Neural-Primitive: An Efficient End-to-end Local Planner with Primitive-based Imitation Learning for Autonomous Flight

Zhitao Liu[†,1,2,3], Guangtong Xu[†,4], Zihan Wang[5], Jialiang Hou[1,6], Chao Xu[1,2], and Fei Gao[1,6]

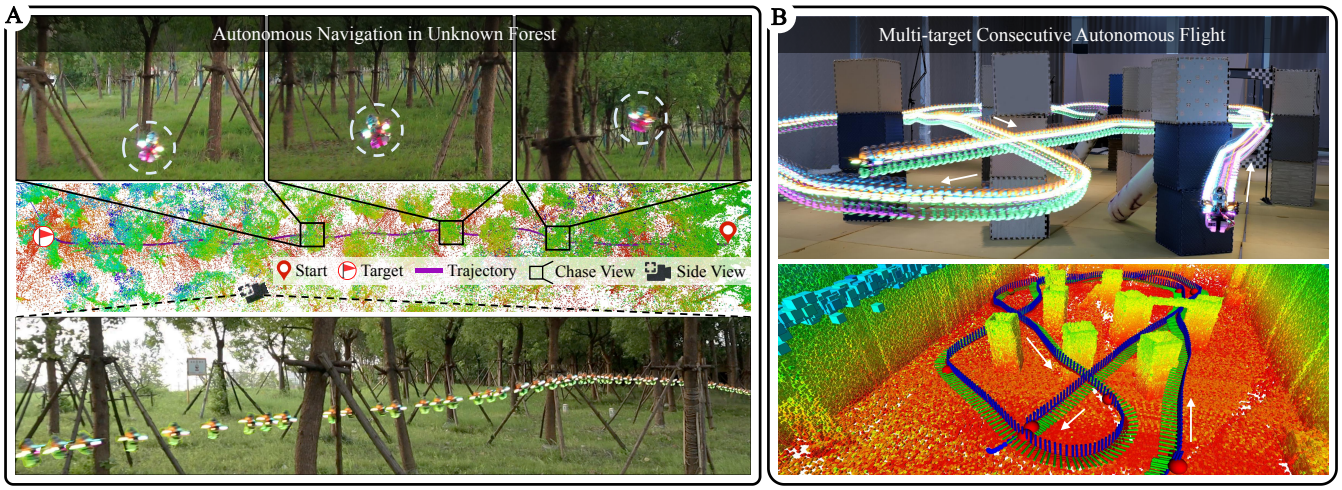

Fig. 1. **Real-world experiments with zero-shot network deployment. (A)** The quadrotor autonomously navigates through a dense cluttered forest and reaches the target 60m ahead with top-level performance. **(B)** Trajectory of multi-target consecutive autonomous flight in confined indoor space with random obstacles.

***Abstract*— Autonomous flight in unknown cluttered environments is hindered by the *computation-quality-memory* trilemma of onboard trajectory generation. In this paper, we propose an efficient end-to-end local planner via imitation learning. A lightweight offline-primitive-based dataset collection framework is designed to produce safe and high-quality trajectory primitives in non-convex environments. A compact neural network directly maps sensory inputs to polynomial coefficients that inherently encode higher-order dynamical information. The learned policy generates smooth, empirically collision-free and dynamically feasible trajectories in real time without back-end solving. It achieves ultra-fast computation (below 1ms on a standard desktop and average 3.68ms during onboard flight), while maintaining low onboard memory requirements (less than 1.5MiB). Extensive simulation benchmarks demonstrate superiority in both planning latency and target-reaching progress quality. Zero-shot deployment in real-world experiments further validates the robust sim-to-real transfer capability of the proposed method.**

[†]Equal Contribution.

[1]Institute of Cyber-Systems and Control, College of Control Science and Engineering, Zhejiang University, Hangzhou 310027, China.

[2]Huzhou Institute, Zhejiang University, Huzhou 313000, China.

[3]Institute of Systems Engineering, China Academy of Engineering Physics, Mianyang 621999, China.

[4]School of Automation, Hangzhou Dianzi University, Hangzhou 310018, China.

[5]Department of Automation, North China Electric Power University (Baoding), Baoding 071003, China.

[6]Differential Robotics Technology Company, Hangzhou 311100, China.

E-mail: zhitaoliu@zju.edu.cn; xugt@hdu.edu.cn; 220232216066@ncepu.edu.cn; jlhou25@zju.edu.cn; cxu@zju.edu.cn; fgaoaaa@zju.edu.cn.

Our project page with videos is at `https://ZhitaoLiu.github.io/neural-primitive/`.

I. Introduction

Autonomous flight in unknown cluttered environments using only onboard noisy sensing and resource-constrained computation, epitomizes one of the grand challenges for unmanned aerial vehicles (UAVs) [1]. Despite remarkable progress [2]–[4], fast generation of dynamically feasible, collision-free, and efficiently target-reaching trajectories still remains open.

Numerous trajectory generation approaches [5]–[8] couple obstacle avoidance into joint optimization through strict constraints or weighted penalties. However, simultaneously considering excessive factors forces trade-offs, making solutions prone to local minima or even intractable, degrading overall trajectory quality. In contrast, motion primitive-based methods [9]–[12] decouple obstacle avoidance out through a stepwise pipeline of free-space sampling, collision removal, and optimal selection. This facilitates near-optimal solutions but renders performance hinging on the primitive library: online-generated methods struggle to produce both dynamically feasible and diverse primitives in a short time; whereas offline-generated methods can pre-compute high-quality ones without latency concerns, yet suffer from higher-order state discontinuities during online concatenation, as the fixed library cannot cover all required initial states with limited onboard memory. Furthermore, these approaches remain subject to processing latency and compounding errors due to hierarchical planning frameworks. Recently, learning-based planning has gained significant attention [13]–[15]. Although claiming end-to-end planning capability, most

still require online back-end optimization [16], closed-form solving [14], or projection [2] after network inference to yield controller-executable trajectories, ultimately restricting achievable planning efficiency. These methods invariably face a *computation-quality-memory* trilemma, namely the inability to simultaneously satisfy fast online computation, near-optimal target-reaching trajectories, and low onboard memory consumption.

To address the aforementioned challenges, we propose *Neural-Primitive*, an efficient end-to-end local planner realized by imitation learning of a customized offline primitive strategy. Unlike naive replication of existing planners, which is limited by expert performance, we inherit the core idea from offline primitive library-based methods, namely generating high-quality primitives through sampling-exploration and obstacle-avoidance-decoupling. At the same time, we resolve their major drawbacks (high-order discontinuities and storage burden of large libraries) by employing online inference with a compact neural network (memory size less than 1.5MB). We design a primitive-strategy-embedded dataset collection procedure that mitigates covariate shift [17] without resorting to resource-intensive approaches such as DAgger [18]. This enables subsequent training to capture precise target-reaching behaviors in cluttered environments. Compared with existing learning-based planners, our policy directly predicts polynomial coefficients from sensory inputs, intrinsically encoding high-order dynamical information and producing practically controller-executable trajectories without back-end solving[1]. This design results in a highly integrated end-to-end planning scheme. Together with a streamlined network architecture that avoids convolutions and other costly operations, the proposed method achieves ultra-fast computation ($<$ 1ms on standard desktop and mean 3.68ms onboard) while maintaining consistency across scenarios. To bridge the sim-to-real gap from sensory mismatch, we design a point cloud preprocessing technique that adapts to obstacle density variations. This boosts robustness and generalization atop domain randomization. Consequently, our method achieves fast online generation of near-straight trajectories toward the target across unknown cluttered environments.

We benchmark with representative state-of-the-art local planners [3, 5, 8, 14] through extensive simulations. Statistical results demonstrate superior performance in computation time, success rate, flight length, and flight time. The generalization and sim-to-real capabilities of the proposed method are further validated in complex previously unseen maps and real-world experiments. The contributions of this paper are summarized as follows:

1) A lightweight offline-primitive-based dataset collection framework, which efficiently produces safe and high-quality trajectory primitives in non-convex environments.
2) A compact imitation learning neural network, which can output polynomial coefficients empirically satisfying safety, dynamical feasibility, and target-reaching progress quality under the supervision of the expert dataset, with enhanced generalization capability enabled by the proposed point cloud pre-processing training technique.
3) An ultra-fast and high-quality end-to-end local planner, which directly maps sensory inputs to high-order continuous polynomial trajectories without back-end solving.
4) Comprehensive simulation benchmarks and zero-shot deployment in real-world experiments are conducted to validate the superiority of our method.

[1]"Without back-end solving" means that no additional optimization, quadratic programming, or boundary-value problem solving is needed after network inference.

II. Related Work

A. Primitive-based Motion Planning for UAV

Primitive-based methods cast motion planning as primitive selection, enhancing solution space multimodality through free-space sampling and obstacle-avoidance decoupling. Based on primitive library generation, they can be categorized into online and offline approaches. In [10, 19, 20], polynomial trajectory candidates are obtained online by sampling terminal states and solving boundary value problems (BVPs) in closed form. However, dynamical feasibility is not considered during generation and must be verified individually through post-checking [21]. Some works [11, 22, 23] sample control inputs and generate primitives online through forward calculation with simplified motion models, which can accommodate input limits but degrade trajectory quality due to model inaccuracy. Online methods can hardly generate trajectory primitives with sufficient diversity and quality within a short planning horizon. Conversely, studies [12, 24] generate plenty of path primitives offline for online selection, but geometric paths neglect dynamical properties (velocity, acceleration, etc.), thus fail to ensure higher-order continuity during online concatenation. Hou et al. [9] further generate time-optimal trajectory primitives offline with dense initial velocity discretizations to promote velocity continuity, but acceleration continuity still remains unresolved. In practice, although offline methods offer broad primitive coverage regardless of latency concerns, storing all high-order initial states for online concatenation without exceeding onboard memory is nearly infeasible (a rough estimate based on [9] indicates that covering initial states up to 3m/s and 3m/s^2 with 0.1 steps would require far more than 1000GB of memory), leaving discontinuity an inherent unresolved drawback.

B. Learning-based Motion Planning for UAV

Neural networks, with their strong capacity for environmental abstraction and rapid online inference, show promising potential in enhancing UAV motion planning efficiency. Wu et al. [15] use a network to predict time allocations for piecewise trajectories, but still rely on the traditional hierarchical pipeline of mapping, path finding, and trajectory optimization, leading to high computational cost. In [13, 16], networks replace mapping and path finding, but online optimization is still required for generating executable

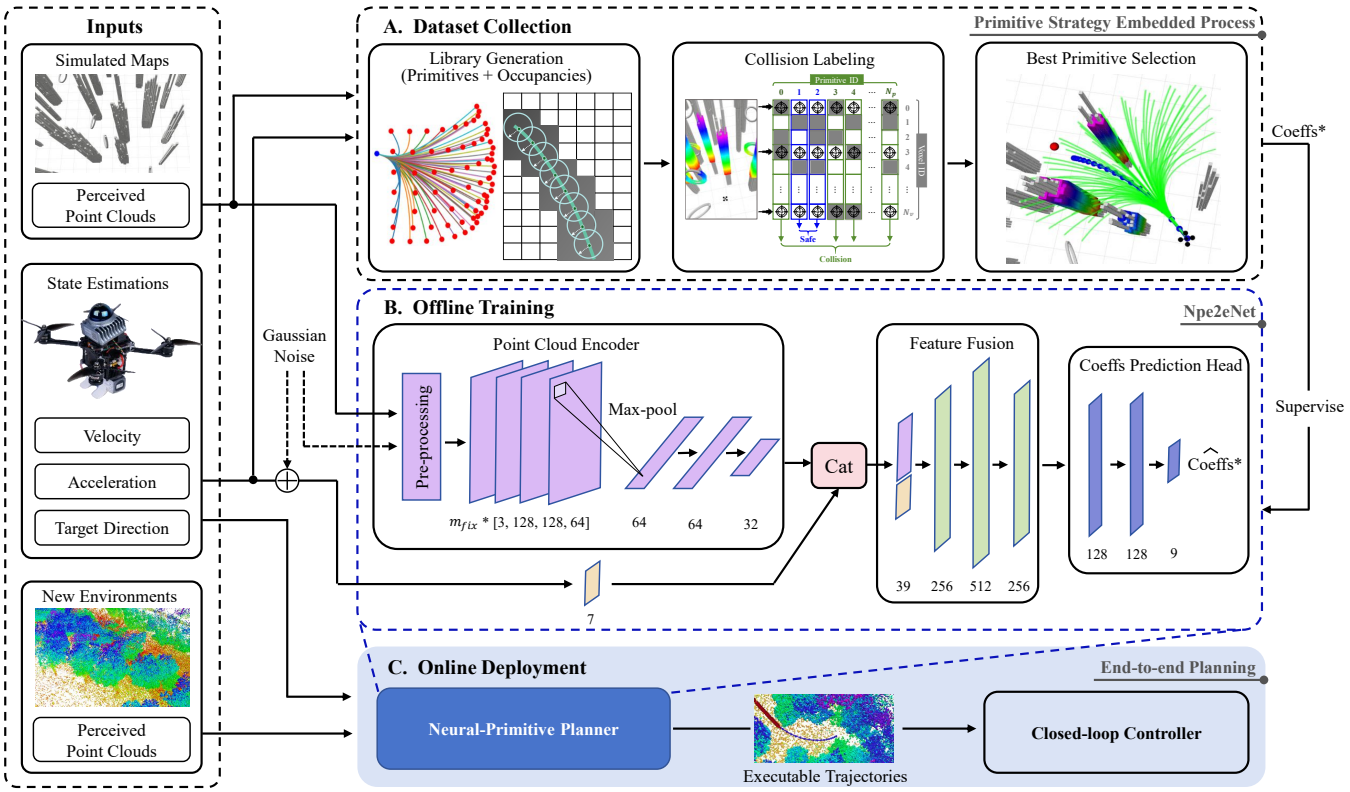

Fig. 2. **System overview.** The end-to-end planner generates smooth, empirically collision-free and dynamically feasible trajectories directly from onboard sensory inputs. **(A)** Training datasets are collected entirely in simulation using a customized primitive strategy. **(B)** A neural network learns to predict polynomial coefficients that inherently encode high-order dynamical information for direct execution by the low-level controller. **(C)** The learned policy is zero-shot deployed for fast online inference in previously unseen environments without fine-tuning.

trajectories. The network in [14] predicts end-state terms for pre-sampled primitives and solves one online via a closed-form BVP back-end. However, dynamical process constraints are not explicitly enforced, and its coupled training objective introduces inherent trade-offs, making solutions prone to local minima. Loquercio et al. [2] utilize a network to output waypoints, but they contain only geometric information without dynamical properties, requiring real-time projection after network inference to obtain polynomial trajectories. Overall, existing learning-based planning methods have yet to fully demonstrate end-to-end planning without back-end trajectory solving, and direct generation of controller-executable trajectories with efficient target-reaching progress quality still remains improvable.

III. System Overview

Fig. 2 illustrates the three-stage pipeline of the proposed system.

Firstly, all training datasets are collected in simulation, incorporating a customized primitive strategy to strengthen the dynamical feasibility, collision avoidance, and task-oriented trajectory quality of the expert primitives. The resulting primitive from each successful replanning step is stored along with the perceived point clouds, drone states, and target direction. Instead of saving path points that only represent geometric positions [2], the polynomial coefficients of the parameterized primitive are recorded, which inherently encode high-order dynamical information.

Subsequently, a policy network based on multilayer perceptrons (MLPs) is trained offline in a supervised manner utilizing the pre-collected datasets. It learns to generate continuous primitives from discrete samples, predicting polynomial coefficients from sensory inputs. This addresses the velocity or acceleration discontinuity issue inherent in traditional offline primitive library-based planning methods [9, 12, 24], while also eliminating the need for back-end trajectory solving [13]–[16, 25] or projection [2] required in existing learning-based planning approaches, thereby enabling an end-to-end planning framework. To reduce the sensory gap for sim-to-real transfer, point clouds are preprocessed prior to formal training and augmented with Gaussian noise, which is also injected into other inputs to enhance generalization.

Finally, the learned policy is zero-shot deployed onboard for inference in previously unseen environments without any fine-tuning. It directly maps onboard sensory data to practically executable trajectories in a receding-horizon fashion, which are sent to the low-level controller for execution.

IV. Methodology

A. Primitive Strategy Embedded Dataset Collection

1) Collection Procedure: Dataset generation for imitation learning needs to account for the covariate shift [17] arising from distribution mismatch between expert demonstrations and real-world deployment. Rather than relying on resource-intensive methods like DAgger [18], we collect the full dataset prior to training. Specifically, data are obtained in a

Algorithm 1: Dataset Collection with Primitive Strategy

Input: n_{max}, n_{thrd}, Θ_{map}, Θ_{init}, Θ_{lib}
Output: $\mathcal{D}$

1 $\mathcal{D} \leftarrow \emptyset$, $n_{valid} \leftarrow 0$
2 **while** $n_{valid} < n_{max}$ **do**
3 $\quad \mathcal{M} \leftarrow randomMapGen(\Theta_{map})$
4 $\quad$ **forall** $thrd_i \in \{1, \dots, n_{thrd}\}$ **do in parallel**
5 $\quad\quad \boldsymbol{s}_0, \boldsymbol{p}_{target}, \boldsymbol{d}_0 \leftarrow randomInitGen(\Theta_{init})$
6 $\quad\quad \mathcal{D}_{temp}.clear()$
7 $\quad\quad$ **while** true **do**
8 $\quad\quad\quad \mathcal{L}.clear()$
9 $\quad\quad\quad$ **forall** $\boldsymbol{s}_f \in endStateSet(\Theta_{lib})$ **do**
10 $\quad\quad\quad\quad \boldsymbol{\xi} \leftarrow solveQP(\boldsymbol{s}_0, \boldsymbol{s}_f, \tau, \Theta_{lib})$
11 $\quad\quad\quad\quad \mathcal{O} \leftarrow sampleTraversalVoxels(\boldsymbol{\xi})$
12 $\quad\quad\quad\quad \mathcal{L}.pushback(prim(\boldsymbol{\xi}, \mathcal{O}))$
13 $\quad\quad\quad \mathcal{P} \leftarrow sensePointCloud(\boldsymbol{s}_0, \mathcal{M})$
14 $\quad\quad\quad \mathcal{L}_{safe} \leftarrow collisionLabeling(\mathcal{L}, \mathcal{P}, R_W^V)$
15 $\quad\quad\quad prim^* \leftarrow selectBest(\mathcal{L}_{safe}, \boldsymbol{p}_{target}, R_W^V)$
16 $\quad\quad\quad$ **if** $prim^*$ is empty **then**
17 $\quad\quad\quad\quad$ **break**
18 $\quad\quad\quad \boldsymbol{c}^* \leftarrow prim^*.getCoeffs()$
19 $\quad\quad\quad \boldsymbol{v}_0, \boldsymbol{a}_0 \leftarrow \boldsymbol{s}_0.getVelAcc()$
20 $\quad\quad\quad \mathcal{D}_{temp}.pushback(data(\boldsymbol{v}_0, \boldsymbol{a}_0, \boldsymbol{d}_0, \mathcal{P}, \boldsymbol{c}^*))$
21 $\quad\quad\quad \boldsymbol{s}_0, \boldsymbol{d}_0 \leftarrow executeTraj(prim^*)$
22 $\quad\quad\quad$ **if** $targetReach(\boldsymbol{s}_0, \boldsymbol{p}_{target})$ **then**
23 $\quad\quad\quad\quad \mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_{temp}$
24 $\quad\quad\quad\quad n_{valid} \leftarrow n_{valid} + \mathcal{D}_{temp}.size()$
25 $\quad\quad\quad\quad$ **break**
26 $\quad\quad$ **if** $n_{valid} \geq n_{max}$ **then**
27 $\quad\quad\quad$ terminate all threads
28 **return** $\mathcal{D}$

closed-loop manner, recording only those from the successful completion of randomly initialized navigation tasks. Evidently, replanning can be triggered at arbitrary initial states throughout the process. However, constructing a large offline library covering all the states for selection is infeasible due to overwhelming memory and computation burden. Therefore, we propose a dataset collection procedure embedded with a customized primitive strategy, as detailed in Algorithm 1.

A randomized simulation map $\mathcal{M}$ is first generated from given parameters Θ_{map} (Line 3). Then up to n_{thrd} navigation tasks run in multithreaded parallel to accelerate the process. Each task randomly initializes the target position $\boldsymbol{p}_{target}$ and drone state $\boldsymbol{s}_0$ based on boundary values defined in parameters Θ_{init}, spanning diverse initial drone-target configurations, with $\boldsymbol{d}_0$ denoting the normalized target direction vector. A trajectory library with occupancy relations, conditioned on the current state, is regenerated at the beginning of each replanning step (Lines 8-12, detailed in Section IV-A.2), eliminating the need for a large-scale precomputed library. Unsafe primitives are rapidly identified and excluded based on perceived point clouds and preassigned occupancy relations, then the safest one is chosen by a composite cost function (Lines 13-15, detailed in Section IV-A.3). If a valid primitive is found, the corresponding data, containing velocity $\boldsymbol{v}_0$, acceleration $\boldsymbol{a}_0$, direction $\boldsymbol{d}_0$, perceived point clouds $\mathcal{P}$, and polynomial coefficients $\boldsymbol{c}^*$ of the selected primitive, are stored in a temporary buffer $\mathcal{D}_{temp}$. The primitive is then partially executed to update the drone state. Only when the drone reaches the target within a predefined tolerance is $\mathcal{D}_{temp}$ appended to the final dataset $\mathcal{D}$. This helps encode drone-target proximity and enables the network to learn precise reaching behavior from the dataset. The process repeats with new maps until the valid data number n_{valid} reaches n_{max}, after which the full dataset $\mathcal{D}$ is returned.

2) Trajectory Library with Occupancy Relations: The library $\mathcal{L}$ is constructed via state lattice discretization, with state dimensions extending to acceleration. Since the initial state $\boldsymbol{s}_0$ is already known at the replanning step, only terminal states $\boldsymbol{s}_f$ need to be sampled (Line 9), which significantly reduces the library size. The offline process further enables multi-constraint optimization to generate a set of candidate primitives that are both dynamically feasible and spatially diverse without considering latency.

The trajectory primitive $\boldsymbol{\xi}(t) \in \mathbb{R}^3$ is parameterized as a single-segment polynomial vector function in three-dimensional space:

$$\boldsymbol{\xi}(t) = [x(t), y(t), z(t)]^T \in \mathbb{R}^3 = \boldsymbol{c}^T \boldsymbol{\beta}(t), \tag{1}$$

where $\boldsymbol{c} = [\boldsymbol{c}_x \ \boldsymbol{c}_y \ \boldsymbol{c}_z] \in \mathbb{R}^{(n+1)\times 3}$,

$$\boldsymbol{c}_\mu = [c_{n\mu}, \dots, c_{2\mu}, c_{1\mu}, c_{0\mu}]^T \in \mathbb{R}^{n+1},\ \mu \in \{x, y, z\},$$

$$\boldsymbol{\beta}(t) = [t^n, \dots, t^2, t, 1]^T \in \mathbb{R}^{n+1},\ t \in [0, \tau].$$

here $x(t)$, $y(t)$ and $z(t)$ are the drone positions; n is the polynomial order; $\boldsymbol{c} \in \mathbb{R}^{(n+1)\times 3}$ is the coefficient matrix; and τ is the fixed time duration.

We solve for the primitive coefficients with minimum control effort. Let $\bar{\boldsymbol{c}} = \text{vec}(\boldsymbol{c}) \in \mathbb{R}^{3(n+1)}$ denote the column-wise vectorization of $\boldsymbol{c}$. Owing to the differential flatness of quadrotors [26], the optimization problem can be formulated as the convex quadratic program (QP) below:

$$\min_{\bar{\boldsymbol{c}}} \quad \bar{\boldsymbol{c}}^T \boldsymbol{Q} \bar{\boldsymbol{c}}, \tag{2}$$

$$\text{s.t.} \quad \boldsymbol{A}\bar{\boldsymbol{c}} = \boldsymbol{b}, \tag{3}$$

$$\boldsymbol{G}\bar{\boldsymbol{c}} \leq \boldsymbol{h}. \tag{4}$$

where $\boldsymbol{Q} \in \mathbb{R}^{3(n+1)\times 3(n+1)}$ is the positive semidefinite quadratic cost matrix; (3) & (4) represent the boundary and dynamical constraints, respectively, with limits specified in parameters Θ_{lib}. Specifically, $\boldsymbol{A} \in \mathbb{R}^{6N_b \times 3(n+1)}$ and $\boldsymbol{b} \in \mathbb{R}^{6N_b}$ are the boundary constraint matrix and vector, with N_b denoting the number of derivative orders enforced at boundary points, satisfying $n = 2N_b - 1$; $\boldsymbol{G} \in \mathbb{R}^{6(N_b-1)N_s \times 3(n+1)}$ and $\boldsymbol{h} \in \mathbb{R}^{6(N_b-1)N_s}$ are the dynamical constraint matrix and vector, with N_s denoting the number of uniformly sampled time points for dynamical limit checking. This QP (Line 10) is solved employing OSQP solver [27]. In this work, we consider minimum-jerk trajectories with $n = 5$, $N_b = 3$ (constraining position, velocity, and acceleration), $N_s = 20$,

and $\tau = 2$s. The boundary constraints on start states are taken from the current drone state at each replanning step, while the terminal ones are sampled from the end-state lattice.

The primitives and all other stored data are expressed in a velocity-aligned frame $\mathcal{F}_V$, where the x-axis is aligned with the drone's velocity vector, the z-axis opposes the gravity vector $\boldsymbol{g}$, and the y-axis completes the right-handed coordinate system. All primitives share the same origin at zero position in this frame. Inspired by [9], we additionally construct occupancy relations in this stage to enable subsequent fast collision checking. In $\mathcal{F}_V$, a bounding box enclosing all primitives is discretized into small voxels and organized as a *kd*-tree. For each primitive, the indices of all traversed voxels within an inflated collision radius are retrieved and stored in a *hash*-based set, which defines its occupancy relations $\mathcal{O}$ (Line 11), representing the spatial region occupied by the inflated primitive. Binding these relations directly to the corresponding primitive also helps form an implicit primitive-obstacle relationship, benefiting subsequent network learning. The complete trajectory library $\mathcal{L}$ comprises all optimized primitives along with their associated occupancy relations (Line 12).

3) Collision Labeling and Best Primitive Selection: The perceived point clouds are downsampled to a fixed size and mapped into the bounding box to obtain the corresponding voxel indices (Line 13), where R_W^V denotes the rotation matrix from the world frame $\mathcal{F}_W$ to frame $\mathcal{F}_V$. A primitive is labeled unsafe and removed from the library if any of these indices are contained in its occupancy relations (Line 14). This process is highly efficient, as voxel inclusion is checked via *hash* lookup, and a primitive is marked unsafe once the first matching voxel is found. Moreover, it eliminates frequent Euclidean Signed Distance Field (ESDF) queries required by existing learning-based methods during data collection or training [13, 14, 16, 25], making it resource-friendly as well.

The best primitive is selected from the remaining safe candidates via a task-oriented cost function (Line 15):

$$\mathcal{C} = w_t \mathcal{C}_{target} + w_l \mathcal{C}_{length}. \tag{5}$$

where w_t and w_l are the weights; $\mathcal{C}_{target}$ and $\mathcal{C}_{length}$ are the target-approach cost and primitive-length cost, respectively, calculated by:

$$\mathcal{C}_{target} = \begin{cases} \|\boldsymbol{\xi}(\tau) - \boldsymbol{p}_{target}^{\mathcal{F}_V}\|, & \text{if } \|\boldsymbol{p}_{target}^{\mathcal{F}_V}\| > \|\boldsymbol{\xi}(\tau)\|, \\ \min\limits_{i=0,\dots,N} \|\boldsymbol{\xi}(t_i) - \boldsymbol{p}_{target}^{\mathcal{F}_V}\|, & \text{otherwise.} \end{cases} \tag{6}$$

$$\mathcal{C}_{length} = \sum_{i=0}^{N-1} \|\boldsymbol{\xi}(t_{i+1}) - \boldsymbol{\xi}(t_i)\|. \tag{7}$$

where $\boldsymbol{p}_{target}^{\mathcal{F}_V}$ is the target position vector in frame $\mathcal{F}_V$; $t_i = \frac{i\tau}{N}$, with N being the discrete sample number for the primitive.

Note that $\mathcal{C}_{target}$ encodes the primary objective of target reaching. Its piecewise formulation promotes forward progress when the target lies beyond the primitive's reachable range, while preventing overshoot to facilitate precise target acquisition when the target is within reach. Meanwhile, $\mathcal{C}_{length}$ serves as a geometric regularizer that penalizes unnecessarily curved or weaving primitives and favors more direct trajectories.

We clarify the optimality of the expert strategy at two levels. a) *Individual-primitive level*: each candidate primitive is the optimal solution of a constrained minimum-jerk QP under its conditioned boundary states, dynamical limits, and fixed duration. It is optimal in the minimum-control-effort sense, not in the time-optimal sense. b) *Library-selection level*: the expert strategy pursues spatial optimality through a step-wise pipeline of free-space sampling, collision removal, and optimal selection. This design decouples obstacle avoidance from goal reaching, avoiding the trade-offs of jointly optimizing both in a single objective, and thus yields spatially near-straight expert trajectories.

B. Fast End-to-end Trajectory Planning Framework

We expect to perform fast online inference of executable trajectories relying solely on onboard sensors. To this end, we consider the following characteristics: a) *End-to-end policy*: Unifying the mapping, front-end path searching, and back-end trajectory solving within the classical planning framework into an integrated process, thereby eliminating inter-module latency and compounding errors; b) *Executable outputs*: Incorporating state continuity, target-reaching progress quality, and empirical obstacle avoidance and dynamical feasibility for direct execution by low-level controller; c) *Lightweight network*: With low inference computation and memory consumption, enabling efficient real-time onboard deployment. The framework consists of the following four components:

1) Policy Input-Output Design: To eliminate the need for back-end trajectory solving, the policy is intended to directly output the polynomial coefficients. For a minimum-jerk primitive expressed in frame $\mathcal{F}_V$, the coefficient matrix in (1) can be expanded as:

$$\boldsymbol{c} = \begin{bmatrix} \boldsymbol{c}_a \\ \boldsymbol{c}_b \end{bmatrix},\ \boldsymbol{c}_a = \begin{bmatrix} c_{5x} & c_{5y} & c_{5z} \\ c_{4x} & c_{4y} & c_{4z} \\ c_{3x} & c_{3y} & c_{3z} \end{bmatrix},\ \boldsymbol{c}_b = \begin{bmatrix} \frac{1}{2}a_{0x} & \frac{1}{2}a_{0y} & \frac{1}{2}a_{0z} \\ \|\boldsymbol{v}_0\| & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{8}$$

It can be seen that $\boldsymbol{c}_b \in \mathbb{R}^{3\times3}$ governs state continuity and can be directly determined from the current velocity $\boldsymbol{v}_0$ and acceleration $\boldsymbol{a}_0$. In contrast, $\boldsymbol{c}_a \in \mathbb{R}^{3\times3}$ needs to be learned to satisfy obstacle avoidance and target-reaching progress quality, which are associated with the perceived point clouds $\mathcal{P}$ and the target direction $\boldsymbol{d}_0$, respectively. By imitating expert coefficients that are carefully selected into the training dataset, the empirical dynamical feasibility of the inferred trajectories can also be reinforced. Therefore, the policy network is designed to take $\|\boldsymbol{v}_0\| \in \mathbb{R}^1$, $\boldsymbol{a}_0 \in \mathbb{R}^3$, $\boldsymbol{d}_0 \in \mathbb{R}^3$ and $\mathcal{P} \in \mathbb{R}^{m\times3}$ as inputs and output $\boldsymbol{c}_a \in \mathbb{R}^9$, where $\boldsymbol{c}_a$ is expanded into a vector here and m denotes the number of 3D point clouds, thereby enabling end-to-end generation of executable trajectories from sensory data without any further solving.

2) Network Architecture: We present *Npe2eNet*, shown in Fig. 2B, which primarily comprises a point cloud encoder, a feature fusion module, and a coefficient prediction head. The encoder plays a pivotal role in capturing environmental context. It first pre-processes arbitrary-sized point clouds to a fixed m_{fix} points (set to 666 to balance input size and perceptual sufficiency). Each point's 3D coordinates pass through an MLP ([128, 128, 64], ReLU), followed by max-pooling to a 64-D global descriptor, which is refined by another MLP ([64, 32], ReLU) to a 32-D representation. This is concatenated with the 7-D state into a 39-D input to a feature fusion MLP ([256, 512, 256], ReLU), yielding a compact yet expressive feature vector that encapsulates environmental, dynamical and target information. This feature finally feeds a coefficient prediction MLP ([128, 128, 9], ReLU) to output the nine polynomial coefficients.

Npe2eNet adopts a streamlined architecture that avoids complex operations like convolutions. It can be calculated that the network only contains approximately 0.355M parameters with a computational cost of approximately 34M FLOPs. The model occupies about 1.38MiB of storage. During online inference, the per-forward-pass consumes about 0.65MiB peak incremental memory and 11.49MiB peak total memory of GPU[2]. Compared with existing learning-based methods built on heavier architectures such as ResNet [14, 25], MobileNet [2], and Transformer [13], its resource demands are substantially lower, enabling fast and efficient real-time inference.

3) Training Method: In real flights, perceived point clouds vary significantly in density and spatial distribution across scenarios, and drone state measurements are inherently noisy. These discrepancies, arising from sensory mismatch, primarily constitute the sim-to-real gap in the end-to-end planning scheme. To bridge this gap and enhance the generalization capability of the simulation-trained policy for zero-shot transfer, we implement two strategies during training: 1) *Point cloud pre-processing*: Considering that real flight point clouds may deviate from the fixed network input size m_{fix} and the simulation dataset cannot cover all variations, we resample the dataset point clouds before formal training. For each data sample, with probability p_{pro}, the point cloud is downsampled by a factor r of its original size, where r is randomly selected from $[r_{min}, 1]$ and $0 \leq r_{min} \leq 1$. If the resulting size is smaller than m_{fix}, it is padded with points whose coordinates are assigned large values (set as 100), denoting distant obstacles; otherwise, it is randomly downsampled to m_{fix}. This broadens density coverage and mitigates unseen scenes during training. Note that in online inference stage, the input point clouds are directly downsampled or padded to m_{fix}. 2) *Domain randomization*: Gaussian noise is injected into both the input state values and the coordinates of the pre-processed point cloud, with standard deviations set to proportions r_{σ_s} and r_{σ_p} of their respective magnitudes. This further improves the robustness of the policy network against sensory noise.

4) Loss Function: The network is trained to minimize the Mean Squared Error (MSE) loss between expert coefficients $\boldsymbol{c}_a^*$ and predicted values $\hat{\boldsymbol{c}}_a$:

$$\mathcal{L}_{\text{coeff}} = \text{MSE}(\boldsymbol{c}_a^*, \hat{\boldsymbol{c}}_a) = \frac{1}{M}\sum_{k=1}^{M}(c_{a,k}^* - \hat{c}_{a,k})^2. \quad (9)$$

where subscript k indexes the M elements of $\boldsymbol{c}_a \in \mathbb{R}^{3\times 3}$, with $M = 9$. AdamW optimizer [28] (learning rate $lr = 1.0 \times 10^{-3}$, weight decay $= 1.0 \times 10^{-4}$) with cosine annealing scheduling (half-cycle length $T_{max} = 0.3\times$maximum epochs, minimum learning rate $\eta_{min} = 0.1 \times lr$) is utilized to ensure fast, stable convergence and mitigate overfitting. Training employs a batch size of 128 (validation 64), with a maximum of 600 epochs and early stopping patience of 20.

V. Evaluations

A. Implementation Details

We conduct dataset collection, network training, and simulation tests on a desktop computer with i7-10700K CPU, GTX 1060 GPU and Ubuntu 20.04 system. The randomized simulation maps for dataset collection consist mainly of cylinders, rectangular columns, and rings, with overall obstacle densities ranging from 1/40 to 1/4. A total of one million valid samples are collected, with each requiring only approximately 0.08s to generate, and the data are stored in H5 format for training the network via PyTorch. The resulting policy model is deployed to ROS via LibTorch for online inference. In real-world experiments, we implement the learned policy model on a quadrotor without any further fine-tuning. The platform integrates a Livox Mid-360 LiDAR[3] for onboard sensing, an NVIDIA Jetson Orin NX[4] for end-to-end planning and SE(3) geometric tracking control [29], and a PX4[5] flight controller for low-level attitude control.

The planned trajectory is executed in an event-triggered receding-horizon manner. Specifically, replanning is triggered by whichever of the following conditions occurs first: the quadrotor travels the preset distance threshold $d_{\text{rep}} = 0.5$m, the execution time reaches the upper limit $T_{\text{rep}}^{\max} = 2\tau/3$, or a potential collision is detected. This mechanism allows the actual execution time to adapt naturally to the flight speed while remaining within the valid time window $[0, \tau]$. Additionally, each inferred trajectory undergoes a lightweight post-inference collision check [3]–[5] before being sent to the controller. The first two-thirds of the trajectory are sampled at intervals of 0.05s and queried against an inflated occupancy grid via an $O(1)$ lookup. Note that the checker operates in an independent parallel module outside the end-to-end trajectory generation pipeline and serves as a decoupled engineering safeguard against rare inputs outside the training distribution.

[2]Peak incremental memory measures extra dynamic GPU memory for inference operation only, while peak total memory measures the overall dynamic GPU memory needed for inference, including model parameters, input tensors, intermediates, etc.

[3]https://www.livoxtech.com/mid-360

[4]https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/

[5]https://px4.io/

TABLE I

ABLATION STUDY ON DOMAIN RANDOMIZATION PARAMETERS

No.	r_{σ_s}	r_{σ_p}	Success Rate
A0	0.00	0.00	0.904
A1	0.03	0.03	0.924
A2	0.03	0.04	**0.936**
A3	0.04	0.04	0.912
A4	0.06	0.06	0.892

Note: conducted with the point cloud pre-processing module disabled

TABLE II

ABLATION STUDY ON POINT CLOUD PRE-PROCESSING PARAMETERS

No.	p_{pro}	r_{min}	Success Rate
B0	0.0	1.0	0.936
B1	0.3	0.7	0.944
B2	0.6	0.5	**0.972**
B3	0.6	0.3	0.932
B4	0.9	0.1	0.888

Note: conducted on top of the selected A2 domain randomization setting

B. Ablation Study

We first conduct a series of ablation experiments to examine how the training augmentation techniques affect network performances, with a focus on the key parameters in the domain randomization (i.e., r_{σ_s} and r_{σ_p}) and point cloud pre-processing (i.e., p_{pro} and r_{min}) modules.

All experiments are performed in 50×20×10m simulated maps with *dense* obstacle configurations (160 cylinders and 40 rings, minimum spacing 2m, and overall density 1/5). The drone autonomously navigates a 64m start-target distance, with success defined as reaching the target without collision (drone radius 0.15 m) and within a 1m target tolerance. The maximum speed is limited to 4m/s. Each ablation setting is evaluated over 250 runs. In every run, obstacle positions and radii are randomly generated. Gaussian noise with zero mean and a 5% standard deviation is injected into both the network input states and point cloud coordinates to emulate measurement uncertainties. The results are reported in Table I and Table II.

Experiments A0 to A4 assess different (r_{σ_s}, r_{σ_p}) combinations under domain randomization, with the point cloud pre-processing module disabled. Compared with the baseline A0, success rates are observed to improve in A1 to A3 when noise is added to state values and point cloud coordinates during training. These perturbations enhance robustness to proprioceptive and exteroceptive disturbances, respectively, and mitigate overfitting to idealized clean observations. However, excessive noise, as in A4, degrades performance due to distortion of the original observation distribution and impairment of stable input-output learning, leading to potential misinterpretation of obstacles and targets. By evaluating both symmetric and asymmetric combinations, we select A2 (r_{σ_s}=0.03, r_{σ_p}=0.04, and success rate 0.936) as the domain randomization setting.

Experiments B0 to B4 further investigate the impact of different (p_{pro}, r_{min}) combinations with the point cloud pre-processing module added on top of domain randomization, where A2 serves as the baseline B0. For autonomous flight in clutters, point clouds encode critical obstacle constraints for safe navigation. Previous noise injection on point cloud coordinates essentially perturbs spatial positions only, but overlooks variations in point cloud density. The proposed pre-processing module exposes the network to a broader density range during training, enhancing adaptation to fluctuations in obstacle point counts. Consequently, success rates are further improved (as in B1 and B2) over the domain randomization baseline B0 in complex scenarios. Moreover, p_{pro} governs the pre-processing frequency, while r_{min} sets the lower bound for point cloud density. Insufficient p_{pro} or excessive r_{min} fails to capture density fluctuations adequately, leading to marginal robustness improvements in B1. Conversely, excessively high p_{pro} paired with minimal r_{min} causes over-sparsification, eroding the obstacle geometric fidelity within the dataset and in turn compromising success rates in B3 and B4. The balanced configuration B2, with p_{pro} = 0.6 and r_{min} = 0.5, achieves a peak success rate of 0.972. These results verify the efficacy of the proposed point cloud pre-processing strategy.

TABLE III

SENSITIVITY ANALYSIS ON POINT CLOUD INPUT SIZE

m_{fix}	Success Rate	Inference Latency (ms)	Peak Incremental Memory (MiB)	Peak Total Memory (MiB)
333	0.916	0.43	0.325	11.162
666	0.972	0.67	0.650	11.491
1024	0.968	0.78	1.003	11.844
2048	0.976	0.93	2.001	12.856

We further conduct a sensitivity analysis on the fixed point cloud input size m_{fix}, examining its effects on success rate, inference latency, peak incremental memory, and peak total memory. The results are reported in Table III. It can be seen that reducing m_{fix} to 333 lowers latency and memory consumption, which is reasonable since the point cloud encoder operates in a point-wise manner, making latency and dynamic memory scale with input size. However, the success rate in case 333 drops to 0.916, indicating insufficient environmental information for reliable planning compared to case 666. Moreover, increasing m_{fix} to 1024 and 2048 yields only marginal changes in success rate, but incurs higher latency and memory costs, with peak incremental memory rising by 54.3% and 207.8%, respectively. These results indicate that case 666 offers a favorable balance among planning reliability, computational efficiency, and memory cost.

C. Acceleration Continuity Analysis

To demonstrate the resolution of the acceleration discontinuity inherent in traditional offline primitive library-based planners, we compare with the single version of *Primitive-swarm* [9] and plot the curves (Fig. 4) under *dense* environments. It is evident that acceleration jumps occur

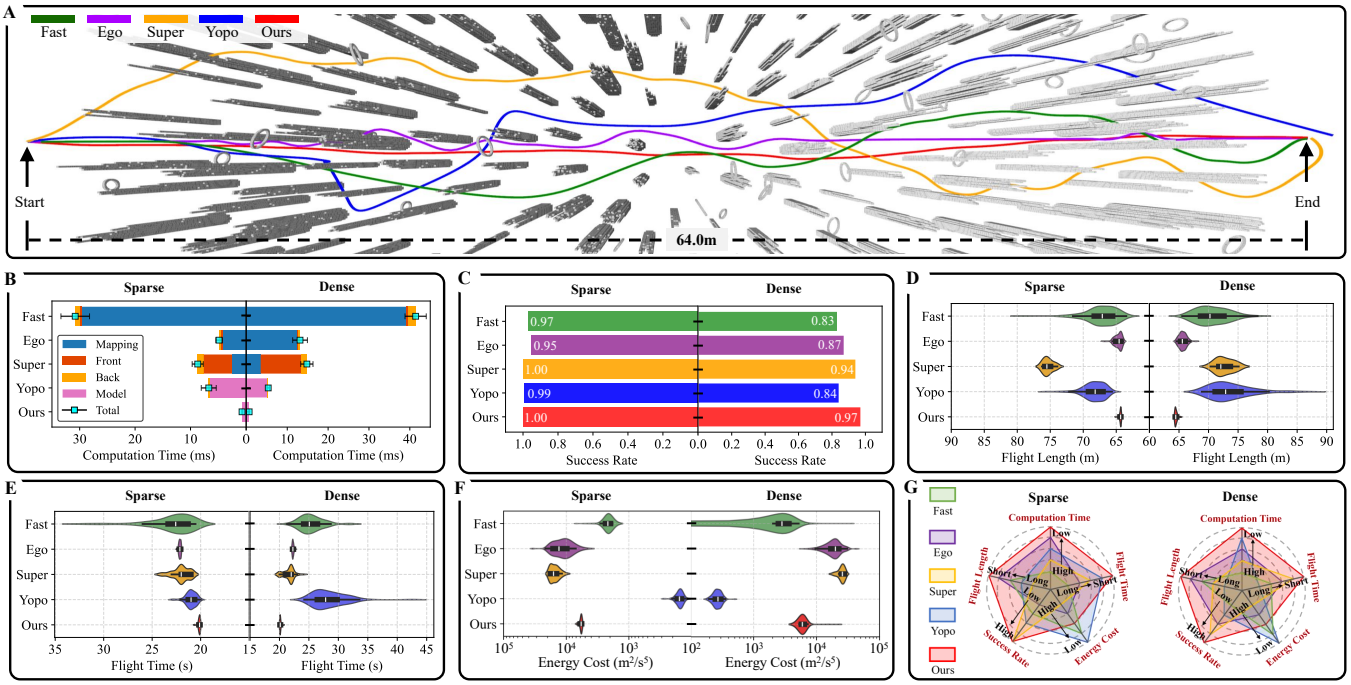

Fig. 3. **Benchmark results under different obstacle densities. (A)** Example (successful) trajectories in dense scenarios. **(B)** Computation time, where "Mapping" includes grid map, point cloud map, and ESDF construction, "Front" denotes path finding and corridor generation, "Back" indicates trajectory optimization or closed-form BVP solving, and "Model" refers to network inference. **(C)** Success rate. **(D)** Flight length. **(E)** Flight time. **(F)** Energy cost. **(G)** Overall comparison across the five metrics.

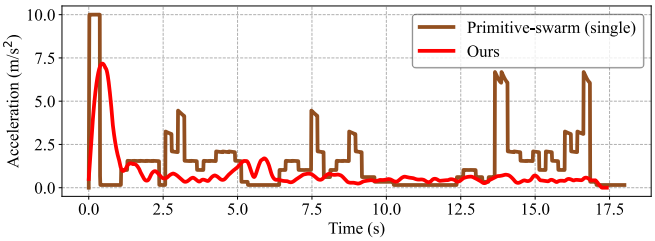

Fig. 4. **Acceleration curves.** Our method overcomes the high-order state discontinuity problem of traditional offline library-based planners [9].

throughout the entire flight of [9]. This occurs because the library cannot cover all possible acceleration states due to size constraints for onboard memory and latency. As a result, when an uncovered initial acceleration arises during online selection, discontinuities appear, degrading trajectory quality and increasing motor load. In contrast, our method explicitly considers higher-order state continuity when designing the network input and output, thereby successfully addressing this problem.

D. Benchmarks

In this section, we compare the proposed method with four representative local planners, including three well-established approaches under the classical hierarchical planning framework and one state-of-the-art learning-based planner, namely: 1) *Fast* [8]: Kinodynamic path searching on an incrementally updated voxel map, followed by gradient-based trajectory optimization relying on a constantly maintained ESDF for safety regularization; 2) *Ego* [5]: A* path searching on the constructed grid map, with subsequent ESDF-free trajectory optimization leveraging collision costs derived directly from grid-obstacle information; 3) *Super* [3]: Path searching and flight corridor generation directly on an efficient spatiotemporal sliding point cloud map, followed by MINCO-based [30] dual-trajectory optimization that jointly accounts for high-speed exploration and safety assurance; 4) *Yopo* [14]: Objective-function-guided unsupervised learning for the network, inferring the end-state related terms of all pre-sampled primitives from sensory inputs, after which one is selected for closed-form BVP solving.

All planners are tested in simulated maps with different obstacle densities. The autonomous navigation task and *dense* environment configuration follow those described in Sect. V-B. The *sparse* configuration consists of 40 cylinders and 10 rings, with a minimum spacing of 4m and an overall density of 1/20. All baselines are evaluated with their official open-source implementations and default parameters. For a fair comparison, all planners use the same 90° ×60° Field-of-View (FOV), collision radius of 0.15m, maximum velocity of 4m/s, and simulator (including the controller and quadrotor model) adopted from [8]. The target reaching tolerance is still set to 1m, but relaxed to 4m for *Yopo*, as it cannot satisfy stricter tolerances. Each planner is evaluated over 100 runs in both dense and sparse scenarios, with obstacles randomly generated in each run. Flight visualizations and statistical results are presented in Fig. 3.

It can be seen that our method outperforms the baselines in both sparse and dense environments (Fig. 3G), whereas the baselines, despite performing well in sparse settings, deteriorate in dense scenarios. In terms of computation time (Fig. 3B), *Fast* incurs the highest cost, primarily due to ESDF maintenance. Although *Ego* and *Super* avoid this

overhead, they still require grid map construction or point cloud-based corridor generation, so their planning time is highly sensitive to obstacle density and increases with it. *Yopo* circumvents mapping and front-end processing, running faster than the first three methods in dense environments. However, in sparse settings, its runtime even slightly exceeds *Ego* due to the need for inference over all pre-sampled primitives, complex in-network operations, and back-end BVP solving costs. Consequently, when grid map construction exerts less influence in sparse environments, *Yopo* loses its efficiency advantage over *Ego*. Our method achieves the shortest computation time, averaging 0.70ms (sparse) and 0.68ms (dense), about 7 to 60 times faster than the baselines, with stable efficiency independent of densities due to its highly integrated end-to-end planning policy and compact yet effective architecture.

Regarding task-oriented trajectory optimality (Fig. 3D and Fig. 3E), the four baselines yield longer and slower trajectories than the proposed method. This arises from their strong coupling of obstacle avoidance and goal navigation into a complex multi-constraint optimization problem (including *Yopo*, which essentially formulates the same problem but solves it with a neural network). As a result, they must trade off among competing factors, often becoming trapped in local minima and producing detours, as shown in Fig. 3A. By contrast, our method benefits from imitating the customized primitive strategy that decouples the two problems by first eliminating colliding primitives and then concentrating on selecting the task-oriented optimal primitive from the safe set, thereby avoiding compromises for obstacle avoidance. This design enables near-direct flight distances (64.51m in dense and 64.39m in sparse) and the shortest flight time to the target.

Fig. 3F shows the energy cost measured by the integral of squared-jerk. Consistent with the trajectory results, our method yields lower accumulated squared-jerk than *Ego* and *Super*, while the lower values of *Fast* and *Yopo* are achieved at the expense of substantially longer detours and less direct motion toward the target. This indicates that our method achieves a favorable balance between efficient target reaching and energy consumption. Finally, while the other planners exhibit declining success rates in dense environments, our method sustains the highest rate of 0.97 (Fig. 3C). This stems from its ultra-low latency, enabling rapid responses to dense obstacles even at high speeds, further reinforced by point-cloud pre-processing and domain randomization for robust environmental abstraction.

Fig. 5 plots success rates against increasing average speeds for all compared planners. Results indicate that all methods achieve perfect success rates at low speeds (≈2m/s). However, performance degrades significantly for *Fast*, *Ego*, and *Yopo* as speed increases, particularly in dense scenarios. In contrast, *Super* and our proposed method maintain success rates above 0.80 even at high average speeds of around 6m/s. The decline for *Fast* and *Ego* primarily arises from hierarchical latency and compounding errors. Moreover, their joint optimization processes frequently converge to local

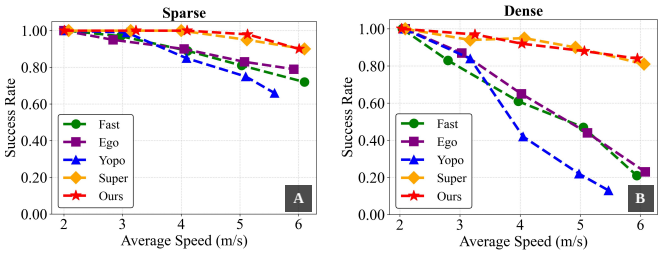

Fig. 5. **Success rates across average speeds. (A)** Results in sparse environments. **(B)** Results in dense environments.

minima, failing to generate safe trajectories under high speed motion. For *Yopo*, we observe inconsistent velocity profiles characterized by abrupt accelerations, leading to collisions when obstacles appear suddenly. Furthermore, the lack of explicit density-aware training also limits its generalization across varying obstacle configurations. Although *Super* employs a hierarchical planning structure, its complex dual-trajectory mechanism ensures an effective balance between speed and safety. Our method achieves comparable success rates to *Super* but avoids its extensive hand-crafted rules and pronounced trajectory detours as demonstrated in Fig. 3A. Through a streamlined end-to-end architecture, our approach produces straighter flight paths while maintaining the same high level of robustness.

E. Generalization Evaluations

To evaluate the generalization capability of the proposed planner, we test the learned policy via simulation in environments not encountered during training. Two types of more complex new maps are employed: 1) an open-sourced point cloud map collected from real-world forests [31], which we crop to a volume of 60×50×20m. This map contains not only tree trunks, branches, and ground vegetation, but also spatially distributed noise points, as illustrated in Fig. 6A; and 2) a simulated unstructured 3D obstacle map resembling caves and mountainous terrain [16], generated using Perlin noise with dimensions of 60×50×10m. In this second map, obstacles exhibit diverse irregular shapes and varying densities that differ from those in the training datasets, as shown in Fig. 6B. For each map type, we perform 150 autonomous traversal trials. Start and target positions are randomly initialized with separation distances of 70∼80m per trial. The policy is the one obtained from ablation experiment B2 in Sect. V-B, without any fine-tuning to the new environments. The resulting success rates are 0.907 and 0.833, respectively. It can be seen that although performance degrades relative to training environments, a common limitation of learning-based methods, the results still remain high and exceed those reported in [16]. This robustness is attributed to the synergistic effects of domain randomization and point cloud pre-processing. These strategies improve tolerance to spatial noise while enhancing adaptability to density variations. Consequently, the proposed planner achieves comparable generalization across both noise-corrupted and density-varying new environments.

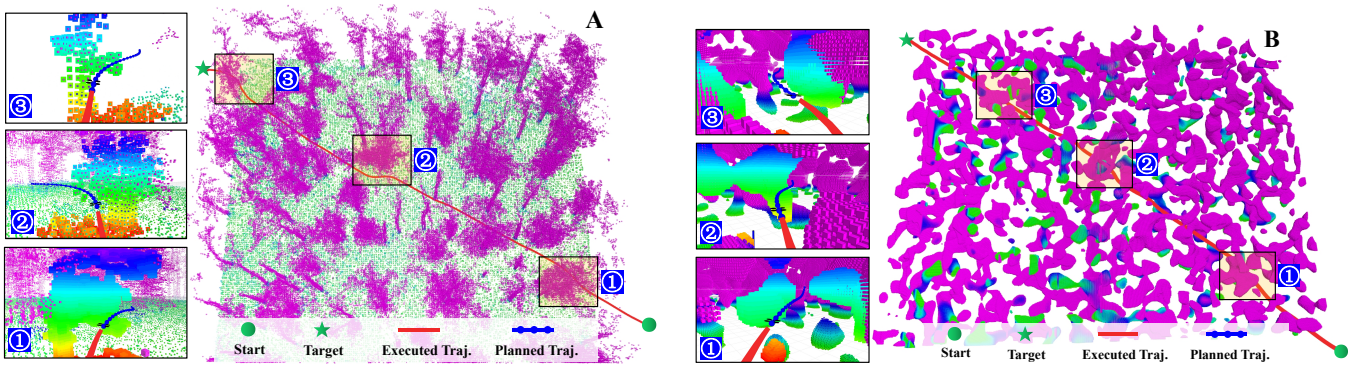

Fig. 6. **Generalization test environments.** The main panels show the global top view, while the adjacent insets present the corresponding first person views of the indexed local regions. **(A)** Point cloud map collected from real world forests [31]. **(B)** Simulated 3D obstacle map resembling caves and mountainous terrain [16].

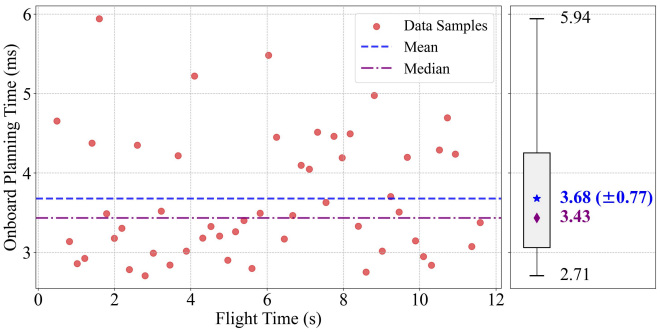

Fig. 7. **Statistics of onboard planning time.** The distribution characterizes the superior real-time onboard efficiency of the proposed end-to-end local planner with a mean planning time of only 3.68ms throughout the flight.

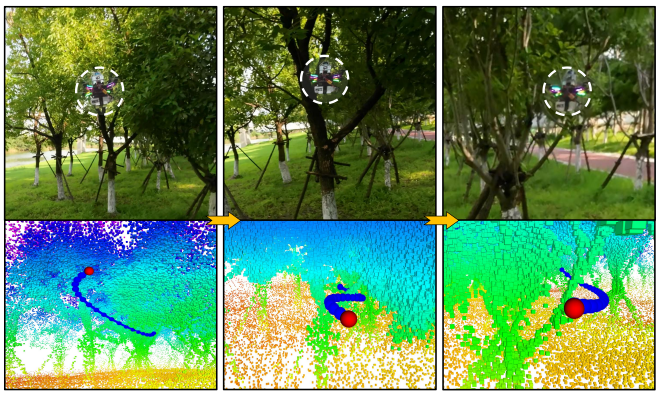

Fig. 8. **Field snapshots and onboard planning visualizations in thick foliage.** Red spheres and blue curves denote current positions and planned trajectories, respectively. The quadrotor performs safe and robust maneuvering through restricted spaces with a minimum passage under 0.8m.

F. Real-world Experiments

We validate our method through real-world experiments in both outdoor unknown forests and indoor cluttered environments. All scenarios are previously unseen, and the training data come solely from simulation without real-world fine-tuning. We refer readers to the supplementary video for more information.

As illustrated in Fig. 1A, we first conduct the outdoor navigation experiments in a non-uniform forest with approximately 1/6 trees/m^2 density and 0.3m average tree diameter. Each tree is irregularly surrounded by multiple tilted wooden stakes, forming a complex cluttered environment. The results show that the quadrotor autonomously flies to a target 60m ahead (within 1m reaching tolerance) under random disturbances including wind and sensor noise. It reaches a maximum speed of 6.10m/s, completes the flight within 12s without collisions, and maintains a nearly straight trajectory without significant detours, demonstrating effective sim-to-real transfer capability.

Fig. 7 presents the statistics of onboard planning time for end-to-end planning throughout the entire flight. The mean is 3.68ms with a standard deviation of $\pm$0.77ms. The median is as low as 3.43ms, with values ranging from 2.71ms to 5.94ms. These metrics highlight the exceptional computational efficiency of the proposed method for onboard deployment.

The flight performance in a denser forest with thick foliage is depicted in Fig. 8. Despite the cluttered obstacles and the narrowest passage under 0.8m (quadrotor diameter $\approx$0.3m), the planner can still successfully generate safe trajectories and execute traversal maneuvers. Furthermore, we also conduct indoor experiments in a confined space with randomly placed obstacles, as shown in Fig. 1B. The quadrotor needs to sequentially visit multiple preset targets within a 0.5m reaching tolerance before proceeding to the next. This experiment validates the stability and safety of the proposed planner during continuous multi-target navigation in unknown, cluttered environments.

VI. Conclusion and Future Work

This paper leverages imitation learning to inherit the advantages of offline primitive-based methods in generating high-quality trajectories, while addressing their inherent discontinuity drawback through online network inference. The resulting planner directly outputs polynomial coefficients from sensory inputs. It produces practically controller-executable trajectories without back-end solving and forms a highly integrated end-to-end planning scheme. The proposed point cloud pre-processing technique during training improves success rates across varying obstacle densities and

enhances generalization capability. Extensive benchmarks against state-of-the-art baselines in both dense and sparse environments demonstrate consistent superiority in computation time, success rate, flight length, and flight time. Furthermore, zero-shot deployment in indoor and outdoor real-world experiments validates the robustness and efficiency of the proposed planner. Nevertheless, we acknowledge that the current framework is primarily designed for static obstacles and remains limited in handling highly dynamic obstacles. Moreover, as a local planner relying on instantaneous limited Field-of-View inputs, it may struggle to escape large local traps such as dead ends and U-shaped obstacles. Overcoming these limitations will be investigated as our future work.

REFERENCES

[1] G.-Z. Yang, J. Bellingham, P. E. Dupont, P. Fischer, L. Floridi, R. Full, N. Jacobstein, V. Kumar, M. McNutt, R. Merrifield, *et al.*, "The grand challenges of science robotics," *Science robotics*, vol. 3, no. 14, p. eaar7650, 2018.

[2] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Learning high-speed flight in the wild," *Science Robotics*, vol. 6, no. 59, p. eabg5810, 2021.

[3] Y. Ren, F. Zhu, G. Lu, Y. Cai, L. Yin, F. Kong, J. Lin, N. Chen, and F. Zhang, "Safety-assured high-speed navigation for mavs," *Science Robotics*, vol. 10, no. 98, p. eado6187, 2025.

[4] X. Zhou, X. Wen, Z. Wang, Y. Gao, H. Li, Q. Wang, T. Yang, H. Lu, Y. Cao, C. Xu, *et al.*, "Swarm of micro flying robots in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm5954, 2022.

[5] X. Zhou, Z. Wang, H. Ye, C. Xu, and F. Gao, "Ego-planner: An esdf-free gradient-based local planner for quadrotors," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 478–485, 2020.

[6] M. Lu, X. Fan, H. Chen, and P. Lu, "Fapp: Fast and adaptive perception and planning for uavs in dynamic cluttered environments," *IEEE Transactions on Robotics*, vol. 41, pp. 871–886, 2025.

[7] M. Wang, Q. Wang, Z. Wang, Y. Gao, J. Wang, C. Cui, Y. Li, Z. Ding, K. Wang, C. Xu, *et al.*, "Unlocking aerobatic potential of quadcopters: Autonomous freestyle flight generation and execution," *Science Robotics*, vol. 10, no. 101, p. eadp9905, 2025.

[8] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, "Robust and efficient quadrotor trajectory generation for fast autonomous flight," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3529–3536, 2019.

[9] J. Hou, X. Zhou, N. Pan, A. Li, Y. Guan, C. Xu, Z. Gan, and F. Gao, "Primitive-swarm: An ultra-lightweight and scalable planner for large-scale aerial swarms," *IEEE Transactions on Robotics*, 2025.

[10] M. Ryll, J. Ware, J. Carter, and N. Roy, "Efficient trajectory planning for high speed flight in unknown environments," in *2019 International conference on robotics and automation (ICRA)*. IEEE, 2019, pp. 732–738.

[11] M. Dharmadhikari, T. Dang, L. Solanka, J. Loje, H. Nguyen, N. Khedekar, and K. Alexis, "Motion primitives-based path planning for fast and agile exploration using aerial robots," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 179–185.

[12] J. Zhang, C. Hu, R. G. Chadha, and S. Singh, "Falco: Fast likelihood-based collision avoidance with extension to human-guided navigation," *Journal of Field Robotics*, vol. 37, no. 8, pp. 1300–1313, 2020.

[13] Z. Han, M. Tian, Z. Gongye, D. Xue, J. Xing, Q. Wang, Y. Gao, J. Wang, C. Xu, and F. Gao, "Hierarchically depicting vehicle trajectory with stability in complex environments," *Science Robotics*, vol. 10, no. 103, p. eads4551, 2025.

[14] J. Lu, X. Zhang, H. Shen, L. Xu, and B. Tian, "You only plan once: A learning-based one-stage planner with guidance learning," *IEEE Robotics and Automation Letters*, vol. 9, no. 7, pp. 6083–6090, 2024.

[15] Y. Wu, X. Sun, I. Spasojevic, and V. Kumar, "Deep learning for optimization of trajectories for quadrotors," *IEEE Robotics and Automation Letters*, vol. 9, no. 3, pp. 2479–2486, 2024.

[16] Z. Han, L. Xu, L. Pei, and F. Gao, "Dynamically feasible trajectory generation with optimization-embedded networks for autonomous flight," *IEEE Robotics and Automation Letters*, vol. 10, no. 10, pp. 9995–10 002, 2025.

[17] J. J. Damanik, J.-W. Jung, C. A. Deresa, and H.-L. Choi, "Lics: Navigation using learned-imitation on cluttered space," *IEEE Robotics and Automation Letters*, 2024.

[18] K. Tejaswi and T. Lee, "Constrained imitation learning for a flapping wing unmanned aerial vehicle," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 10 534–10 541, 2022.

[19] X. Yang, J. Cheng, and N. Michael, "An intention guided hierarchical framework for trajectory-based teleoperation of mobile robots," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 482–488.

[20] Y. Lee, J. Park, B. Jeon, S. Jung, and H. J. Kim, "Bpmp-tracker: A versatile aerial target tracker using bernstein polynomial motion primitives," *IEEE Robotics and Automation Letters*, vol. 9, no. 12, pp. 10 938–10 945, 2024.

[21] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," *IEEE transactions on robotics*, vol. 31, no. 6, pp. 1294–1310, 2015.

[22] X. Yang, A. Agrawal, K. Sreenath, and N. Michael, "Online adaptive teleoperation via motion primitives for mobile robots," *Autonomous Robots*, vol. 43, no. 6, pp. 1357–1373, 2019.

[23] M. Collins and N. Michael, "Efficient planning for high-speed mav flight in unknown environments using online sparse topological graphs," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 11 450–11 456.

[24] J. Zhang, R. G. Chadha, V. Velivela, and S. Singh, "P-cap: Pre-computed alternative paths to enable aggressive aerial maneuvers in cluttered environments," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 8456–8463.

[25] J. Lu, B. Tian, H. Shen, X. Zhang, and Y. Hui, "Lpnet: A reaction-based local planner for autonomous collision avoidance using imitation learning," *IEEE Robotics and Automation Letters*, vol. 8, no. 11, pp. 7058–7065, 2023.

[26] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 2520–2525.

[27] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "Osqp: An operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.

[28] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[29] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se (3)," in *49th IEEE conference on decision and control (CDC)*. IEEE, 2010, pp. 5420–5425.

[30] Z. Wang, X. Zhou, C. Xu, and F. Gao, "Geometrically constrained trajectory optimization for multicopters," *IEEE Transactions on Robotics*, vol. 38, no. 5, pp. 3259–3278, 2022.

[31] K. Chaney, F. Cladera, Z. Wang, A. Bisulco, M. A. Hsieh, C. Korpela, V. Kumar, C. J. Taylor, and K. Daniilidis, "M3ed: Multi-robot, multi-sensor, multi-environment event dataset," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2023, pp. 4016–4023.