Move\_to}, \texttt{RG1}, \texttt{Place}, \texttt{G}, \texttt{T1}] \tag{5.1}$$

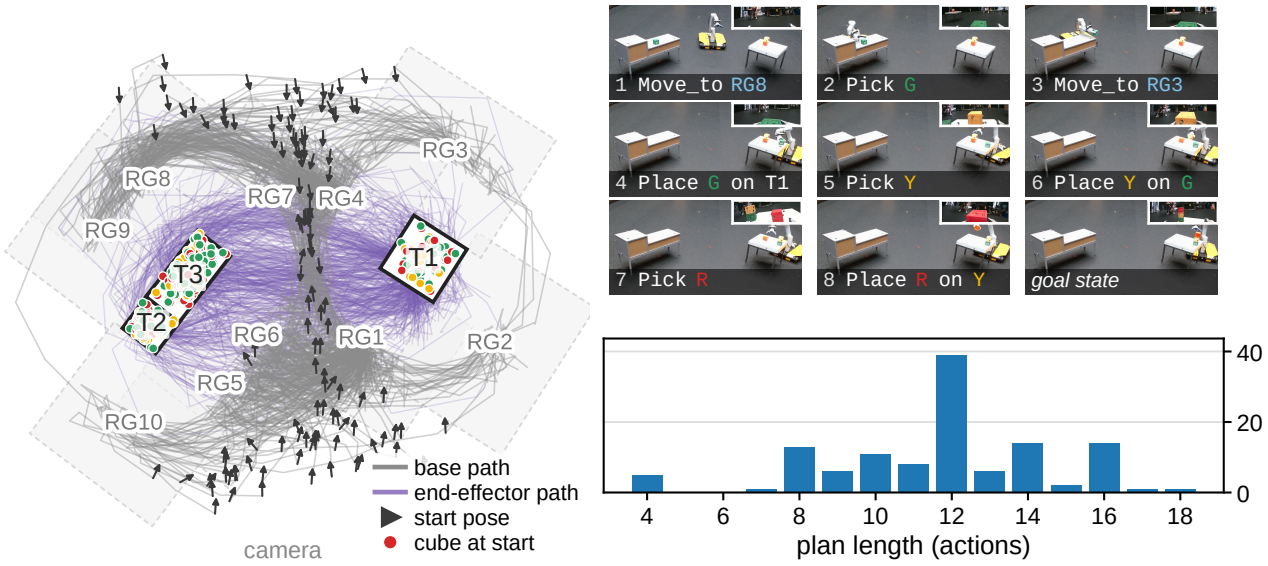


Figure 5.4: The 109 (+12 validation) demonstrations. Left: top view with the tables T1–T3 and the approach regions RG1–RG10: base paths (grey), end-effector paths (violet), start poses (arrows), and initial cube positions (dots). Right: one demonstration represented by observations $o_1, \ldots, o_8$ at the action boundaries, together with the corresponding actions, wrist-camera views, and the distribution of plan lengths at the bottom right [28].

This sequence instructs the robot to move to approach region `RG6` (Table 3), pick up the green cube, move to approach region `RG1` (Table 1), and place the cube on *Table 1*.

The suffix annotations assign an action index to every continuous waypoint, thereby identifying the trajectory segment associated with each high-level action. A high-level action may be represented by multiple tokens in the discrete sequence, such as *Move_to* followed by an approach-region identifier. The suffix annotations are used during preprocessing to align the continuous trajectory with the discrete task sequence and to generate the subtrajectories described in Section 4.6 in order to increase the number of available training samples.

5.2.1. Arm Joint-State Collection

A baseline VR teleoperation implementation [27] was used and adapted to obtain the pose of the right Meta Quest 3 controller and map it to the desired 3D pose of the robot end-effector. The controller motion is interpreted incrementally with respect to the current end-effector pose. When the gripper button on the right controller (button 4 in Figure 5.5) is pressed, the current controller pose is taken as the reference, allowing subsequent controller motion to define relative changes in the desired end-effector pose.

Unity Hub was used to adapt the controller measurements to the coordinate system of the MuJoCo simulation environment. The resulting end-effector pose was then used to solve the inverse kinematics of the robot and determine the corresponding arm joint angles that move the real system. The joint angles are then collected from the robot's ROS topics and included as part of the observation and continuous movement of the robot in the dataset. To ensure consistency between the real and simulated systems, the initial end-effector pose was set to correspond to the robot arm's *home* configuration in both the MuJoCo environment and the Unity Hub implementation.

Two additional functionalities were implemented using the trigger buttons of the Meta Quest 3 controllers:

- Right-controller trigger (button 1 in Figure 5.5): Sends a command to the robot gripper to toggle its state, enabling convenient opening and closing directly from the VR controller.
- Left-controller trigger (button 1 in Figure 5.5): Resets the robot arm to its predefined *home* configuration. This functionality facilitates the continuous collection of multiple demonstrations by allowing the arm to be quickly returned to a consistent initial configuration between trials.

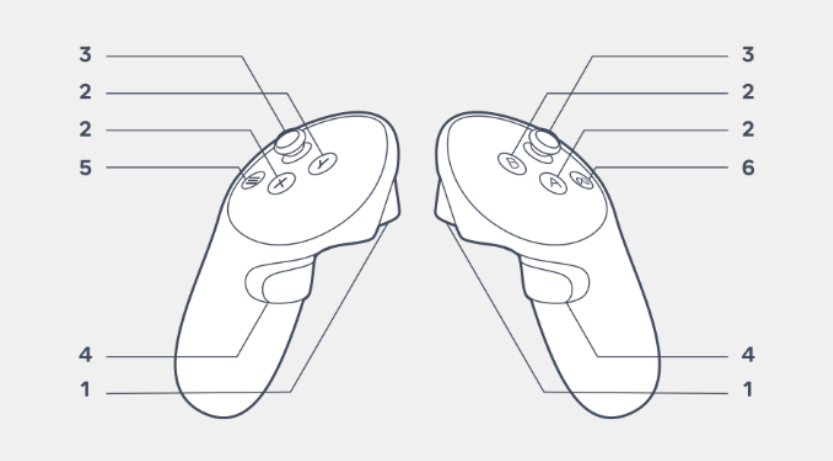


Figure 5.5: Meta Quest 3 controllers schematic. Available: `https://developers.meta.com/horizon/design/controllers/`

Although the joint angles are obtained during teleoperation through inverse kinematics, the model ultimately operates on an end-effector representation rather than directly on the arm joint configuration, as discussed previously. A joint-space representation was initially considered, as it would simplify trajectory execution in both simulation and on the real robot by avoiding the repeated conversion between Cartesian poses and joint configurations through inverse and forward kinematics.

However, the end-effector representation was selected because it provides a more direct description of the spatial relationship between the robot and the manipulated objects. In particular, representing the arm through the end-effector pose makes it easier for the model to capture when the robot is sufficiently close to a cube to perform a *Pick* action or to the desired placement location to perform a *Place* action. This representation was therefore considered more informative for learning the relationship between the continuous robot trajectory and the corresponding symbolic actions, despite requiring additional kinematic transformations for execution and visualisation.

Finally, an example of the demonstration-collection process is available at `https://youtu.be/JDVGCyttcfI`. The video shows a top-down view of the recording of a complete demonstration and is played at four times the original speed. In practice, collecting a single demonstration required approximately 15–20 minutes, depending on the complexity of the initial configuration.

5.3. Comparisons

Training and inference The transformer has four layers of width 256 with eight heads, 7.5 million parameters. It is trained for 20 000 steps with batches of 45 samples, with the weights averaged over training and using the valid mask for the loss calculation, in about 20 minutes on one NVIDIA RTX 3090. Inference uses $K = 100$ steps. Every configuration is trained with three seeds; we report the mean and the standard deviation over them.

Metrics A sampled plan is *valid* under the symbolic model of Section 3, evaluated from the scene at the start of the task. Execution is evaluated in a kinematic simulation of the scene: the sampled waypoints move the robot, a cube within 9 cm of the end-effector attaches when the gripper closes, and a released cube settles on the support beneath it. Tolerances are set to the 99th percentile of the demonstrations. The *task completion* rate is the fraction of samples after which the goal tower stands, whatever plan was executed. In the *open-loop*, the whole sampled trajectory is executed; in *closed-loop*, only the first action segment is executed before replanning. Results are reported overall and by the number of actions remaining to the goal, d. Finally, the end-effector root mean square error with respect to the demonstrated trajectory in meters, is also included to study how much the robot end point deviates from the original plan.

Policies *Hybrid FM* is the hybrid flow matching planner of Section 4, trained on the suffix samples with the full training and generation scheme. *Motion-only FM* (flow matching) is a continuous flow matching on the waypoints alone, the same network without the plan tokens, with a training sample starting at every waypoint of the demonstrations, not at the action boundaries; *Hybrid diffusion* is the formulation of Høeg et al. [18] on our network and data (diffusion on the waypoints, masked diffusion on the tokens, their loss weight, sampler, and optimiser), trained as published on whole demonstrations or on our suffix samples.

5.3.1. Discrete loss weight ablation

Following Høeg et al. [18], the proposed Hybrid FM model uses a weighting parameter, λ_d, to balance the contribution of the discrete loss relative to the continuous loss in the combined training objective. Such weighting is necessary because the two objectives correspond to different prediction problems and operate on different numerical scales.

To study the influence of this parameter, several values of λ_d were evaluated and compared in terms of symbolic-plan validity and task-completion performance. The results are reported in Table 5.1.

Increasing λ_d generally improves the validity of the symbolic plan, particularly when comparing the smallest values. Increasing the weight from 0.01 to 0.05 raises the overall symbolic-plan validity from 44.4% to 55.1%, while the long-horizon validity for $d \geq 13$ increases from approximately 0.1% to 4.7%. A further increase to $\lambda_d = 0.1$ results in an overall validity of 57.2% and a substantially higher long-horizon validity of approximately 17.4%.

The long-horizon metric, however, is not monotonic with respect to λ_d. For example, it decreases to approximately 6.7% for $\lambda_d = 0.3$, increases again for $\lambda_d = 1.0$, and reaches approximately 17.6% for $\lambda_d = 3.0$. These fluctuations suggest that long-horizon symbolic performance is more sensitive to training variability, particularly given the relatively small number of distinct validation configurations available at larger planning distances.

A clearer trend can be observed in task-completion performance. Low values of λ_d preserve stronger motion-execution performance, while placing greater emphasis on the discrete objective generally reduces both open-loop and closed-loop task completion. For instance, increasing λ_d from 0.05 to 0.3 reduces open-loop success from 30.4% to 27.4% and closed-loop success from 40.9% to 37.0%. Although $\lambda_d = 1.0$ produces a small recovery in open-loop performance, a larger value of $\lambda_d = 3.0$ causes a substantial decline, reaching only 21.3% in open-loop and 23.2% in closed-loop.

Overall, the ablation reveals a trade-off between symbolic-plan quality and continuous execution performance. Increasing the influence of the discrete objective generally benefits symbolic prediction, but excessive weighting can negatively affect the continuous component of the model and, consequently, task completion. Since accurate motion generation is particularly important for successful manipulation, the selected value should improve symbolic planning without substantially degrading the continuous trajectory predictions.

Based on this trade-off, $\lambda_d = 0.05$ was selected for the final model. This value achieves an overall symbolic-plan validity of 55.1% and a long-horizon validity of approximately 4.7%, while maintaining task-completion rates of 30.4% in open-loop and 40.9% in closed-loop. Compared with $\lambda_d = 0.01$, it provides a substantial improvement in symbolic-plan validity while preserving almost identical task-completion performance.

Høeg et al. [18] use a discrete-loss weight of $\lambda_d = \frac{1}{30}$. In our setting, a slightly larger value was selected to provide greater influence to the symbolic objective while preserving the performance of the continuous motion component.

5.3.2. Model Architecture Ablations

Model architecture ablations The lower part of Table 5.2 evaluates the cumulative effect of the main implementation changes introduced during development. The initial configuration uses a joint-space representation without plan-suffix augmentation and without remasking. This setup achieves an overall symbolic-plan validity of only 5%, which decreases to approximately 1% for long-horizon plans with

Table 5.1: Performance in percent for different discrete loss weights λ_d, averaged over 3 training seeds. Values show mean $\pm$ standard deviation across seeds.

	Symbolic plan valid (%)		Task completed (%)	
λ_d	all d	$d \geq 13$*	open-loop	closed-loop
0.01	44.4 ± 1.0	≈ 0.1	30.6 ± 1.6	41.1 ± 2.4
0.05 = Hybrid FM	55.1 ± 0.6	≈ 4.7	30.4 ± 1.6	40.9 ± 3.7
0.1	57.2 ± 1.6	≈ 17.4	28.5 ± 2.5	38.9 ± 2.9
0.3	57.0 ± 1.2	≈ 6.7	27.4 ± 1.2	37.0 ± 1.7
1.0	62.2 ± 0.7	≈ 10.7	29.4 ± 1.1	37.6 ± 2.0
3.0	60.3 ± 3.7	≈ 17.6	21.3 ± 2.8	23.2 ± 3.3

*Equal-weight average over distances $d = 13, \ldots, 18$, computed from rounded per-distance results; not weighted by the number of evaluation samples.

$d \geq 13$. Task completion is below 1%, while the median end-effector RMSE reaches $0.866\mathrm{m}$, indicating poor motion prediction accuracy.

Replacing the joint-space representation with an end-effector representation provides an initial improvement in the continuous predictions. The median end-effector RMSE decreases from $0.866\mathrm{m}$ to $0.701\mathrm{m}$, while the overall symbolic-plan validity increases slightly from 5% to 6%. However, this improvement is not yet sufficient to produce successful task executions, with both open-loop and closed-loop task-completion rates remaining below 1%. This suggests that changing the motion representation alone improves trajectory accuracy but does not address the limited coverage of the training data.

A substantially larger improvement is obtained after introducing plan-suffix augmentation. By generating training samples from intermediate stages of each demonstration, the effective number and diversity of training examples are increased. With this modification, overall symbolic-plan validity rises from 6% to 55%, while long-horizon validity reaches approximately 4%. The improvement is also reflected in motion execution: open-loop task completion increases to 30%, closed-loop completion reaches 41%, and the median end-effector RMSE decreases substantially to $0.255\mathrm{m}$. These results indicate that plan-suffix augmentation is the implementation change with the largest impact on both symbolic planning and continuous trajectory prediction.

Finally, remasking is introduced during discrete inference to allow low-confidence token predictions to be reconsidered over successive inference iterations. Its effect on the aggregate metrics is comparatively small: overall symbolic validity remains at 55%, while long-horizon validity increases from approximately 4% to 5%. No measurable change is observed in task-completion rate or end-effector RMSE. Nevertheless, the improvement at larger planning horizons suggests that remasking can provide a modest benefit for more difficult symbolic predictions by allowing early uncertain decisions to be revised.

Methods comparison The final Hybrid Flow Matching (Hybrid FM) model generates a valid symbolic plan in 55% of the evaluated samples. As shown in Figure 5.6, performance is substantially higher for shorter planning horizons, reaching approximately 80% or more for sequences with five actions or fewer. This behaviour suggests that the model performs best on configurations that are more strongly represented in the training data. In contrast, without plan-suffix augmentation, symbolic-plan validity remains below 10%, even for comparatively short sequences, highlighting the importance of exposing the model to intermediate stages of the demonstrations during training.

For the continuous component, Hybrid FM achieves a task-completion rate of 30% in open-loop and 41% in closed-loop, with a median end-effector RMSE of $0.255\mathrm{m}$. Figure 5.6 shows that open-loop performance generally improves as the number of actions remaining to the goal decreases. This trend is consistent with the distribution introduced by plan-suffix augmentation, which provides a larger number of training samples corresponding to shorter remaining horizons. A similar tendency is observed under closed-loop execution, although replanning provides a particularly noticeable improvement for medium- and long-horizon sequences. For some configurations requiring approximately five to eight remaining actions, closed-loop execution increases the task-completion rate by up to approximately 20 percentage points relative to open-loop execution.

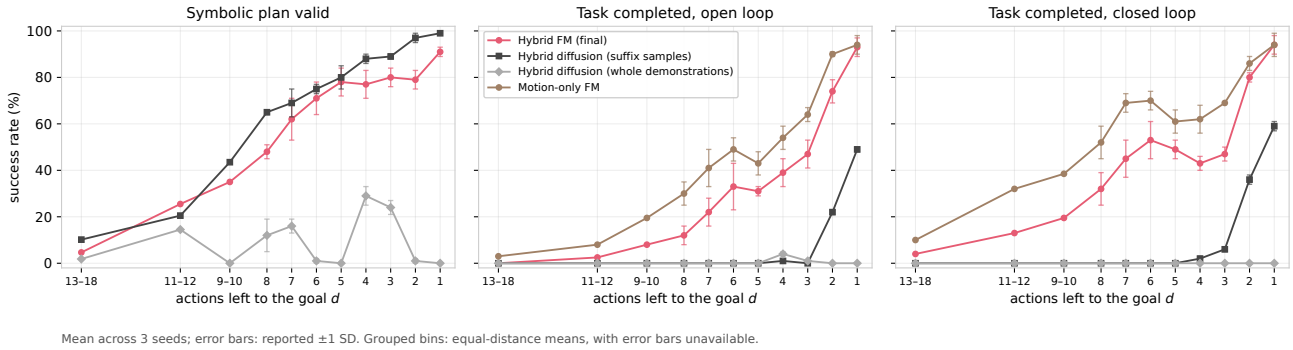


Figure 5.6: Success rate as a function of the number of actions remaining to the goal, evaluated on the 12 validation demonstrations (mean and standard deviation over three training seeds, 50 samples per task). Values of d with few demonstrations are grouped: 13–18, 11–12, and 9–10 hold 850, 750, and 900 tasks, the other values 550 to 600 each. Left: the sampled plan is valid. Middle: the tower stands after the whole sampled trajectory is executed. Right: the tower stands when the policy replans after every action. Hybrid diffusion is the formulation of Høeg et al. [18], trained on our suffix samples or on whole demonstrations. Motion-only FM is flow matching on the trajectory alone; it has no plan and no left panel. Hybrid FM is our Hybrid Flow Matching implementation.

The comparison with Hybrid Diffusion [18] reveals a different balance between symbolic and continuous performance. When Hybrid Diffusion is trained using our suffix-augmented samples, it achieves an overall symbolic-plan validity of 62%, compared with 55% for Hybrid FM, and approximately 10% validity for $d \geq 13$, compared with 5% for Hybrid FM. However, this stronger symbolic performance does not translate into equally accurate motion generation. Hybrid Diffusion achieves only 6% task completion in open-loop and 9% in closed-loop, while its median end-effector RMSE reaches $0.710\mathrm{m}$. In comparison, Hybrid FM attains substantially higher task-completion rates and reduces the median end-effector RMSE to $0.255\mathrm{m}$. These results indicate that, although Hybrid Diffusion produces more valid symbolic plans, Hybrid FM provides a better balance between symbolic planning and continuous trajectory generation for the considered task.

The importance of plan-suffix augmentation is also visible for Hybrid Diffusion itself. When trained only on whole demonstrations, symbolic-plan validity decreases to 8%, while both open-loop and closed-loop task completion remain at 0%. As illustrated in Figure 5.6, most of the limited symbolic success of this configuration is concentrated at very short planning horizons. Introducing suffix samples therefore improves both methods substantially by exposing the models to intermediate task states and shorter remaining horizons.

Several factors may contribute to the difference between the results reported here and those obtained by Høeg et al. [18]. As described in Section 5.2, our dataset consists of human-generated demonstrations, which naturally contain greater variability in trajectory shape and execution accuracy than demonstrations produced by a scripted expert. Moreover, the original method was evaluated using more than 200 demonstrations per task, whereas our complete dataset contains 121 demonstrations distributed across a comparatively broad set of task and motion configurations. The system considered in this work also introduces additional complexity through the combination of a mobile base and a manipulator, increasing the dimensionality of the continuous motion problem and the number of possible spatial configurations.

Despite this increased variability, Hybrid FM performs considerably better on configurations that are sufficiently represented in the training data. In particular, symbolic-plan validity reaches (94%) when evaluated on training demonstrations. Together with the performance degradation observed on validation long-horizon configurations, this suggests that the primary limitation is not the capacity of the model to represent long plans, but rather its ability to generalise to under-represented scene configurations.

Following Høeg et al. [18], we additionally compare the hybrid architecture with a motion-only Flow Matching model in order to evaluate the effect of jointly learning symbolic and continuous outputs. The motion-only model achieves task-completion rates of 39% in open-loop and 52% in closed-loop, outperforming Hybrid FM by 9 and 11 percentage points, respectively. In contrast, the difference in

median end-effector RMSE is relatively small: $0.249\mathrm{m}$ for motion-only FM compared with $0.255\mathrm{m}$ for Hybrid FM, corresponding to approximately $6\mathrm{mm}$.

Although small in absolute terms, differences in motion accuracy can have a considerable effect on task completion in manipulation tasks, where a displacement of only a few centimetres can cause a failed grasp or placement. Moreover, the motion-only model solves a simpler prediction problem, since it does not need to jointly generate a symbolic action sequence. The hybrid model must instead allocate model capacity to both task-level planning and continuous motion generation, making the learning problem more demanding under a limited-data regime.

Overall, the results expose a trade-off between specialisation and joint task-and-motion prediction. Motion-only FM achieves the highest execution success, while Hybrid Diffusion trained with suffix samples achieves the highest symbolic-plan validity. Hybrid FM lies between these two extremes: it produces explicit symbolic plans while maintaining substantially stronger motion-execution performance than Hybrid Diffusion. Further improvements in data coverage and motion accuracy are therefore likely necessary for the hybrid architecture to match or exceed the execution performance of the motion-only model while retaining its task-level planning capability, consistent with the potential of jointly modelling symbolic planning and continuous motion highlighted by Høeg et al. [18].

Table 5.2: Performance on the 12 validation demonstrations, averaged over 3 training seeds and 50 samples per task. Success rates are percentages; EE RMSE is in metres. Values show mean $\pm$ standard deviation across seeds.

	Symbolic plan valid† (%)		Task completed‡ (%)		EE RMSE
Policy	all d	$d \geq 13$*	open-loop	closed-loop	median (m)
Motion-only FM	–	–	39 ± 2	52 ± 3	0.249 ± 0.009
Hybrid diffusion [18]:					
whole demonstrations	8 ± 1	2	0	0	0.812 ± 0.027
our suffix samples	62 ± 1	10	6 ± 0	9 ± 0	0.710 ± 0.003
Hybrid FM (final)	55 ± 1	5	30 ± 2	41 ± 4	0.255 ± 0.010
Cumulative implementation stages					
Joint representation	5 ± 2	1	0	–	0.866 ± 0.025
End-effector representation	6 ± 1	0	0	0	0.701 ± 0.008
+ suffix augmentation	55 ± 1	4	30 ± 2	41 ± 4	0.255 ± 0.010
+ remasking = Hybrid FM (final)	55 ± 1	5	30 ± 2	41 ± 4	0.255 ± 0.010

† The sampled plan is valid under the symbolic model. Motion-only FM predicts no symbolic plan.
‡ The tower stands after execution in the kinematic simulation, of the whole sampled trajectory (open-loop) or with replanning after every action (closed-loop), or waypoint. Joint representation closed-loop results are not reported.
*Equal-weight average over distances $d = 13, \ldots, 18$, computed from rounded per-distance results; not weighted by evaluation sample counts. EE RMSE reports the mean and standard deviation of the per-seed median end-effector RMSE.
Stages are cumulative: end-effector representation replaces joint coordinates with end-effector poses; suffix augmentation adds suffix augmentation data to the dataset demonstrations; remasking adds a remasking pipeline during prediction. Motion-only FM also uses suffix augmentation.

5.4. Real-System Deployment

After evaluating the proposed method in simulation, the next step was to deploy it on the real robot. This required obtaining the relevant environment and robot observations in real time, generating predictions from the trained model, and converting the predicted trajectories into commands that could be executed by the robot.

The deployment pipeline followed the steps below:

1. **Observation acquisition.** The required observations were obtained using the Motion Capture System and the ROS topics previously used during dataset collection. Once all values were acquired, they were stored following a predefined ordering so that each element could be consistently associated with its corresponding source and model input.
2. **Prediction and trajectory processing.** The prediction pipeline was executed in the same manner as for the validation demonstrations during training. Given the current observations, the model generated both a symbolic action sequence and a continuous motion trajectory. The predicted outputs were displayed to the user, after which the continuous end-effector waypoints were processed through inverse kinematics to obtain the corresponding arm joint configurations. These

joint values were then concatenated with the predicted mobile-base pose and transmitted through a ROS topic to the robot execution pipeline.

3. **Trajectory execution.** On the robot, the execution pipeline received the sequence of predicted waypoints to be followed from the current state. Each waypoint was decomposed into the values required by the corresponding low-level interfaces. Motion commands were executed using the existing low-level controller, while gripper state changes were handled separately through the predefined gripper ROS service.
4. **Replanning.** The execution can be stopped and steps 1–3 can be repeated whenever replanning is required.

The motion action receives the six arm joint angles in radians together with the planar base position in meters and the base orientation in radians. The resulting input vector is defined as

$$\mathbf{ac\_}\mathrm{mov} = \{q_1, q_2, q_3, q_4, q_5, q_6, x_{\mathrm{base}}, y_{\mathrm{base}}, \theta_{\mathrm{base}}\}. \tag{5.2}$$

For each waypoint, the controller attempts to drive the arm joints and mobile base toward their corresponding target values simultaneously. During the initial real-system experiments, both the base and arm motions exhibited overshooting. Since this behaviour could obscure the quality of the trajectories generated by the model, several adjustments were introduced to the low-level controller.

- The maximum velocity was reduced to prevent the robot from reaching unnecessarily high speeds during trajectory execution.
- To obtain more precise motion close to the target pose, the commanded velocity was adapted according to the distance between the current robot configuration and the desired waypoint. As the robot approached the target, the commanded velocity was reduced relative to the defined maximum velocity.
- A positional tolerance was introduced to determine when a waypoint could be considered reached. A threshold of $0.05\,\mathrm{m}$ was selected empirically, corresponding to a maximum translational error of approximately $5\,\mathrm{cm}$. This tolerance provided a compromise between accurately reaching the predicted waypoint and avoiding execution failures caused by requiring a level of precision that the low-level controllers could not reliably achieve.

This last point's limitation is particularly relevant when interpreting the real-system results. The low-level controller introduces an execution bottleneck that lies outside the main scope of this work. Even an error of 1–$2\,\mathrm{cm}$ can be sufficient for the end-effector to miss a cube during a *Pick* action or to place an object inaccurately during a *Place* action. Therefore, failures observed on the real robot cannot always be attributed exclusively to the high-level planner, since part of the execution error may originate from the underlying controller.

On the other hand, gripper commands were executed separately from the arm and base motion. Whenever the predicted gripper state differed from that of the previous waypoint, the corresponding open or close command was issued after the robot had reached the corresponding pose. This ensured that grasping and releasing actions were performed only after the required arm and base configuration had been reached.

5.5. Real-System Evaluation

The real-system experiments were conducted using the robot's existing low-level controller. Most of the computation, including model inference and trajectory processing, was performed on a local laptop rather than on the robot's onboard computer to take advantage of its greater computational capacity. An SSH connection was used to remotely access the robot and execute the required components.

As described in Section 5.4, the low-level controller was adjusted prior to evaluation to reduce overshoot during both base and arm motion.

Following these adjustments, the proposed method was evaluated on the real robotic system. The accompanying videos (`https://youtu.be/L-OdmQKRe1c`, `https://youtu.be/Z0Mh8ZYy2-w`) and Figure 5.7 show successful task execution from several initial configurations, with behaviour broadly con-

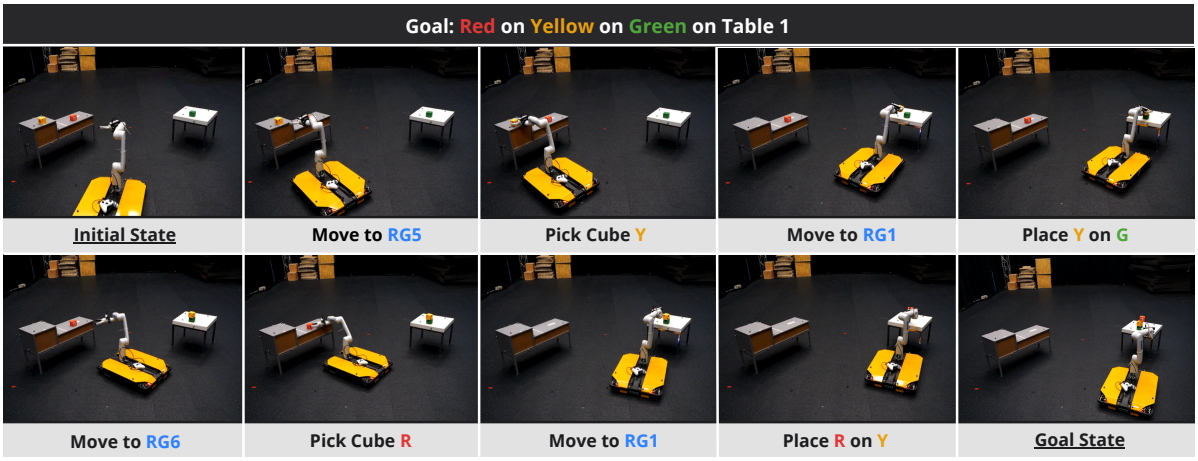


Figure 5.7: Successful execution of a stacking task on the real robot with green cube on Table 1, yellow cube on Table 2 and red cube on Table 3 in the initial state.

sistent with that observed in simulation. These experiments provide preliminary evidence that the kinematic transformations derived from the simulation model are sufficiently accurate for execution in the demonstrated configurations. More importantly, they show that the proposed model can generate both a valid symbolic task plan and a corresponding continuous motion trajectory that can be executed successfully on the real robot.

In these trials, the green cube was initially located on the goal table, Table 1, while the robot manipulated the remaining two cubes to construct the target stack. The task was completed from different initial arrangements of the red and yellow cubes, showing that the method can handle multiple scene configurations rather than reproducing a single fixed sequence.

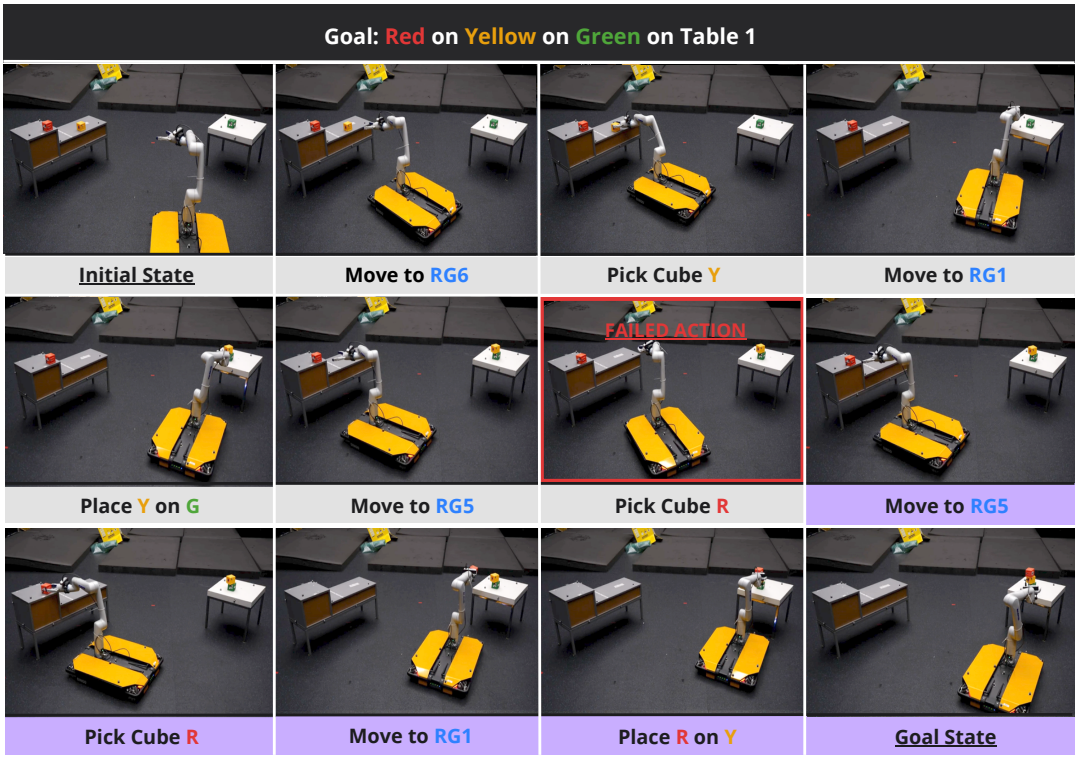


Figure 5.8: Closed-loop behaviour on the real robot. Successful stacking with the green cube initially on Table 1, the yellow cube on Table 3, and the red cube on Table 2, following a failed attempt to pick the red cube.

Nevertheless, the real-system experiments also revealed several limitations. During some *Pick* and *Place* actions, the end-effector deviated from the desired position by approximately 1–$2\,\mathrm{cm}$, which could be sufficient to miss a cube or produce an inaccurate placement (`https://youtu.be/c4HTyrUBSsg`). More challenging initial configurations also resulted in unsuccessful executions. In particular, scenarios requiring the complete three-cube stack to be constructed when none of the cubes was initially located on Table 1 required longer manipulation sequences and proved more difficult for the system. These results are consistent with the degradation observed for longer planning horizons in simulation, as commented in Section 5.3. Limited coverage of similar configurations in the training demonstrations may contribute to these failures, while inaccuracies introduced by the low-level controller can further affect execution. Consequently, the observed failures cannot be attributed exclusively to either the high-level planner or the controller without additional evaluation.

The method additionally supports closed-loop replanning during real-system execution. If a *Pick* action fails, or if a cube is subsequently relocated to one of the other tables, the planner can be queried again using the updated scene configuration. The robot can then generate a new symbolic plan and motion trajectory, re-approach the displaced cube, and continue the task (`https://youtu.be/M47dCfoOzls`, Figure 5.8). This behaviour demonstrates that the system is not restricted to executing the initially generated trajectory open-loop. Instead, replanning at symbolic action boundaries allows it to react to execution failures or changes in the environment and continue progressing toward the task goal.

6

Conclusions

6.1. Conclusion

We presented a Hybrid Flow Matching planner for coupled task and motion planning, trained on 109 human-generated demonstrations and evaluated on 12 validation demonstrations. The dataset contains three categories of initial configurations designed to provide variability in the symbolic task sequences. The experimental environment consists of three tables, three cubes, and a mobile manipulator combining a holonomic base and a robotic arm. To isolate the planning problem while maintaining diversity in the required action sequences and motions, a single task goal was considered: constructing a tower on Table 1 with the red cube on the yellow cube and the yellow cube on the green cube.

The proposed method builds on the Hybrid Diffusion Planner (HDP) of Høeg et al. [18], which jointly generates discrete symbolic plans and continuous trajectories using a hybrid diffusion model. In this work, the diffusion process is replaced by flow matching and the hybrid formulation is applied to coupled mobile manipulation using a dataset collected through human teleoperation. Three main additions are also introduced: modality embeddings to explicitly distinguish the different input modalities within the architecture, suffix data augmentation to increase the effective number of training samples and improve generalisation to intermediate task states, and remasking during discrete inference, which allows low-confidence token predictions to be reconsidered as additional context becomes available throughout the inference process.

On the validation demonstrations, Hybrid Flow Matching achieves task-completion rates of 30% in open-loop execution and 41% in closed-loop execution. In comparison, Hybrid Diffusion trained with the same suffix-augmented data reaches 6% and 9%, respectively. Hybrid Diffusion nevertheless achieves higher symbolic-plan validity, reaching 62% compared with 55% for Hybrid Flow Matching. These results show that, for the considered task and dataset, Hybrid Flow Matching provides substantially stronger continuous-motion generation while retaining the ability to explicitly generate a symbolic task plan.

The motion-only Flow Matching baseline achieves the highest execution performance, with task-completion rates of 39% in open-loop and 52% in closed-loop execution. It also obtains a slightly lower median end-effector RMSE of $0.249\,\mathrm{m}$, compared with $0.255\,\mathrm{m}$ for Hybrid Flow Matching. This indicates that jointly predicting symbolic and continuous outputs introduces an additional learning challenge under the limited-data setting considered in this work. The motion-only model solves a simpler prediction problem, whereas the hybrid model must simultaneously represent two modalities and their interactions throughout the generative process.

Despite this additional complexity, explicitly generating the symbolic sequence provides advantages beyond trajectory prediction alone. In particular, the symbolic plan defines action boundaries at which the system can obtain an updated observation and replan. This enables closed-loop execution without requiring replanning after every continuous waypoint. The generated symbolic sequence also provides an interpretable representation of the robot's intended behaviour, allowing a human observer to under-

stand the sequence of actions being executed.

The proposed method was additionally deployed on the real robotic system. Successful executions were obtained for multiple initial cube configurations, providing evidence that the jointly generated symbolic plans and continuous trajectories can be transferred to the real mobile manipulator. The real-system experiments also exposed limitations in both motion accuracy and long-horizon generalisation, with small end-effector execution errors and configurations requiring longer action sequences remaining particularly challenging.

Overall, the results support the potential of jointly modelling symbolic and continuous planning highlighted by Høeg et al. [18], while also showing the challenges introduced by joint prediction in a limited-data mobile-manipulation setting where limited generalisation appears to be the main bottleneck. Further improvements may help close the remaining performance gap with motion-only Flow Matching while retaining the explicit task-level planning and replanning capabilities of the Hybrid Flow Matching formulation.

6.2. Future work

- **Incorporation of visual observations:** During dataset collection, images were recorded from both a wrist-mounted camera and an external camera observing the scene. These images were initially intended to form part of the model observations, providing richer information about the state of the environment. A possible extension would therefore be to incorporate visual observations into the model by encoding them with the patchifier used for the continuous data and combining these features with the remaining state observations.

  Including visual information could be particularly beneficial during closed-loop execution, where updated images would provide additional information about intermediate scene configurations after each executed action. This could improve the model's ability to react to deviations from the predicted trajectory or changes in object positions. However, the increased dimensionality and complexity of the observation space may also require a larger and more diverse set of demonstrations.
- **Including a perception pipeline:** A further extension would be to reduce the reliance on explicitly provided object poses with the Motion Capture System. Instead of supplying the model with ground-truth object positions, visual representations extracted directly from the camera observations could be used. Pre-trained visual encoders, such as DINOv2, could provide informative image features that are subsequently used by the policy, while dedicated object detection or pose-estimation methods could be used to estimate the required scene state directly from visual observations. This would move the system toward a more self-contained perception-to-action pipeline and reduce its dependence on external tracking infrastructure [28].
- **Additional data augmentation strategies:** As discussed in Section 2, further strategies could be investigated to improve data efficiency beyond the plan-suffix augmentation used in the final method. Geometric transformations of the scene, such as rotations for configurations in which the task remains invariant, could provide additional training examples without requiring new demonstrations. Object permutations could also be considered by exchanging the roles of the different cubes. In this case, however, the desired final stacking order would need to be explicitly provided to the model as part of the goal specification. Such augmentation strategies could increase the diversity of the training distribution and potentially improve generalisation to unseen configurations [28].
- **Adaptive balancing of discrete and continuous losses:** The current implementation follows the combined-loss formulation used in [18], where a fixed weighting factor determines the contribution of the discrete loss to the final loss. Although this formulation already provides satisfactory results, a fixed weighting may not remain optimal throughout the entire training process. An adaptive loss-balancing strategy could instead dynamically adjust the relative contribution of the two objectives, which could help prevent one component from dominating the optimisation while the other remains under-trained, and may lead to a better balance between symbolic-plan prediction and continuous-trajectory generation [28].

- **Grammar-constrained discrete generation:** A further improvement to the current implementation could be to constrain the discrete predictions according to the structure of the symbolic action representation. Currently, each revealed token is selected from the complete discrete vocabulary, without explicitly enforcing constraints on which token types may follow one another. Incorporating such constraints could reduce the generation of structurally invalid symbolic plans and potentially improve discrete-plan validity.

  For example, if a *Move_to* token is generated, the following token could be restricted to an approach-region identifier, such as *RG1*, *RG2*, …, *RG10*. Similar constraints could be defined for the remaining actions according to their token structure, as described in Section 5.2. Likewise, the *EOS* token could be restricted to positions following the completion of a valid action, preventing the sequence from terminating in the middle of an action definition. Such grammar-constrained decoding would incorporate prior knowledge about the symbolic representation directly into the generation process [28].
- **Improved demonstration coverage:** As described in Section 5.2, the collected dataset covers a large number of task-level configurations involving the three tables and three cubes. At the motion level, however, the number of possible trajectories and object arrangements is considerably larger, while the final dataset contains only 121 demonstrations. This variability is further increased by the use of human-generated demonstrations, which naturally introduce differences in trajectory shape, execution speed, and precision compared with the more consistent behaviour produced by a scripted expert, as used in [18]. The distribution of these demonstrations is shown in Figure 5.4. Future work could therefore reconsider how the available demonstrations are distributed across the task space. Reducing the number of distinct task-level configurations while collecting more motion-level variation for each configuration could provide denser coverage of the corresponding state and trajectory distributions, which would likely improve generalisation, particularly for long-horizon configurations that are currently under-represented in the dataset .

References

[1] Muhammad Zain Amin and Amir Ali. “Application of Multilayer Perceptron (MLP) for Data Mining in Healthcare Operations”. In: *Proceedings of the 3rd International Conference on Biotechnology*. University of South Asia. Lahore, Pakistan, Feb. 2017, pp. 2–11.

[2] Marcin Andrychowicz et al. “Hindsight Experience Replay”. In: *Adv. Neural Inf. Process. Syst. (NeurIPS)*. Vol. 30. 2017.

[3] Anonymous. *HDFlow: Hierarchical Diffusion-Flow Planning for Long-horizon Robotic Assembly*. 2026. URL: https://openreview.net/forum?id=nokbt6AbcM.

[4] Autonomous Multi-Robots Lab. *Our Robots and Installations*. https://autonomousrobots.nl/robots/. Accessed: 2026-08-28. 2023.

[5] Kevin Black, Noah Brown, Danny Driess, et al. “π_0: A Vision-Language-Action Flow Model for General Robot Control”. In: *Proc. Robot.: Sci. Syst. (RSS)*. 2025. DOI: 10.15607/RSS.2025.XXI.010.

[6] Giovanni Buizza Avanzini, Andrea Maria Zanchettin, and Paolo Rocco. “Reactive constrained model predictive control for redundant mobile manipulators”. In: *Intelligent Autonomous Systems 13: Proceedings of the 13th International Conference IAS-13*. Springer. 2015, pp. 1301–1314.

[7] C. Chi et al. “Diffusion policy: Visuomotor policy learning via action diffusion”. In: *The International Journal of Robotics Research* 44.10-11 (2025), pp. 1684–1704. DOI: 10.1177/02783649241273668.

[8] Rohan Chitnis et al. “Guided Search for Task and Motion Plans Using Learned Heuristics”. In: *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. 2016, pp. 447–454. DOI: 10.1109/ICRA.2016.7487165.

[9] Murtaza Dalal et al. “Imitating Task and Motion Planning with Visuomotor Transformers”. In: *Proc. Conf. Robot Learn. (CoRL)*. 2023.

[10] Alexey Dosovitskiy et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=YicbFdNTTy.

[11] Danny Driess, Jung-Su Ha, and Marc Toussaint. “Deep Visual Reasoning: Learning to Predict Action Sequences for Task and Motion Planning from an Initial Scene Image”. In: *Proc. Robot.: Sci. Syst. (RSS)*. 2020. DOI: 10.15607/RSS.2020.XVI.003.

[12] Wayne H Enright. *The Numerical Analysis of Ordinary Differential Equations: Runge-Kutta and General Linear Methods (JC Butcher)*. 1989.

[13] Lutfi Eren Erdogan et al. “Plan-and-Act: Improving Planning of Agents for Long-Horizon Tasks”. In: *Proceedings of the 42nd International Conference on Machine Learning*. Ed. by Aarti Singh et al. Vol. 267. Proceedings of Machine Learning Research. PMLR, July 2025, pp. 15419–15462. URL: https://proceedings.mlr.press/v267/erdogan25a.html.

[14] Zeyu Feng et al. “LTLDoG: Satisfying temporally-extended symbolic constraints for safe diffusion-based planning”. In: *IEEE Robotics and Automation Letters* (2024).

[15] Zipeng Fu, Tony Z. Zhao, and Chelsea Finn. “Mobile ALOHA: Learning Bimanual Mobile Manipulation using Low-Cost Whole-Body Teleoperation”. In: *Proc. Conf. Robot Learn. (CoRL)*. 2024, pp. 4066–4083.

[16] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. “PDDLStream: Integrating Symbolic Planners and Blackbox Samplers via Optimistic Adaptive Planning”. In: *Proc. Int. Conf. Autom. Plan. Sched. (ICAPS)*. 2020.

[17] Caelan Reed Garrett et al. “Integrated task and motion planning”. In: *Annual review of control, robotics, and autonomous systems* 4.1 (2021), pp. 265–293.

[18] Sigmund H. Høeg et al. “Hybrid Diffusion for Simultaneous Symbolic and Continuous Planning”. In: *IEEE Robotics and Automation Letters* 11.4 (Apr. 2026), pp. 4489–4496. ISSN: 2377-3774. DOI: 10.1109/lra.2026.3664616. URL: http://dx.doi.org/10.1109/LRA.2026.3664616.

[19] Michael Janner et al. *Planning with Diffusion for Flexible Behavior Synthesis*. 2022. arXiv: 2205.09991 [cs.LG]. URL: https://arxiv.org/abs/2205.09991.

[20] Leslie Pack Kaelbling and Tomás Lozano-Pérez. “Hierarchical Task and Motion Planning in the Now”. In: *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. 2011, pp. 1470–1477. DOI: 10.1109/ICRA.2011.5980391.

[21] Qinghua Li et al. “A hierarchical motion planning for mobile manipulator”. In: *IEEJ Transactions on Electrical and Electronic Engineering* 15.9 (2020), pp. 1390–1399.

[22] Zhixuan Liang et al. “AdaptDiffuser: Diffusion Models as Adaptive Self-evolving Planners”. In: *Proceedings of the 40th International Conference on Machine Learning*. Ed. by Andreas Krause et al. Vol. 202. Proceedings of Machine Learning Research. PMLR, July 2023, pp. 20725–20745. URL: https://proceedings.mlr.press/v202/liang23e.html.

[23] Zhixuan Liang et al. “SkillDiffuser: Interpretable Hierarchical Planning via Skill Abstractions in Diffusion-Based Task Execution”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2024, pp. 16467–16476.

[24] Yaron Lipman et al. “Flow Matching for Generative Modeling”. English. In: 11th International Conference on Learning Representations, ICLR 2023 ; Conference date: 01-05-2023 Through 05-05-2023. May 2023.

[25] I. Mishani, Y. Shaoul, and M. Likhachev. “MOSAIC: Skill-Centric Manipulation Planning with Physics Simulation (Extended Abstract)”. In: *Proceedings of the International Symposium on Combinatorial Search*. Vol. 19. 1. 2026, pp. 308–309. DOI: 10.1609/socs.v19i1.43110.

[26] Utkarsh Aashu Mishra et al. “Generative Skill Chaining: Long-Horizon Skill Planning with Diffusion Models”. In: *Proceedings of The 7th Conference on Robot Learning*. Ed. by Jie Tan, Marc Toussaint, and Kourosh Darvish. Vol. 229. Proceedings of Machine Learning Research. PMLR, Nov. 2023, pp. 2905–2925. URL: https://proceedings.mlr.press/v229/mishra23a.html.

[27] PickNik Robotics. *Meta Quest Teleoperation*. https://github.com/PickNikRobotics/meta_quest_teleoperation.

[28] Zuleika Redondo Garcia, Andreu Matoses Gimenez, and Javier Alonso-Mora. *Learning Task and Motion Plans from Real Demonstrations with Hybrid Flow Matching*. Oct. 2026. arXiv: 2610.04771 [cs.RO]. URL: https://arxiv.org/abs/2610.04771.

[29] Thushara Sandakalum and Marcelo H. Ang. “Motion Planning for Mobile Manipulators—A Systematic Review”. In: *Machines* 10.2 (2022). ISSN: 2075-1702. DOI: 10.3390/machines10020097. URL: https://www.mdpi.com/2075-1702/10/2/97.

[30] S P Sharan et al. “Plan Diffuser: Grounding LLM Planners with Diffusion Models for Robotic Manipulation”. In: *Bridging the Gap between Cognitive Science and Robot Learning in the Real World: Progresses and New Directions*. 2024. URL: https://openreview.net/forum?id=2a3sgm5YeX.

[31] Jiaxin Shi et al. “Simplified and Generalized Masked Diffusion for Discrete Data”. In: *Advances in Neural Information Processing Systems*. Ed. by A. Globerson et al. Vol. 37. Curran Associates, Inc., 2024, pp. 103131–103167. DOI: 10.52202/079017-3277. URL: https://proceedings.neurips.cc/paper_files/paper/2024/file/bad233b9849f019aead5e5cc60cef70f-Paper-Conference.pdf.

[32] Anthony Simeonov et al. “A Long Horizon Planning Framework for Manipulating Rigid Pointcloud Objects”. In: *Proceedings of the 2020 Conference on Robot Learning*. Ed. by Jens Kober, Fabio Ramos, and Claire Tomlin. Vol. 155. Proceedings of Machine Learning Research. PMLR, Nov. 2021, pp. 1582–1601. URL: https://proceedings.mlr.press/v155/simeonov21a.html.

[33] Max Spahn, Bruno Brito, and Javier Alonso-Mora. “Coupled Mobile Manipulation via Trajectory Optimization with Free Space Decomposition”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 12759–12765. DOI: 10.1109/ICRA48506.2021.9561821.

[34] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[35] Siddharth Srivastava et al. “Combined task and motion planning through an extensible planner-independent interface layer”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 639–646. DOI: 10.1109/ICRA.2014.6906922.

[36] Zhe Tang et al. “Deep Cross-Attention Network for Crowdfunding Success Prediction”. In: *IEEE Transactions on Multimedia* 25 (2023), pp. 1306–1319. DOI: 10.1109/TMM.2022.3141256.

[37] Marc Toussaint. “Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning.” In: *IJCAI*. 2015, pp. 1930–1936.

[38] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[39] Dian Wang et al. “Equivariant Diffusion Policy”. In: *Proc. Conf. Robot Learn. (CoRL)*. 2024.

[40] Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. *Diffusion Policies as an Expressive Policy Class for Offline Reinforcement Learning*. 2023. arXiv: 2208.06193 [cs.LG]. URL: https://arxiv.org/abs/2208.06193.

[41] Shuangyu Xie et al. “Coupled Active Perception and Manipulation Planning for a Mobile Manipulator in Precision Agriculture Applications”. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. 2024, pp. 12665–12671. DOI: 10.1109/ICRA57147.2024.10610753.

[42] Zongwu Xie et al. “Diffusion as reasoning: Enhancing object navigation via diffusion model conditioned on LLM-based object-room knowledge”. In: *Knowledge-Based Systems* 340 (2026), p. 115691. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2026.115691. URL: https://www.sciencedirect.com/science/article/pii/S0950705126004314.

[43] Tony Z. Zhao et al. “Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware”. In: *Proc. Robot.: Sci. Syst. (RSS)*. 2023. DOI: 10.15607/RSS.2023.XIX.016.

A

Training and Inference Pseudocode

Algorithm 1 Hybrid continuous–discrete training step

Require: Training dataset $\mathcal{D}$, model f_θ, optimizer, discrete-loss scale λ_d
Ensure: Updated model parameters θ

Sample a batch $\{\mathbf{z}_b^c, \mathbf{z}_b^d, \mathbf{o}_{b,0}, \mathbf{m}_b^c, \mathbf{m}_b^d\}_{b=1}^B$ from $\mathcal{D}$
for $b = 1, \dots, B$ **do**
    Sample $t_b^c, t_b^d \sim \mathcal{U}(0, 1-\varepsilon)$ independently
    if $\gamma_b \sim \mathrm{Bernoulli}(0.25) = 1$ **then**
        Sample $\tilde{t}_b \sim \mathcal{U}(0, 0.2)$ and set $t_b^c, t_b^d \leftarrow \tilde{t}_b$
    end if
    Sample $\mathbf{x}_{0,b}^c \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    $\mathbf{x}_{t_c,b}^c \leftarrow (1 - t_b^c)\mathbf{x}_{0,b}^c + t_b^c \mathbf{z}_b^c$
    $\mathbf{u}_b \leftarrow \mathbf{z}_b^c - \mathbf{x}_{0,b}^c$
    $\mathbf{x}_{t_d,b}^d \leftarrow \mathbf{z}_b^d$
    for $i = 1, \dots, N_{\max}$ **do**
        if $m_{b,i}^d = 1$ **then**
            $r_{b,i} \sim \mathrm{Bernoulli}(t_b^d)$
            if $r_{b,i} = 0$ **then**
                $x_{t,b,i}^d \leftarrow x_{\mathrm{mask}}$
            end if
        else
            $x_{t_d,b,i}^d \leftarrow x_{\mathrm{mask}}$
            $r_{b,i} \leftarrow 1$
        end if
    end for
    $(\widehat{\mathbf{v}}_b, \boldsymbol{\ell}_b) \leftarrow f_\theta(\mathbf{x}_{t,b}^c, \mathbf{x}_{t,b}^d, t_b^c, t_b^d, \mathbf{o}_{b,0})$
    Compute $\mathcal{L}_b^c$ using Eq. (4.23)
    Compute $\mathcal{L}_b^d$ using Eq. (4.21)
end for
$\mathcal{L}^c \leftarrow \frac{1}{B}\sum_{b=1}^B \mathcal{L}_b^c, \quad \mathcal{L}^d \leftarrow \frac{1}{B}\sum_{b=1}^B \mathcal{L}_b^d$
$\mathcal{L} \leftarrow \mathcal{L}^c + \lambda_d \mathcal{L}^d$
Reset gradients and backpropagate $\mathcal{L}$
Update θ using the optimizer
return θ

Algorithm 2 Hybrid continuous–discrete prediction step

Require: Initial-state observation $\mathbf{o}_0$, trained model f_θ, number of inference steps K, discrete horizon N_d, remasking iterations $\mathcal{R}$
Ensure: Continuous trajectory $\mathbf{x}_K^c$ and discrete plan $\mathbf{x}_{\text{final}}^d$

Sample $\mathbf{x}_0^c \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
Initialize $x_{0,j}^d \leftarrow x_{\text{mask}}$, $j = 1, \ldots, N_d$
Define time schedules $\{t_k^c\}_{k=0}^K$ and $\{t_k^d\}_{k=0}^K$
for $k = 0, \ldots, K-1$ **do**
    Store the previously revealed positions:

$$\mathcal{U}_k \leftarrow \left\{ j \mid x_{k,j}^d \neq x_{\text{mask}} \right\}$$

    $(\widehat{\mathbf{v}}_k, \boldsymbol{\ell}_k) \leftarrow f_\theta(\mathbf{x}_k^c, \mathbf{x}_k^d, t_k^c, t_k^d, \mathbf{o}_0)$
    $\Delta t_k^c \leftarrow t_{k+1}^c - t_k^c = \frac{1}{K}$
    $\mathbf{x}_{k+1}^c \leftarrow \mathbf{x}_k^c + \Delta t_k^c \widehat{\mathbf{v}}_k$
    $\widehat{\mathbf{p}}_k \leftarrow \text{softmax}(\boldsymbol{\ell}_k)$
    Determine the unmasking probability q_k
    Select a subset $\mathcal{S}_k$ from the positions that remain masked according to q_k
    $\mathbf{x}_{k+1}^d \leftarrow \mathbf{x}_k^d$
    for each $j \in \mathcal{S}_k$ **do**
        $x_{k+1,j}^d \leftarrow \arg\max_c \widehat{p}_{k,j}(c)$
    end for
    if $k \in \mathcal{R}$ and $\mathcal{U}_k \neq \emptyset$ **then**
        for each $j \in \mathcal{U}_k$ **do**
            $c_{k,j} \leftarrow \widehat{p}_{k,j}\left(x_{k,j}^d\right)$
        end for
        $n_{\text{remask}} \leftarrow \max\left(1, \lceil 0.1 |\mathcal{U}_k| \rceil\right)$
        $\mathcal{Q}_k \leftarrow$ positions of the n_{remask} lowest values of $c_{k,j}$
        for each $j \in \mathcal{Q}_k$ **do**
            $x_{k+1,j}^d \leftarrow x_{\text{mask}}$
        end for
        Adapt the discrete generative time t_{k+1}^d according to the remasking procedure.
    end if
end for
Let j_{EOS} denote the position of the first *EOS* token in $\mathbf{x}_K^d$
if an *EOS* token is present **then**
    $\mathbf{x}_{\text{final}}^d \leftarrow \mathbf{x}_{K,1:j_{\text{EOS}}}^d$
else
    $\mathbf{x}_{\text{final}}^d \leftarrow \mathbf{x}_K^d$
end if
return $\mathbf{x}_K^c, \mathbf{x}_{\text{final}}^d$