Transformer-based Model Predictive Control: Trajectory Optimization via Sequence Modeling

Davide Celestini[1*], Daniele Gammelli[2*], Tommaso Guffanti[2], Simone D'Amico[2], Elisa Capello[1], and Marco Pavone[2]

***Abstract*—Model predictive control (MPC) has established itself as the primary methodology for constrained control, enabling general-purpose robot autonomy in diverse real-world scenarios. However, for most problems of interest, MPC relies on the recursive solution of highly non-convex trajectory optimization problems, leading to high computational complexity and strong dependency on initialization. In this work, we present a unified framework to combine the main strengths of optimization-based and learning-based methods for MPC. Our approach entails embedding high-capacity, transformer-based neural network models within the optimization process for trajectory generation, whereby the transformer provides a near-optimal initial guess, or target plan, to a non-convex optimization problem. Our experiments, performed in simulation and the real world onboard a free-flyer platform, demonstrate the capabilities of our framework to improve MPC convergence and runtime. Compared to purely optimization-based approaches, results show that our approach can improve trajectory generation performance by up to 75%, reduce the number of solver iterations by up to 45%, and improve overall MPC runtime by 7x without loss in performance.**
Project website, code, and videos: https://transformermpc.github.io

***Index Terms*—Optimization and Optimal Control, Deep Learning Methods, Machine Learning for Robot Control**

I. Introduction

Trajectory generation is crucial to achieving reliable robot autonomy, endowing autonomous systems with the capability to compute a state and control trajectory that simultaneously satisfies constraints and optimizes mission objectives. As a result, trajectory generation problems have been formulated in many practical areas, including space, aerial, and underwater vehicles [2], [3], [4], robot motion planning [5], building climate control [6], chemical processes [7], and more. Crucially, the ability to solve the trajectory generation problem in real time is pivotal to safely operate within real-world scenarios, allowing the autonomous system to rapidly recompute an optimal plan based on the most recent information.

Motivated by its widespread applications, a collection of highly effective solution strategies exist for the trajectory generation problem. For example, numerical optimization provides a systematic mathematical framework to specify mission objectives as costs or rewards and enforce state and control specifications via constraints [8]. However, for most problems of interest, the trajectory optimization problem is almost always nonconvex, leading to high computational complexity, strong dependency on initialization, and a lack of guarantees of either obtaining a solution or certifying that a solution does not exist [9]. The above limitations are further exacerbated in the case of model predictive control (MPC) formulations, where the trajectory generation problem needs to be solved repeatedly and in real-time as the mission evolves, enforcing strong requirements on computation time. Moreover, MPC formulations typically require significant manual trial-and-error in defining ad-hoc terminal constraints and cost terms to, e.g., achieve recursive feasibility or exhibit short-horizon behavior that aligns with the full trajectory generation problem.

Beyond methods based on numerical optimization, recent advances in machine learning (ML) have motivated the application of learning-based methods to the trajectory generation problem [10]. ML approaches are typically highly computationally efficient and can be optimized for (potentially nonconvex) performance metrics from high-dimensional data (e.g., images). However, learning-based methods are often sensitive to distribution shifts in unpredictable ways, whereas optimization-based approaches are more readily characterized both in terms of robustness and out-of-distribution behavior. Additionally, ML methods often perform worse on these problems due to their high-dimensional action space, which increases variance in, e.g., policy-gradient algorithms [11], [12]. As a result, real-world deployment of learning-based methods has so far been limited within safety-critical applications.

In this work, we propose a framework to exploit the specific strengths of optimization-based and learning-based methods for trajectory generation, specifically tailored for MPC formulations (Fig. 1). By extending the framework introduced in [1], we propose a pre-train-plus-fine-tuning strategy to train a transformer to generate near-optimal state and control sequences and show how this allows to (i) warm-start the optimization with a close-to-optimal initial guess, i.e., leading to improved performance and faster convergence, and (ii) provide *long-horizon* guidance to *short-horizon* problems in MPC formulations, avoiding the need for expensive tuning of cost terms or constraints within the optimization process. Crucially, we show how the proposed fine-tuning scheme results in substantially improved robustness to distribution shifts caused by closed-loop execution and how the injection of learning-based guidance within MPC formulations drastically decreases the loss in performance due to the reduction of the planning horizon, enabling the solution of substantially smaller optimization problems without sacrificing performance.

The contributions of this paper are threefold:

- We present a framework to combine the strengths of *offline* learning and *online* optimization for efficient trajectory generation within model predictive control pipelines.
- We investigate design decisions and learning strategies within our framework, such as the effects of fine-tuning on MPC execution, the benefits of learned terminal cost definitions, and their impact on performance.

* Equal contribution.

[1] Department of Mechanical and Aerospace Engineering, Politecnico di Torino, 10129 Torino, Italy. e-mail : davide.celestini@polito.it, elisa.capello@polito.it

[2] Department of Aeronautics and Astronautics, Stanford University, 94305, USA. e-mail : gammelli@stanford.edu, tommaso@stanford.edu, damicos@stanford.edu, pavone@stanford.edu

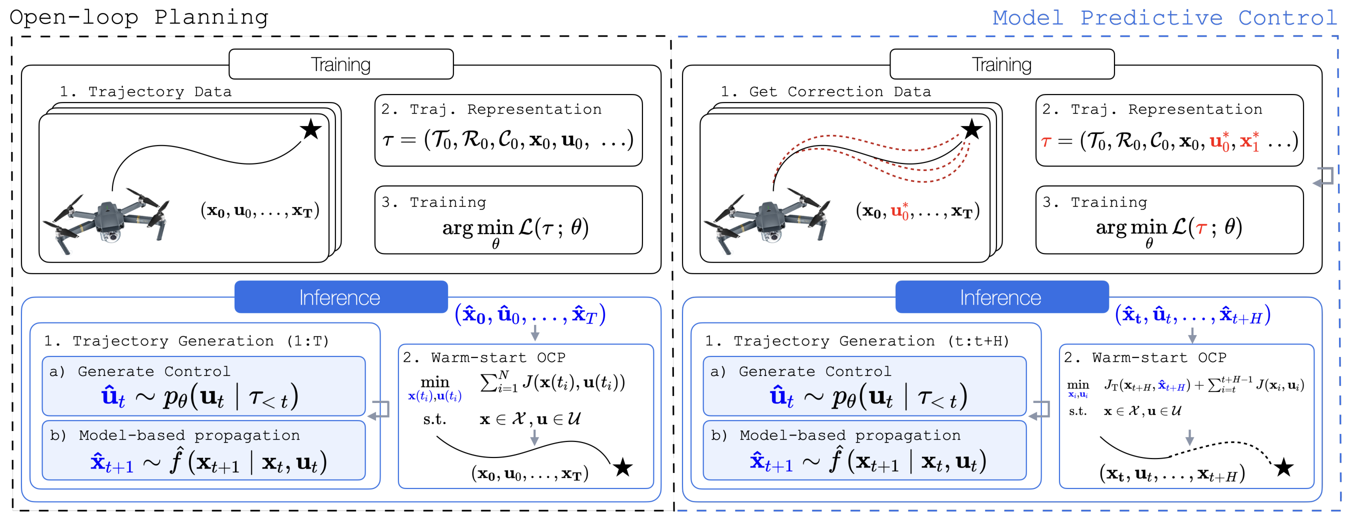
Open-loop Planning
Model Predictive Control
Training
1. Trajectory Data
$(\mathbf{x}_0, \mathbf{u}_0, \ldots, \mathbf{x}_T)$
2. Traj. Representation
$\tau = (\mathcal{T}_0, \mathcal{R}_0, \mathcal{C}_0, \mathbf{x}_0, \mathbf{u}_0, \ldots)$
3. Training
$\arg\min_\theta \mathcal{L}(\tau\,;\,\theta)$
1. Get Correction Data
$(\mathbf{x}_0, \mathbf{u}_0^*, \ldots, \mathbf{x}_T)$
$\tau = (\mathcal{T}_0, \mathcal{R}_0, \mathcal{C}_0, \mathbf{x}_0, \mathbf{u}_0^*, \mathbf{x}_1^* \ldots)$
Inference
$(\hat{\mathbf{x}}_0, \hat{\mathbf{u}}_0, \ldots, \hat{\mathbf{x}}_T)$
1. Trajectory Generation (1:T)
a) Generate Control
$\hat{\mathbf{u}}_t \sim p_\theta(\mathbf{u}_t \mid \tau_{<t})$
b) Model-based propagation
$\hat{\mathbf{x}}_{t+1} \sim \hat{f}(\mathbf{x}_{t+1} \mid \mathbf{x}_t, \mathbf{u}_t)$
2. Warm-start OCP
$\min_{\mathbf{x}(t_i),\mathbf{u}(t_i)} \sum_{i=1}^N J(\mathbf{x}(t_i), \mathbf{u}(t_i))$
s.t. $\mathbf{x} \in \mathcal{X}, \mathbf{u} \in \mathcal{U}$
$(\mathbf{x}_0, \mathbf{u}_0, \ldots, \mathbf{x}_T)$
$(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t, \ldots, \hat{\mathbf{x}}_{t+H})$
1. Trajectory Generation (t:t+H)
$\min_{\mathbf{x}_i,\mathbf{u}_i} J_T(\mathbf{x}_{t+H}, \hat{\mathbf{x}}_{t+H}) + \sum_{i=t}^{t+H-1} J(\mathbf{x}_i, \mathbf{u}_i)$
$(\mathbf{x}_t, \mathbf{u}_t, \ldots, \mathbf{x}_{t+H})$

Fig. 1: We propose a framework to combine high-capacity sequence models and optimization-based methods for MPC. The core idea is to train a transformer model to generate near-optimal trajectories (top), which can then be used to warm-start trajectory optimization problems at inference (bottom). We leverage methods introduced in [1] as a pre-training step, whereby a transformer is trained on pre-collected (open-loop) trajectory data (top left) to provide an initial guess for the full optimal control problem (OCP) (bottom left). For effective MPC, we fine-tune the model through closed-loop corrections (top right) and use it to provide both an initial guess for the OCP and a target state to approximate future cost within the short-sighted problem (bottom right).

- Through experiments performed in simulation (i.e., spacecraft rendezvous and quadrotor control scenarios) and real-world robotic platforms (i.e., a free flyer testbed), we show how our framework substantially improves the performance of off-the-shelf trajectory optimization methods in terms of cost, runtime, and convergence rates, demonstrating its applicability within real-world scenarios characterized by nonlinear dynamics and constraints.

II. Related Work

Our work is closely related to previous approaches that aim to combine learning and optimization for control [13], [14], [15], [16], exploit high-capacity neural network models for control [17], [18], [19], and methods for warm-starting nonlinear optimization problems [20], [21], [1], providing a way to train general-purpose trajectory generation models that guide the solution of inner optimization problems.

Numerous strategies have been developed for learning to control via formulations that leverage optimization-based planning as an inner component. For instance, prior work focuses on developing hierarchical formulations, where a high-level (learning-based) module generates a waypoint-like representation for a low-level planner, e.g., based on sampling-based search [13], [15], trajectory optimization [16], or model-based planning [14]. Within this context, the learning-based component is typically trained through either (online) reinforcement learning or imitation of oracle guidance algorithms. An alternate strategy consists of directly optimizing through an inner controller. A large body of work has focused on exploiting exact solutions to the gradient of (convex) optimization problems at fixed points [22], [23], allowing the inner optimization to be used as a generic component in a differentiable computation graph (e.g., a neural network). Analogously to previous work, our approach uses the output of a learning-based module as input to an optimization problem. However, we focus on providing an initial guess for the unmodified trajectory optimization problem, avoiding the misalignment caused by intermediate objectives that do not necessarily correlate with the downstream task (e.g., fixed-point iteration in value function learning).

Our method is closely related to recent work that leverages high-capacity generative models for control. For example, prior work has shown how transformers [17], [18] and diffusion models [19] trained via supervised learning on pre-collected trajectory data are amenable to both model-free feedback control [17] or (discrete) model-based planning [18]. These methods have two main drawbacks: (i) they typically ignore non-trivial state-dependent constraints, a setting that is both extremely common in practice and challenging for purely learning-based methods and (ii) they do not exploit all the information available to system designers, which typically includes approximate knowledge of the system dynamics. As in [1], our method alleviates both of these shortcomings by (i) leveraging online trajectory optimization to enforce non-trivial constraint satisfaction, and (ii) taking a model-based view to transformer-based trajectory generation and leveraging readily available approximations of the system dynamics to improve autoregressive generation.

Lastly, prior work has explored the idea of using ML to warm-start the solution of optimization problems [20], [21], [1]. While the concept of warm-starting optimization solvers is appealing in principle, current approaches are typically (i) limited by the choice of representation for the output of the ML model (e.g., a fixed degree polynomial), and (ii) restricted to the open-loop planning setting [1], whereby the impact of distribution shifts due to closed-loop execution [24] (e.g., model mismatch, stochasticity, etc.) is often overlooked. To address these limitations, we present an extended and revised version of [1], whereby we (1) introduce a pre-train-plus-fine-tuning learning strategy to maximize closed-loop performance, (2) develop an MPC formulation that enables the transformer to provide long-horizon guidance to short-horizon problems, and (3) augment the trajectory representation and the transformer architecture to handle conditioning on a target state. Crucially, we take full advantage of the transformer's autoregressive generation capabilities within receding horizon control formulations, enabling the same model to be used with varying planning horizon specifications.

III. Problem Statement

Let us consider the time-discrete optimal control problem (OCP):

$$\underset{\mathbf{x}(t_i),\mathbf{u}(t_i)}{\text{minimize}} \quad \mathcal{J}=\sum_{i=1}^{N} J(\mathbf{x}(t_i),\mathbf{u}(t_i)) \tag{1a}$$

$$\text{subject to} \quad \mathbf{x}(t_{i+1})=\mathbf{f}(\mathbf{x}(t_i),\mathbf{u}(t_i)) \quad \forall i\in[1,N], \tag{1b}$$

$$\mathbf{x}(t_i)\in\mathcal{X}_{t_i},\mathbf{u}(t_i)\in\mathcal{U}_{t_i} \quad \forall i\in[1,N], \tag{1c}$$

where $\mathbf{x}(t_i) \in \mathbb{R}^{n_x}$ and $\mathbf{u}(t_i) \in \mathbb{R}^{n_u}$ are respectively the n_x-dimensional state and n_u-dimensional control vectors, $J:\mathbb{R}^{n_x+n_u}\to\mathbb{R}$ defines the running cost, $\mathbf{f}:\mathbb{R}^{n_x+n_u}\to\mathbb{R}^{n_x}$ represents the system dynamics, $\mathcal{X}_{t_i}$ and $\mathcal{U}_{t_i}$ are generic state and control constraint sets, and where $N\in\mathbb{N}$ defines the number of discrete time instants t_i over the full OCP horizon T.

In particular, we consider a receding-horizon reformulation of Problem (1), whereby we repeatedly solve an OCP characterized by a moving (and typically shorter) horizon $\mathcal{H}=[h,h+H]\subseteq[1,N]$, with h being the moving initial timestep and $H\in(0,N]$ the length of the horizon. To ensure desirable long-horizon behavior from the solution of a series of short-horizon problems, MPC formulations typically require the definition of a terminal cost J_T or constraint set $\mathcal{X}_{h+H}$. Specifically, we will consider cost functions of the form:

$$\mathcal{J}=J_\text{T}(\mathbf{x}(t_{h+H}))+\sum_{i=h}^{h+H-1} J(\mathbf{x}(t_i),\mathbf{u}(t_i)), \tag{2}$$

where the running cost J is evaluated exclusively over $\mathcal{H}$ and $J_\text{T}:\mathbb{R}^{n_x}\to\mathbb{R}$ is the terminal cost function.

In this work, we explore approaches to address Problem (2) by incorporating high-capacity neural network models within the optimization process for trajectory generation, explicitly tailored for closed-loop performance.

IV. Trajectory Optimization via Sequence Modeling

We consider a strategy for trajectory generation whereby state and control sequences $\mathbf{X}=(\mathbf{x}_1,...,\mathbf{x}_N),\mathbf{U}=(\mathbf{u}_1,...,\mathbf{u}_N)$ are obtained through the composition of two components,

$$\left(\hat{\mathbf{X}},\hat{\mathbf{U}}\right)\sim p_\theta(\mathbf{X},\mathbf{U}|\sigma_0) \tag{3}$$

$$(\mathbf{X},\mathbf{U})=\text{Opt}\left(\mathbf{x}(t_1),\hat{\mathbf{X}},\hat{\mathbf{U}}\right), \tag{4}$$

where $p_\theta(\cdot)$ denotes the conditional probability distribution over trajectories (given an initial condition σ_0) learned by a transformer model with parameters θ, Opt denotes the trajectory optimization problem, and $\hat{\mathbf{X}}$, $\hat{\mathbf{U}}$ denote the predicted (state and control) trajectories that are taken as an input to the optimization problem. The initial condition σ_0 is used to inform trajectory generation in the form of, e.g., an initial state $\mathbf{x}(t_1)$, a target state $\mathbf{x}(t_N)$ to be reached by the end of the trajectory, or other performance-related parameters. In this section, we will first dive into the details of the trajectory representation for sequence modeling. We will then discuss both open-loop pre-training and MPC fine-tuning formulations of our approach, together with specific inference algorithms for open-loop planning and MPC.

A. Trajectory Representation

At the core of our approach is the treatment of trajectory data as a sequence to be modeled by a transformer model [1]. Specifically, given a pre-collected dataset of trajectories of the form $\tau_\text{raw}=(\mathbf{x}_1,\mathbf{u}_1,r_1,...,\mathbf{x}_N,\mathbf{u}_N,r_N)$, where $\mathbf{x}_i$ and $\mathbf{u}_i$ denote the state and control at time t_i, and $r_i=-J(\mathbf{x}(t_i),\mathbf{u}(t_i))$ is the instantaneous reward, or negative cost, we define the following trajectory representation:

$$\tau=(\mathcal{T}_1,\mathcal{R}_1,\mathcal{C}_1,\mathbf{x}_1,\mathbf{u}_1,...,\mathcal{T}_N,\mathcal{R}_N,\mathcal{C}_N,\mathbf{x}_N,\mathbf{u}_N), \tag{5}$$

where $\mathcal{T}_i\in\mathbb{R}^{n_x},\mathcal{R}_i\in\mathbb{R}$ and $\mathcal{C}_i\in\mathbb{N}^+$ represent a set of performance parameters that allow for effective trajectory generation. In particular, we define $\mathcal{T}_i$ as the *target state*, i.e., the state we wish to reach by the end of the trajectory, that could be either constant during the entire trajectory or time-dependent. We further define $\mathcal{R}_i$ and $\mathcal{C}_i$ as the *reward-to-go* and *constraint-violation-budget* evaluated at t_i, respectively. Formally, as in [1], we define $\mathcal{R}_i$ and $\mathcal{C}_i$ to express future optimality and feasibility of the trajectory as:

$$\mathcal{R}(t_i)=\sum_{j=i}^{N} r_j, \qquad \mathcal{C}(t_i)=\sum_{j=i}^{N}\mathsf{C}(t_j), \tag{6}$$

where $\mathsf{C}(t_j)$ checks for constraint violation as

$$\mathsf{C}(t_j)=\begin{cases}1, & \text{if } \exists\mathbf{x}(t_j),\mathbf{u}(t_j)\notin\mathcal{X}_{t_j},\mathcal{U}_{t_j}\\ 0, & \text{otherwise.}\end{cases} \tag{7}$$

This definition of the performance parameters is appealing for two reasons: (i) during training, the performance parameters are easily derivable from raw trajectory data by applying (6) and (7) for $\mathcal{R}_i$ and $\mathcal{C}_i$, and by setting $\mathcal{T}_i=\mathbf{x}_N$, (ii) at inference, according to (3), it allows to condition the generation of predicted state and control trajectories $\hat{\mathbf{X}}$ and $\hat{\mathbf{U}}$ through user-defined initial conditions σ_0. Namely, given user-defined $\sigma_0=(\mathcal{T}_1,\mathcal{R}_1,\mathcal{C}_1,\mathbf{x}_1)$, a successfully trained transformer should be able to generate a trajectory $(\hat{\mathbf{X}},\hat{\mathbf{U}})$ starting from $\mathbf{x}(t_1)$ that achieves a reward of $\mathcal{R}_1$, a constraint violation of $\mathcal{C}_1$, and terminates in state $\mathcal{T}_1$.

B. Open-loop Pre-training

In what follows, we introduce the pre-training strategy used to obtain a transformer model capable of generating near-optimal trajectories $\hat{\mathbf{X}}=(\hat{\mathbf{x}}_1,...,\hat{\mathbf{x}}_N),\hat{\mathbf{U}}=(\hat{\mathbf{u}}_1,...,\hat{\mathbf{u}}_N)$ in an open-loop setting.

Dataset generation. The first step entails generating a dataset for effective transformer training. To do so, we generate N_D trajectories by repeatedly solving Problem (1) with randomized initial conditions and target states, and then re-arranging the raw trajectories according to (5). We construct the datasets to include both solutions to Problem (1) as well as to its relaxations. We observe that diversity in the available trajectories is crucial to enable the transformer to learn the effect of the performance metrics; for example, solutions to relaxations of Problem (1) will typically yield trajectories with low cost and non-zero constraint violation (i.e., high $\mathcal{R}$ and $\mathcal{C}>0$), while direct solutions to Problem (1) will be characterized by higher cost and zero constraint violation (i.e., lower $\mathcal{R}$ and $\mathcal{C}=0$), allowing the transformer to learn a broad range of behaviors from the specification of the performance parameters.

Training. We train the transformer with the standard teacher-forcing procedure used to train sequence models. Specifically, denoting the

L2-norm as $||\cdot||_2$, we optimize the following mean-squared-error (MSE) loss function:

$$\mathcal{L}(\tau) = \sum_{n=1}^{N_D}\sum_{i=1}^{N}\Big(\|\mathbf{x}_i^{(n)} - \hat{\mathbf{x}}_i^{(n)}\|_2^2 + \|\mathbf{u}_i^{(n)} - \hat{\mathbf{u}}_i^{(n)}\|_2^2\Big), \tag{8}$$

where n denotes the n-th trajectory sample from the dataset, $\hat{\mathbf{x}}_i^{(n)} \sim p_\theta(\mathbf{x}_i^{(n)} \mid \tau_{<t_i}^{(n)}), \hat{\mathbf{u}}_i^{(n)} \sim p_\theta(\mathbf{u}_i^{(n)} \mid \tau_{<t_i}^{(n)}, \mathbf{x}_i^{(n)})$ are the one-step predictions for both state and control vectors, and where we use $\tau_{<t_i}$ to denote a trajectory spanning from timesteps $t \in [t_1, t_{i-1}]$.

Inference. Once trained, the transformer can be used to autoregressively generate an open-loop trajectory, i.e., $(\hat{\mathbf{X}}, \hat{\mathbf{U}}) \sim p_\theta(\mathbf{X}, \mathbf{U} \mid \sigma_0)$, from a given initial condition $\sigma_0 = (\mathcal{T}_1, \mathcal{R}_1, \mathcal{C}_1, \mathbf{x}_1)$ that encodes the initial state, target state, and desired performance metrics. Given σ_0, the autoregressive generation process is defined as follows: (i) the transformer generates a control $\mathbf{u}_1$, (ii) as in [1], we take a model-based view on autoregressive generation, whereby the next state $\mathbf{x}_2$ is generated according to a (known) approximate dynamics model $\hat{f}(\mathbf{x}, \mathbf{u})$—which we feel is a reasonable assumption across the problem settings we investigate—, (iii) the performance metrics are updated by decreasing the reward-to-go $\mathcal{R}_1$ and constraint-violation-budget $\mathcal{C}_1$ by the instantaneous reward r_1 and constraint violation $\mathsf{C}(t_1)$, respectively, (iv) the target state $\mathcal{T}_1$ is either kept constant or updated to a new target state, and (v) the previous four steps are repeated until the horizon is reached. To specify the performance parameters, we select $\mathcal{R}_1$ as a (negative) quantifiable lower bound of the optimal cost and $\mathcal{C}_1 = 0$, incentivizing the generation of near-optimal and feasible trajectories.

C. Long-horizon Guidance for Model Predictive Control

The core of our approach relies on leveraging the recursive computation within sequence models for effective model predictive control. Specifically, given a planning horizon H, we fine-tune a transformer to generate trajectories $\hat{\mathbf{X}}_{h:h+H} = (\hat{\mathbf{x}}_h, \hat{\mathbf{x}}_{h+1}, ..., \hat{\mathbf{x}}_{h+H}), \hat{\mathbf{U}}_{h:h+H} = (\hat{\mathbf{u}}_h, \hat{\mathbf{u}}_{h+1}, ..., \hat{\mathbf{u}}_{h+H})$. In our formulation, the generated trajectories are used to provide a warm-start to the trajectory optimization problem, as in the open-loop planning scenario, but also, crucially, to specify an effective terminal cost J_T. As a result, we use the transformer to (i) improve runtime by enabling the solution of smaller optimization problems and (ii) define learned terminal cost terms and avoid expensive cost-tuning strategies.

Closed-loop fine-tuning. Rather than naively applying the transformer trained on open-loop data to the MPC setting, we devise a fine-tuning scheme to achieve effective closed-loop behavior. It is well-known that methods based on imitation learning are exposed to severe error compounding when applied in closed-loop [24]. The main reason behind this error compounding is the covariate shift problem, i.e., the fact that the *actual* performance of the learned policy depends on its own state distribution, whereas *training* performance is only affected by the expert's state distribution (i.e., the policy used to generate the data). To address this problem, we resort to iterative algorithms, namely DAGGER [25], in which the current policy interacts with the training environment to collect trajectory data and define a new dataset for training based on expert corrections. In our framework, we care about the closed-loop performance of MPC policies and define a DAGGER algorithm tailored for trajectory optimization according to three main design choices (Algorithm 1). First, to incentivize the transformer to learn optimal solutions for the full-horizon OCP, we define the expert policy as the open-loop solution to Problem (1). Second, to allow the same transformer model to be used with multiple choices of the planning horizon, we randomize H in the exploration policy. Third, as is usual for DAGGER implementations, we alternate between two exploration policies: the transformer from the last dagger iteration and the expert policy.

Algorithm 1: Fine-tuning strategy for MPC

```
Input: Pre-trained $p_\theta$, Expert $\pi^*$, Open-loop data $\mathcal{D}_{\mathrm{ol}}$
1  Initialize $\mathcal{D}_{\mathrm{ft}} \leftarrow \{\mathcal{D}_{\mathrm{ol}}\}$;
2  for $i = 1$ to dagger iterations do
3      for num trajectories do
4          Select exploration policy $\pi^t_{\mathrm{exp}} = \{p_\theta, \pi^*\}, \forall t$;
5          Sample planning horizon $H \in (0, N]$;
6          Get dataset $\mathcal{D}_i = \{\mathbf{x}, \pi^*(\mathbf{x})\}$
             of states visited by $\pi_{\mathrm{exp}}$ and controls given by $\pi^*$;
7      end
8      Aggregate dataset $\mathcal{D}_{\mathrm{ft}} \leftarrow \mathcal{D}_{\mathrm{ft}} \cup \mathcal{D}_i$;
9      Train model on $\mathcal{D}_{\mathrm{ft}}$;
10 end
```

Inference. After fine-tuning, we use the transformer to generate a trajectory $(\hat{\mathbf{X}}_{h:h+H}, \hat{\mathbf{U}}_{h:h+H}) \sim p_\theta(\mathbf{X}_{h:h+H}, \mathbf{U}_{h:h+H} \mid \tau_{<h})$ following the same five-step process defined in the open-loop scenarioveroooo, whereby the autoregressive generation is executed for H steps, rather than for the full OCP horizon N. We further use the generated trajectory to (i) provide a warm-start for the short-horizon OCP, and (ii) define a cost function of the form:

$$\mathcal{J} = J_\mathrm{T}(\mathbf{x}(t_{h+H}), \hat{\mathbf{x}}(t_{h+H})) + \sum_{i=h}^{h+H} J(\mathbf{x}(t_i), \mathbf{u}(t_i)), \tag{9}$$

where the terminal cost function $J_\mathrm{T}(\mathbf{x}_{h+H}, \hat{\mathbf{x}}_{h+H})$ is defined as a distance metric that penalizes deviation from the predicted terminal state $\hat{\mathbf{x}}_{h+H}$. As a consequence of this formulation, we use the transformer—trained to generate near-optimal trajectories for the *full* OCP—to incentivize solutions that better align with the long-horizon problem.

V. EXPERIMENTS

In this section, we demonstrate the performance of our framework on three trajectory optimization problems: two in simulation (i.e., *spacecraft rendezvous* and *quadrotor control* scenarios) and one real-world robotic platform (i.e., a *free-flyer* testbed). We consider three instantiations of Problem (1) based on the following general formulation:

$$\begin{aligned}
\underset{\mathbf{x}_i, \mathbf{u}_i}{\text{minimize}} \quad & \sum_{i=1}^{N} \|\mathbf{u}_i\|_p^q && && \text{(10a)}\\
\text{subject to} \quad & \mathbf{x}_{i+1} = \mathbf{f}(\mathbf{x}_i, \mathbf{u}_i) && \forall i \in [1, N], && \text{(10b)}\\
& \Gamma(\mathbf{x}_i) \geq \mathbf{0} && \forall i \in [1, N], && \text{(10c)}\\
& \mathbf{x}_1 = \mathbf{x}_{\mathrm{start}}, && && \text{(10d)}\\
& \mathbf{x}_{N+1} = \mathbf{x}_{\mathrm{goal}}, && && \text{(10e)}
\end{aligned}$$

where the objective function in (10a) expresses the minimization of the control effort (with $p, q \in \{1, 2\}$ denoting an application-dependent parameter), the constraint in (10b) represents the (potentially nonlinear) system dynamics, (10c) defines the obstacle avoidance constraints through a nonlinear distance function $\Gamma : \mathbb{R}^{m \times n_x} \rightarrow$

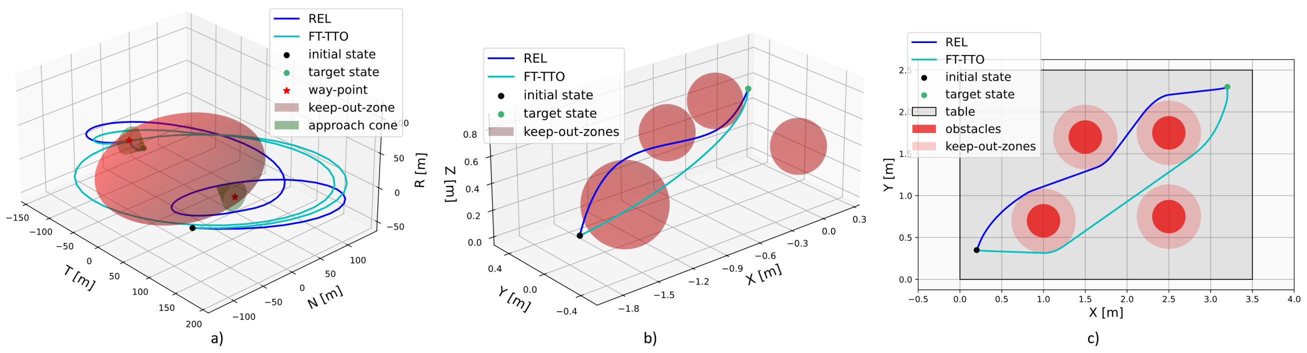

Fig. 2: Visualization of the three scenarios considered in this work. For the tasks of (a) spacecraft rendezvous, (b) quadrotor and (c) free flyer control, we show three example trajectories obtained by warm-starting the SCP through a relaxation of the full OCP (blue) or through our proposed approach (cyan). Keep-out-zones and obstacles are denoted by the red shaded areas.

$\mathbb{R}^m$ with respect to $m \in [1, M]$ non-convex keep-out-zones, and where (10d) and (10e) are the initial and terminal state constraints.

Experimental design. While the specific formulations of the trajectory optimization problem will necessarily depend on the individual application, we design our experimental setup to follow some common desiderata.

First, we care to isolate the benefits of (i) the initial guess (in both open-loop and MPC settings) and (ii) learning the terminal cost for MPC; thus, we keep both the formulation and the solution algorithm for the OCP fixed and only evaluate the effect of different initializations. In particular, we resort to sequential convex programming (SCP) methods for the solution of the trajectory optimization problem and compare different approaches that provide an initial guess for the sequential optimizer (or, for MPC, also a specification of the terminal cost). Second, in the open-loop setting, we always compare our approach, identified as Transformers for Trajectory Optimization (TTO), with the following *classes* of methods: (i) a cost *Lower Bound* (LB) baseline, characterized by the solution of a convex relaxation to the open-loop Problem (10) and thus representing a (potentially infeasible) cost lower bound to the full problem and (ii) an SCP-based approach where the initial guess to Problem (10) is provided by the LB solution described in (i). Third, in the model predictive control setting, we always consider terminal cost terms of the form $J_{\text{T}}(\mathbf{x}_{h+H}, \bar{\mathbf{x}}) = \|\mathbf{x}_{h+H} - \bar{\mathbf{x}}\|_P$, where $\|\cdot\|_P$ denotes the weighted norm with respect to a diagonal matrix P and compare different choices for $\bar{\mathbf{x}}$ and P. Lastly, as introduced in Section IV-B, to obtain diverse trajectories for offline training we solve $\frac{N_d}{2}$ instances of Problem (10) and $\frac{N_d}{2}$ relaxations of the same problem where we ignore obstacle avoidance constraints from (10c).

In our experiments, we care about three main performance metrics: cost, speed of optimization (expressed in terms of the number of SCP iterations), and overall runtime.As in [1], throughout our experiments, we assume knowledge of the obstacle positions and perfect state estimation at all times. We further assume that the target state is reachable within N discrete time instants.

Transformer architecture. Our model is a transformer architecture designed to account for continuous inputs and outputs. Given an input sequence and a pre-defined maximum context length K, our model takes as input the last $5K$ sequence elements, one for each modality: target state, reward-to-go, constraint-violation-budget, state, and control. The sequence elements are projected through a modality-specific linear transformation, obtaining a sequence of $5K$ embeddings. As in [18], we further encode an embedding for each timestep in the sequence and add it to each element embedding. The resulting sequence of embeddings is processed by a causal GPT model consisting of six layers and six self-attention heads. Lastly, the GPT architecture autoregressively generates a sequence of latent embeddings which are projected through modality-specific decoders to the predicted states and controls.

A. *Problem Description*

In this section, we describe our experimental scenarios, the definition of cost, state, and control variables, and the specific implementation of constraints in (10b-10e).

Spacecraft rendezvous and proximity operations. We consider the task of performing an autonomous rendezvous, proximity operation, and docking (RPOD) transfer, in which a servicer spacecraft equipped with impulsive thrusters approaches and docks with a target spacecraft. The relative motion of the servicer with respect to the target is described using the quasi-nonsingular Relative Orbital Elements (ROE) formulation, i.e. $\mathbf{x} := \delta\boldsymbol{\alpha} \in \mathbb{R}^6$, [26], [1]. Cartesian coordinates in the Radial Tangential Normal (RTN) frame can be obtained through a time-varying linear mapping [1] and employed for imposing geometric constraints.

Problem (10) is detailed as follows: (i) in (10a), we select $p=2$ and $q=1$, to account for impulsive thrusters aligned with the desired control input; (ii) in (10d), $\mathbf{x}_{\text{start}}$ is defined by a random passively-safe relative orbit with respect to the target; (iii) in (10e), $\mathbf{x}_{\text{goal}}$ is selected among two possible docking ports located on the T axis, as shown in Fig. 2a; (iv) in (10b), $\delta\boldsymbol{\alpha}_{i+1} = \Phi_i(\Delta t)(\delta\boldsymbol{\alpha}_i + B_i\mathbf{u}_i)$ is used to enforce dynamics constraints, with $\Phi_i(\Delta t) \in \mathbb{R}^{6\times 6}$ and $B_i \in \mathbb{R}^{6\times 3}$ being the linear time-varying state transition and control input matrices [1], Δt the time discretization and $\mathbf{u}_i = \Delta v_i \in \mathbb{R}^3$ the impulsive delta-velocity applied by the servicer; (v) in (10c), we consider an elliptical keep-out-zone (KOZ) centered at the target, limiting the time index i to $[1, N_{\text{wp}}]$. We further consider two additional domain-specific constraints: (vi) a zero relative velocity pre-docking waypoint to be reached at $i = N_{\text{wp}}$; (vii) a second-order cone constraint enforcing the service spacecraft to approach the docking port inside a cone with aperture angle γ_{cone} for $N_{\text{wp}} \le i \le N$, with axis $\mathbf{n}_{\text{cone}} \in \mathbb{R}^3$ and vertex $\delta\mathbf{r}_{\text{cone}} \in \mathbb{R}^3$ determined accordingly to the selected docking port.

Quadrotor control. We consider the task of autonomous quadrotor flight, in which a quadrotor robot has to reach a target state

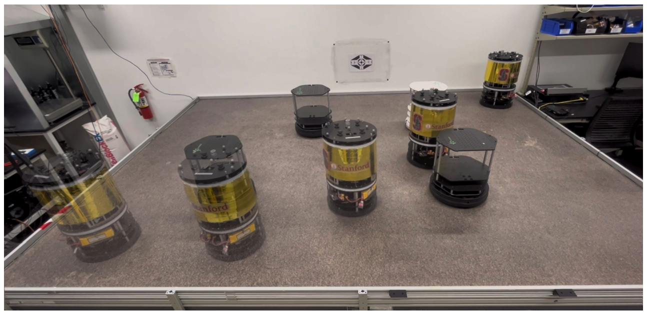

Fig. 3: Free flyer testbed and real-world execution of the FT-TTO trajectory in Fig. 2c. Transparency identifies the time progression from the start (faded) to the end (opaque) of the trajectory.

while navigating an obstacle field, Fig. 2b. We use a Cartesian reference frame $\mathrm{O_{xyz}}$, with the z-axis pointing upward, to define $\mathbf{x} := (\mathbf{r}, \mathbf{v}) \in \mathbb{R}^6, \mathbf{u} := \mathbf{T} \in \mathbb{R}^3$, with $\mathbf{r}, \mathbf{v}, \mathbf{T} \in \mathbb{R}^3$ representing the quadrotor's position, velocity and thrust vectors, respectively.

We define the following elements of Problem (10): (i) in (10a), we select $p = q = 2$ to adhere to the continuous nature of the control input; (ii) in (10d) and (10e), we sample $\mathbf{x}_{\text{start}}$ and $\mathbf{x}_{\text{goal}}$ from pre-determined start and goal regions; (iii) in (10b), system dynamics are given by the nonlinear model $\mathbf{r}_{i+1} = \mathbf{r}_i + \mathbf{v}_i \Delta t$, $\mathbf{v}_{i+1} = \mathbf{v}_i + \frac{1}{m}\left(-\beta_{\text{drag}} \|\mathbf{v}_i\|_2^2 + \mathbf{u}_i\right) \Delta t$, with m and β_{drag} being the mass and the drag coefficient of the drone, respectively; (iv) in (10c), we consider multiple spherical KOZs distributed between the start and goal regions.

Free flyer testbed. Lastly, we consider a real-world free-flyer platform in which a floating robot—the free flyer—can move over a granite table in absence of friction, simulating space flight. We consider the task of controlling the free flyer, equipped with eight ON-OFF thrusters allowing bi-dimensional roto-translational motion, to achieve a target state by traversing an obstacle field. The simulation scenario and the real-world testbed are reported in Fig. 2c and 3, respectively. We use a global Cartesian frame $\mathrm{O_{xy}}$ to define $\mathbf{x} := (\mathbf{r}, \psi, \mathbf{v}, \omega) \in \mathbb{R}^6, \mathbf{u} := R_{\text{GB}}(\psi) \Lambda \Delta \mathbf{V} \in \mathbb{R}^3$, where $\mathbf{r}, \mathbf{v} \in \mathbb{R}^2$ and $\psi, \omega \in \mathbb{R}$ are respectively the position, velocity, heading angle and angular rate of the free flyer, $\Delta \mathbf{V} \in \mathbb{R}^8$ consists of the impulsive delta-velocity applied by each thruster, $\Lambda \in \mathbb{R}^{3 \times 8}$ is the thrusters' configuration matrix and $R_{\text{GB}} \in \mathbb{R}^{3 \times 3}$ is the rotation matrix from the body reference frame of the free flyer to $\mathrm{O_{xy}}$.

Problem (10) is specified as follows: (i) in (10a), we select $p = q = 1$ as the ON-OFF thruster are not aligned with the control input; (ii) in (10d) and (10e), $\mathbf{x}_{\text{start}}$ and $\mathbf{x}_{\text{goal}}$ are sampled from pre-determined start and goal regions; (iii) in (10b), the dynamics are propagated using the impulsive model $\mathbf{x}_{i+1} = \mathbf{x}_i + \text{diag}([\Delta t, \Delta t, \Delta t, 1, 1, 1]) \mathbf{u}_i$; (iv) in (10c), we consider multiple spherical KOZs distributed between the start and goal regions, comprising the obstacles' and the free flyer's radii. We further consider two domain-specific constraints: (v) actuation limits defined by $0 \leq \Lambda^{-1} R_{\text{GB}}^{-1} \mathbf{u} \leq \Delta \mathbf{V}_{\max}$, with $\Delta \mathbf{V}_{\max} = \frac{T \Delta t}{m}$, where T and m denote the thrust level of the thrusters and the mass of the free flyer, respectively; (vi) state bounds, i.e. $\mathbf{x} \in \mathcal{X}_{\text{table}}$, with $\mathcal{X}_{\text{table}}$ defining the table region.

B. *Open-loop Planning*

As a first experiment, we aim to solve the trajectory generation problem in an open-loop setting. As shown in Fig. 2, this results

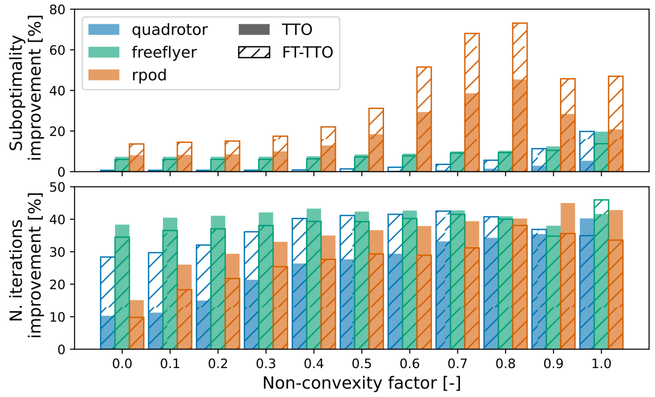

Fig. 4: Percentage improvement in terms of cost suboptimality (top) and number of SCP iterations (bottom) with respect to REL achieved by warm-starting the SCP with FT-TTO and TTO. Each bar represents the improvement averaged over non-convexity factors greater or equal to the corresponding x-axis value.

in a single optimal trajectory that can be robustly tracked by a downstream control system. Specifically, as a form of comparison, we warm-start the SCP using trajectories generated by (i) solving the relaxation of Problem (10) which does not enforce the keep-out-zone constraint (denoted as REL), (ii) the transformer *before* fine-tuning (TTO), and (iii) the transformer *after* fine-tuning (FT-TTO). To initialize the generation process, we set the performance parameter $\mathcal{R}_1$ to a negative quantifiable lower bound (e.g., the cost of the REL solution) and $\mathcal{C}_1 = 0$. We evaluate trajectory generation performance across 40,000 problem specifications which we order according to a *non-convexity factor*, computed as $\frac{\mathcal{C}_1^j}{\mathcal{C}_{\max}}$, where $\mathcal{C}_1^j$ is the number of constraints violations observed in the dataset for each problem specification $j \in [1, 40{,}000]$ and $\mathcal{C}_{\max} = \max_j \mathcal{C}_1^j$ is the maximum constraint violation observed across the entire dataset. In practice, we use the non-convexity factor as a direct metric of the difficulty of the considered scenario, whereby scenarios with a factor of 0 represent OCPs for which there exists a convex solution and hence the solution of REL is optimal for the full problem; whereas scenarios with a factor of 1 represent OCPs for which the closest convex relaxation is highly infeasible and where a better selection of the warm-start can have a larger impact on the solution to the non-convex problem.

Results in Fig. 4 confirm the findings of [1] and show a clear advantage in warm-starting the non-convex problem with TTO, which strongly outperforms initial guesses based on problem relaxations, both in terms of cost and number of SCP iterations. Additionally, Fig. 4 clearly highlights the benefits of the proposed fine-tuning strategy on open-loop performance. In particular, Fig. 4 (top) shows how FT-TTO is able to reduce the sub-optimality gap with respect to the convex lower bound when compared to the model before fine-tuning: FT-TTO achieves approximately a 2x cost improvement compared to TTO in the case of spacecraft RPOD (peak value 75%) and quadrotor control (peak value 20%), while resulting in similar cost performance in the free flyer testbed simulation. Furthermore, in all the scenarios considered, warm-starting with transformers consistently reduces the number of SCP iterations when compared to initial guesses based on REL. Specifically, Fig. 4 (bottom) shows how FT-TTO achieves at least a 10% reduction in SCP iterations across all scenarios and non-convexity levels, achieving up to 40% improvement for high values of the non-convexity factor. Crucially,

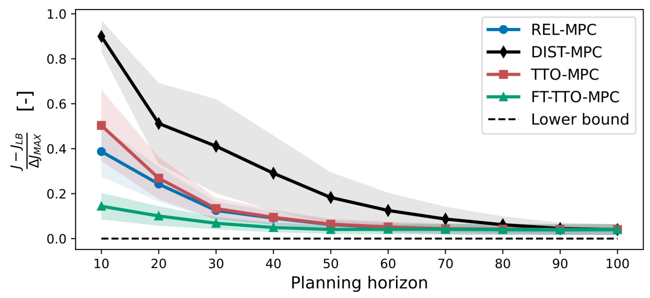

Fig. 5: Average normalized cost increment with respect to the problem lower bound.

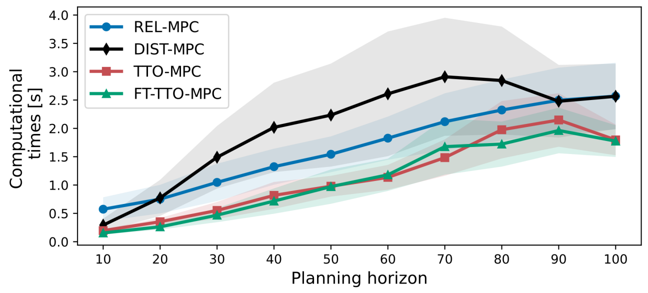

Fig. 6: Average maximum runtime required by a single MPC iteration. Here, the runtime is defined as the total computation time needed to generate the warm-start and solve the SCP.

the substantial cost reduction and the comparable results in SCP iterations demonstrate that, from an open-loop perspective, the fine-tuning process of FT-TTO does not cause any degradation in performance, improving open-loop behavior.

C. Model Predictive Control

The second part of our experiments focuses on solving the trajectory generation problem within a model predictive control scheme. We compare four methods for the generation of the warm-start and the definition of the terminal cost $J_{\mathrm{T}}(\mathbf{x}_{h+H}, \bar{\mathbf{x}}) = \|\mathbf{x}_{h+H} - \bar{\mathbf{x}}\|_2$: (i) REL-MPC solves the full-horizon REL problem and defines $\bar{\mathbf{x}}$ as the state obtained by the REL solution at time t_{h+H}, i.e. $\bar{\mathbf{x}} := \mathbf{x}^{(REL)}_{h+H}$, (ii) DIST-MPC solves the REL problem over the short-horizon H and defines J_{T} to encode the distance from the goal, i.e. $\bar{\mathbf{x}} := \mathbf{x}_{\text{goal}}$, (iii) TTO-MPC and (iv) FT-TTO-MPC employ the transformer models before and after DAGGER fine-tuning, respectively, to generate a short-horizon warm-start and define $\bar{\mathbf{x}} := \hat{\mathbf{x}}_{h+H}$.

For each of the three simulation scenarios considered in this work, we evaluate all methods on 500 problem instances uniformly distributed across the non-convexity factor. Each trajectory is discretized in 100 timesteps and solved according to ten different planning horizons, ranging from 10 to 100 in increments of 10. To facilitate comparison across tasks, we normalize costs to the range between 0 and 1 by computing $\texttt{normalized cost increment} = \frac{\texttt{cost - lowerbound}}{\texttt{maxcost - lowerbound}}$. A normalized cost increment of 0 corresponds to a trajectory that matches the performance of the lower bound. A normalized cost increment of 1 corresponds to a trajectory that matches the worst performance observed across all planning horizons. Aggregate results in Fig. 5 and 6 show how FT-TTO-MPC is able to substantially outperform all other approaches both in terms

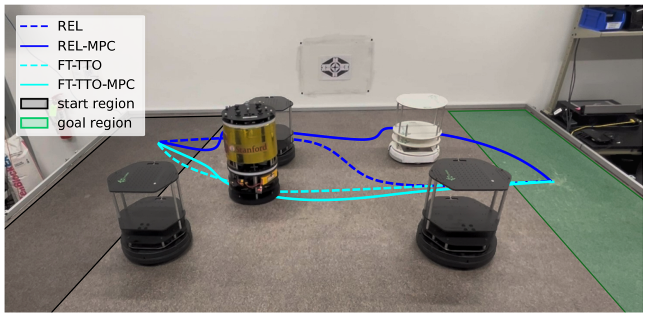

Fig. 7: Qualitative representation of the experimental results obtained employing FT-TTO (cyan) and REL (blue) in the open-loop (dashed) and MPC (solid) settings. Videos available at: https://transformermpc.github.io.

of cost and runtime[1]. In particular, Fig. 5 highlights a few key take-aways. First, due to the short-sighted definition of the terminal cost, DIST-MPC exhibits the biggest performance degradation, showcasing the limits of heuristically defined terminal costs. Second, TTO-MPC (trained solely open-loop data) is highly affected by the covariate shift problem, which severely impacts its closed-loop performance causing it to perform worse than REL-MPC. Lastly, FT-TTO-MPC showcases the importance of fine-tuning transformer models for MPC, resulting in a significative advantage over all the other methods. In particular, the cost induced by FT-TTO-MPC remains stable across different planning horizons, achieving results with $H = 10$ that are on par with other methods with $H = 30$ and allowing for the solution of substantially smaller OCPs.

Results in Fig. 6 further highlight the runtime benefits of FT-TTO-MPC, which outperforms other non-learning-based methods across all planning horizons. Crucially, by evaluating runtime results in the context of their cost performance from Fig. 5, some relevant findings emerge. For example, in the context of REL-MPC, planning with $H = 30$ is necessary to achieve performance comparable to the one obtained by FT-TTO-MPC with $H = 10$. Comparing the corresponding computational times, we see how the long-term guidance introduced by FT-TTO leads to approximately a 7x reduction in runtime for the same performance.

D. Real-world Experiments

Finally, we perform experiments evaluating the real-world effectiveness of our approach through the free flyer testbed described in Section V-A. Specifically, we use the cumulative firing time of the eight thrusters of the free-flyer as a performance metric to compare FT-TTO and REL. In the open-loop setting, we use a PID controller to track the solution obtained by warm-starting the SCP with FT-TTO and REL, respectively. In the MPC framework, we directly control the free flyer using FT-TTO-MPC and REL-MPC with a planning horizon $H = 10$.

Fig. 7 and Tab. I show both qualitative and quantitative results for the free-flyer experiments, respectively. In the open-loop setting (O-L), FT-TTO facilitates the convergence to a solution with a reduced number of sharp turns (Fig. 7)—and thus, propellant consumption (Tab. I)—clearly outperforming REL. In the MPC framework, FT-TTO-MPC leads to a closed-loop execution that closely mimics the open-loop solution, limiting the increase in propellant consumption

[1]Computation times are from a Linux system equipped with a 4.20GHz processor and 128GB RAM, and a NVIDIA RTX 4090 24GB GPU.

TABLE I: Quantitative results of experimental tests shown in Fig. 7.

Method		Target acquisition time $[s]$	Frequency $[Hz]$	Total firing time $[s]$
O-L	REL	40.0	-	99.70
	FT-TTO	40.0	-	**75.78**
MPC	REL-MPC	80.0	1.25	83.76
	FT-TTO-MPC	**40.0**	**2.5**	91.34

to approximately 20%, i.e., a value in line with the numerical analysis in simulation. On the other hand, REL-MPC results in a terminal cost that aggressively steers the free-flyer toward the target, leading to significant deviations from the open-loop solution and abrupt turns in the proximity of the obstacles. Furthermore, in our experiments, it was necessary to double the target acquisition time and halve the control frequency to have REL-MPC successfully reach the target while accounting for higher computation times and preventing collisions—a scenario we encountered consistently during experimentation with REL-MPC. As a result, the increased target acquisition times lead to an overall lower firing time, as shown in Tab. I. Crucially, FT-TTO-MPC was the only MPC algorithm able to be deployed within real-world scenarios that did not require loosened time constraints.

VI. Conclusions

Despite its potential, the application of learning-based methods for the purpose of trajectory optimization has so far been limited by the lack of safety guarantees and characterization of out-of-distribution behavior. At the same time, methods relying on numerical optimization typically lead to high computational complexity and strong dependency on initialization. In this work, we address these shortcomings by proposing a framework to leverage the flexibility of high-capacity neural network models while providing the safety guarantees and computational efficiency needed for real-world operations. We do so by defining a structure where a transformer model—trained to generate near-optimal trajectories—is used to provide an initial guess and a predicted target state as long-horizon guidance within MPC schemes. This decomposition, coupled with a pre-training-plus-fine-tuning learning strategy, results in substantially improved performance, runtime, and convergence rates for both open-loop planning and model predictive control formulations.

In future work, we plan to extend our formulation to handle multi-task, reconfigurable, and stochastic trajectory optimization problems through tailored learning strategies, structured treatment of uncertainty, and scene representations. Moreover, while we have investigated the impact of fine-tuning based on (iterative) supervised learning in this work, the exploration of other fine-tuning strategies (e.g., based on reinforcement learning) is a highly promising direction that warrants future work. More generally, we believe this research opens several promising directions to leverage the power of learning-based methods for trajectory optimization within safety-critical robotic applications.

Acknowledgments

This work is supported by Blue Origin (SPO #299266) as Associate Member and Co-Founder of the Stanford's Center of AEroSpace Autonomy Research (CAESAR) and the NASA University Leadership Initiative (grant #80NSSC20M0163). This article solely reflects the opinions and conclusions of its authors and not any Blue Origin or NASA entity.

References

[1] T. Guffanti, D. Gammelli, S. D'Amico, and M. Pavone, "Transformers for trajectory optimization with application to spacecraft rendezvous," in *IEEE Aerospace Conference*, 2024.

[2] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus, "Advances in trajectory optimization for space vehicle control," *Annual Reviews in Control*, vol. 52, 2021.

[3] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proc. IEEE Conf. on Robotics and Automation*, 2011.

[4] M. Ruzzene, R. Kamada, C. Bottasso, and F. Scorcelletti, "Trajectory optimization strategies for supercavitating underwater vehicles," *Journal of Vibration and Control*, vol. 14, 2008.

[5] M. Mohanan and A. Salgoankar, "A survey of robotic motion planning in dynamic environments," *Robotics and Autonomous Systems*, vol. 100, 2018.

[6] F. Oldewurtel, A. Parisio, C. N. Jones, D. Gyalistras, M. Gwerder, V. Stauch, B. Lehmann, and M. Morari, "Use of model predictive control and weather forecasts for energy efficient building climate control," *Energy and Buildings*, vol. 45, 2012.

[7] M. Morari and J. H. Lee, "Model predictive control: past, present and future," *Computers & Chemical Engineering*, vol. 23, 1999.

[8] J. T. Betts, "Survey of numerical methods for trajectory optimization," *AIAA Journal of Guidance, Control, and Dynamics*, vol. 21, 1998.

[9] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge Univ. Press, 2009.

[10] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1–40, 2016.

[11] C. Wu, A. Rajeswaran, Y. Duan, V. Kumar, A. Bayen, S. Kakade, I. Mordatch, and P. Abbeel, "Variance reduction for policy gradient with action-dependent factorized baselines," in *Int. Conf. on Learning Representations*, 2018.

[12] J. Zhang, C. Ni, Z. Yu, C. Szepesvari, and M. Wang, "On the convergence and sample efficiency of variance-reduced policy gradient method," in *Conf. on Neural Information Processing Systems*, 2021.

[13] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *Proc. IEEE Conf. on Robotics and Automation*, 2018.

[14] S. Bansal, V. Tolani, S. Gupta, J. Malik, and C. Tomlin, "Combining optimal control and learning for visual navigation in novel environments," in *Conf. on Robot Learning*, 2020.

[15] F. Xia, C. Li, R. Martín-Martín, O. Litany, A. Toshev, and S. Savarese, "Relmogen: Integrating motion generation in reinforcement learning for mobile manipulation," in *Proc. IEEE Conf. on Robotics and Automation*, 2021.

[16] T. Lew, S. Singh, M. Prats, J. Bingham, J. Weisz *et al.*, "Robotic table wiping via reinforcement learning and whole-body trajectory optimization," in *Proc. IEEE Conf. on Robotics and Automation*, 2023.

[17] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling," in *Conf. on Neural Information Processing Systems*, 2021.

[18] M. Janner, Q. Li, and S. Levine, "Offline reinforcement learning as one big sequence modeling problem," in *Conf. on Neural Information Processing Systems*, 2021.

[19] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song, "Diffusion policy: Visuomotor policy learning via action diffusion," in *Robotics: Science and Systems*, 2023.

[20] S. Banerjee, T. Lew, R. Bonalli, A. Alfaadhel, I. A. Alomar, H. M. Shageer, and M. Pavone, "Learning-based warm-starting for fast sequential convex programming and trajectory optimization," in *IEEE Aerospace Conference*, 2020.

[21] A. Cauligi, P. Culbertson, B. Stellato, D. Bertsimas, M. Schwager, and M. Pavone, "Learning mixed-integer convex optimization strategies for robot planning and control," in *Proc. IEEE Conf. on Decision and Control*, 2020.

[22] B. Amos, I. D. J. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable MPC for end-to-end planning and control," in *Conf. on Neural Information Processing Systems*, 2018.

[23] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter, "Differentiable convex optimization layers," in *Conf. on Neural Information Processing Systems*, 2019.

[24] N. Rajaraman, L. F. Yang, J. Jiao, and K. Ramchandran, "Toward the fundamental limits of imitation learning," in *Conf. on Neural Information Processing Systems*, 2020.

[25] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proc. Int. Conf. on Artificial Intelligence and Statistics*, 2011.

[26] A. W. Koenig, T. Guffanti, and S. D'Amico, "New state transition matrices for spacecraft relative motion in perturbed orbits," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 7, pp. 1749–1768, 2017.