EmbodiedCoder: Parameterized Embodied Action via Modern Coding Agent

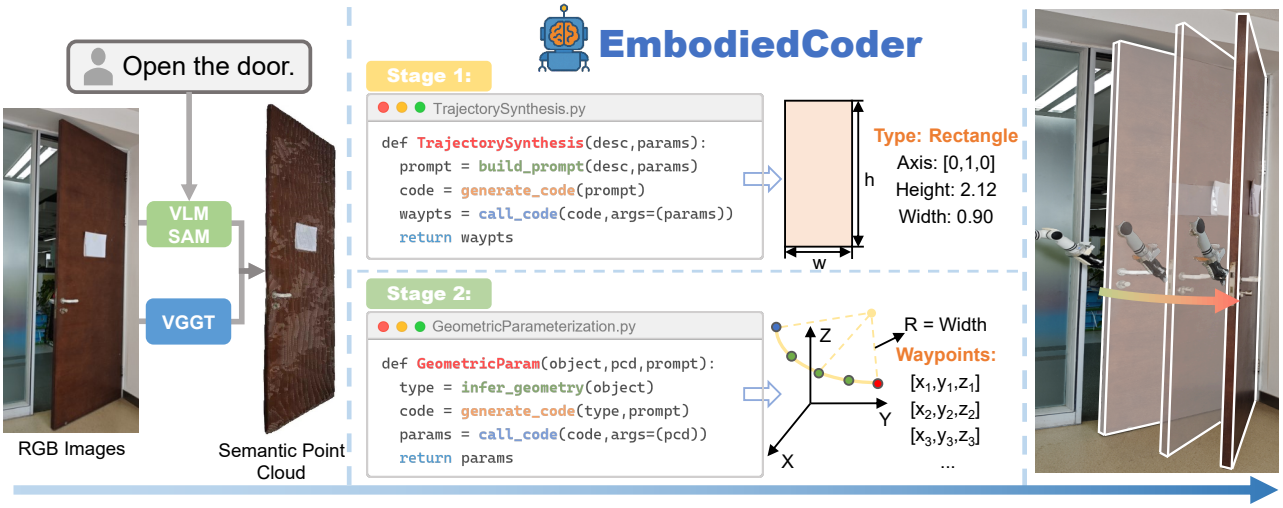

Fig. 1: **EmbodiedCoder** leverages large language models to achieve object parameterization and task trajectory synthesis through prompting, without the need for additional training. This enables direct application of model knowledge to everyday tasks. The figure illustrates the subtask of ***Open the door*** obtained from a long-horizon task decomposition. By parameterizing the door and fitting a semicircular trajectory along its hinge axis, the robotic arm can execute the opening motion by following sampled waypoints along the trajectory.

***Abstract*— Achieving human-level proficiency in diverse robotic tasks within unstructured environments remains a central challenge in robotics. Recent advances in vision-language-action (VLA) models have enabled end-to-end mapping from sensory inputs and natural language to robot actions, but these models struggle to generalize to novel scenarios and demand extensive annotated datasets. Existing hierarchical approaches mitigate this issue by using vision-language models as high-level planners that trigger predefined primitives, yet they are fundamentally limited by the fixed set of available skills. We introduce EmbodiedCoder, a training-free framework for open-world mobile manipulation that integrates the knowledge of large-scale language and coding models directly into robotic control. EmbodiedCoder combines semantic scene understanding with structured geometric representations, parameterizing objects to capture their functional affordances. Using this information, the system synthesizes executable trajectory code that respects both task semantics and physical constraints, enabling robots to perform complex, contact-rich manipulations without additional data collection or fine-tuning. Experiments on real mobile robots demonstrate that EmbodiedCoder achieves superior generalization across diverse long-horizon tasks, including previously unseen objects and environments. This work provides a new paradigm for bridging high-level reasoning and low-level control, moving beyond predefined primitives toward scalable, versatile robot intelligence.**

I. INTRODUCTION

Enabling robots to perform diverse tasks with human-like proficiency in complex, unstructured environments has long been a central goal in robotics [1].

Recent progress in vision-language-action (VLA) models has brought this ambition closer to reality by enabling end-to-end mapping from sensory inputs and natural language instructions to robot actions. However, their generalization ability remains limited. Even slight changes in the environment, such as variations in object appearance or illumination, can significantly degrade performance. Furthermore, these models typically require massive annotated datasets, making their deployment costly and less scalable.

To overcome these challenges, hierarchical strategies have been proposed. A common design, such as DovSG [2] and OK-Robot [3], is to employ vision-language models (VLMs) [4], [5] as high-level planners that decompose tasks into subtasks and invoke predefined robotic primitives, such as navigation, grasping, or pick-and-place. This paradigm is theoretically appealing, since it allows robots to leverage the commonsense knowledge encoded in large-scale models while relying on robust control modules for low-level execution. In practice, however, its effectiveness is fundamentally constrained by the repertoire of available primitives. Many real-world tasks, such as opening doors or drawers, require nuanced interactions that cannot be reduced to a finite set of primitives.

Recent work has attempted to extend beyond this primitive-based architecture by generating executable code for robotic control. Code-as-Policies [6] demonstrated that an LLM can write low-level code to control a robot, but this early attempt was limited to tasks with very simple, specific geometries. RoboCodeX [7] uses a multimodal code

generation framework to broaden task generality, yet it relies on learned models to handle physical constraints, which reduces its adaptability to novel scenarios. VoxPoser [8] computes obstacle-aware end-effector trajectories for tasks like drawer opening, but it cannot perform more intricate, contact-rich manipulations. Likewise, Code-as-Monitor [9] generates code to detect and recover from execution failures, but it does not expand the robot's basic manipulation repertoire beyond the original primitives. For wheeled robots, the task complexity becomes even higher [10]. The robot must be able to retain information about the environment, which allows it to incorporate objects beyond its immediate field of view into the task planning process.

To address these challenges, we introduce **EmbodiedCoder**, a new paradigm for open-world mobile robot manipulation. This framework integrates the knowledge embedded in large-scale models directly into robotic actions, enabling robots to execute meaningful operations even in novel environments with previously unseen objects. Importantly, the entire pipeline is training-free, eliminating the need for additional costly data collection or fine-tuning.

Our method projects high-level instructions into executable robot trajectories by prompting an LLM with real-world scene and object information. We first employ VGGT [11] and VLM to capture the environment and ground object semantics into 3D point representations, providing the robot with a structured understanding of the scene. Guided by this semantic representation and environmental constraints, the system decomposes the instruction into subtasks. Each relevant object is then fitted with a parameterized geometric model encoding its functional affordances. For instance, a drawer can be approximated as a cuboid with a principal axis that indicates its pulling direction. Second, the coding model synthesizes trajectory code that respects both physical constraints and task semantics, producing motion commands that can be directly executed by the robot. Through this integration, the system achieves robust performance in novel environments without additional training.

In summary, EmbodiedCoder offers a new perspective on robot learning by combining the structural abstraction of objects with the generative power of coding models. It not only alleviates the dependency on predefined primitives but also eliminates costly data collection and fine-tuning. Our contributions are threefold:

- We introduce a framework that integrates large language and coding models with embodied agents, enabling complex long-horizon manipulations in realistic environments.
- We propose a novel method for parameterizing objects into functional geometric abstractions, allowing pretrained knowledge to be grounded into executable trajectories for sophisticated tasks such as door opening.
- We validate EmbodiedCoder on real mobile robots and demonstrate its effectiveness in handling diverse tasks, showing improved generalization and training-free deployment compared to existing approaches.

II. RELATED WORKS

A. Data-Driven Robotic Policies

Vision-Language-Action (VLA) models map visual observations and instructions directly to low-level robot actions via large transformer policies [12], [13], [14], [15], [16]. For example, RT-2 [17] extends earlier multi-task policies by incorporating web-scale vision-language pretraining to improve zero-shot semantic understanding. However, such models demand massive robot demonstration datasets (e.g., RT-1 [18] used 130k teleoperated trajectories collected over 17 months) and often struggle to generalize under distribution shifts (e.g., changes in lighting or object appearance). Recent works seek to improve efficiency and robustness [19], [20], [21], [22]. TinyVLA [23] is a compact VLA model that achieves faster inference and greater data efficiency by eliminating the need for extensive pre-training. Meanwhile, GR00T N1 [24] introduces a dual-system architecture for generalist humanoid control, combining fast reactive execution with slower deliberative reasoning. These advances broaden VLA capabilities, but even state-of-the-art policies still rely on curated datasets and fail to achieve systematic generalization in novel scenarios. [25], [26], [27]

B. LLM-Driven Trajectory and Code Generation

Another line of research uses large language models (LLMs) as high-level planners or policy generators for robotics [28], [29]. Code-as-Policies [6] showed that an LLM can write robot-executable code from a natural language command by stitching together existing skill primitives. This enables flexible recombination of behaviors, but generalization is limited by the fixed API library of actions. VoxPoser [8] produces 3D affordance maps from language descriptions, which serve as objective functions for motion planning. This approach enabled zero-shot execution of tasks like opening drawers with the ability to replan online, though it focused on relatively simple manipulations. ReKep [30] uses vision models to detect keypoints on objects, then prompts an LLM to output Python cost functions over those keypoints' relations. Optimizing these relational constraints yields multi-stage, long-horizon manipulation behaviors without task-specific training, although tasks not easily described by sparse keypoint relations remain difficult. RoboCodeX [7] is a multimodal code-generation framework that decomposes a high-level instruction into object-centric manipulation units with specified affordances and safety constraints, then generates structured code for each unit. Using a curated multimodal dataset for pre-training and self-refinement, RoboCodeX [7] achieved state-of-the-art results across simulated benchmarks and real robots. Nonetheless, its reliance on specialized training data limits adaptability beyond its training distribution. Complementary to planning, Code-as-Monitor [9] translates natural language constraints into executable monitors for proactive and reactive failure detection. It improves robustness by catching and handling failures at runtime, but does not expand the robot's underlying skill set. In summary, these LLM-driven

methods [31], [28], [29] illustrate how language models can compose complex behaviors and guard against failures, yet they still struggle with contact-rich interactions and often depend on substantial prior data or predefined skill libraries.

C. Modular Systems with Predefined Skills

A third strategy couples high-level reasoning with a fixed library of skill primitives. SayCan [32] combines a language model with an affordance-based value function to choose among predefined skills, allowing robots to execute long-horizon instructions in real-world settings. This yields interpretable plans but is inherently limited by the finite skill repertoire. OK-Robot [3] takes a systems approach by integrating open-vocabulary object recognition with navigation and grasping modules. It achieved about 58.5% success on zero-shot pick-and-place tasks in home environments with no additional training, significantly outperforming prior work. However, its scope is confined to pick-and-place scenarios. DovSG [2] leverages dynamic open-vocabulary 3D scene graphs to continuously update the robot's world model during execution. This enables a mobile manipulator to adapt its plan as the environment changes, supporting reliable long-term task execution in dynamic scenes. Despite their robustness and interpretability, modular frameworks are inherently bounded by predefined skill sets, limiting their capacity to address tasks requiring novel motor behaviors or tool use without manual augmentation.

III. METHOD

A. Problem Setup

We consider a mobile manipulation robot in everyday environments, tasked with executing long-horizon instructions involving both navigation and object manipulation. The robot must plan a sequence of actions $[a_1, \ldots, a_N]$ that carries it from an initial state to a goal state satisfying the given instruction. We formalize the objective as a constrained motion planning process

$$F : (I_{rgbd}, L, C) \rightarrow [a_1, \ldots, a_N], \tag{1}$$

where I_{rgbd} represents the RGB-D observations of the scene, L is the natural language instruction, and C denotes the set of constraints, including physical constraints of objects, environmental obstacles, and kinematic limits of the robot. The output is a sequence of actions aligned with the subtasks extracted from the instruction.

Rather than relying on a predefined library of low-level primitives whose limited expressiveness constrains manipulation generalization, we employ pre-trained coding models to leverage their commonsense knowledge and strong code generation ability for object parameterization and trajectory resolution. Since LLMs acquire extensive everyday knowledge during pre-training, they can infer task-relevant strategies. For example, they may reason that doors should be opened along an arc around their hinges, or that robot trajectories should avoid obstacles. Because our objective is to achieve functional completion rather than path optimality, we leverage modern code generation capabilities to synthesize executable programs that encode task-specific trajectories.

We propose **EmbodiedCoder**, a zero-shot framework that integrates LLMs with robotic systems to plan and execute such tasks without additional training.

B. System Overview

The proposed system consists of three main modules as shown in Fig. 2: (i) **Scene Understanding and Task Decomposition** This module captures RGB-D images of the environment, performs semantic grounding of objects, and decomposes long-horizon instructions into a set of subtasks. (ii) **EmbodiedCoder** This core module generates the robot's operation trajectories. Given a subtask and a semantic point cloud, EmbodiedCoder prompts the LLM to first parameterize objects by fitting point clouds to geometric primitives, and then to synthesize trajectories that respect object geometry, environmental constraints, and physical feasibility. (iii) **Motion Execution** This module carries out the planned motion by navigating to the target location and executing the manipulation with the robot's end effector. The comparison with other methods is shown in the Tab I. We next describe each component in detail.

TABLE I: Comparison of code-generation methods. *Skill Lib.* denotes reliance on a dedicated skill library; *Long-Term* indicates ability for long-horizon tasks.

Method	Train	Code Type	Skill Lib.	Long-Term
Code as Policies [6]	✗	motion planing	✓	✗
Code as Monitor [9]	✓	constraints	✓	✗
RoboCodeX [7]	✓	motion planing	✓	✗
VoxPoser [8]	✓	voxel value map	✗	✗
ReKep [30]	✓	constraints	✗	✗
Ours	✗	parameters & trajectory	✗	✓

C. Semantic Scene Understanding and Task Decomposition

To execute instructions in realistic environments, the robot must first perceive and understand its surroundings. The process begins with RGB-D image sequences I captured by the onboard cameras. We use VGGT [11] to recover 3D point clouds from RGB-D observations, ensuring real-world scale is preserved. A vision-language model (VLM) processes the images to obtain semantic descriptions of the scene, while SAM extracts segmentation masks for each object. These masks are projected onto the point cloud to yield semantically grounded point clouds for all detected objects. For compact storage, the semantic point clouds are compressed into a two-dimensional semantic map.

During task execution, the instruction is processed by a VLM in conjunction with the semantic map. Carefully designed prompts guide the model to decompose the long-horizon instruction into atomic subtasks such as navigation, parameterization, and manipulation. The semantic map serves as contextual input, reducing hallucinations and ensuring that task decomposition remains consistent with the actual physical environment. This mechanism allows the

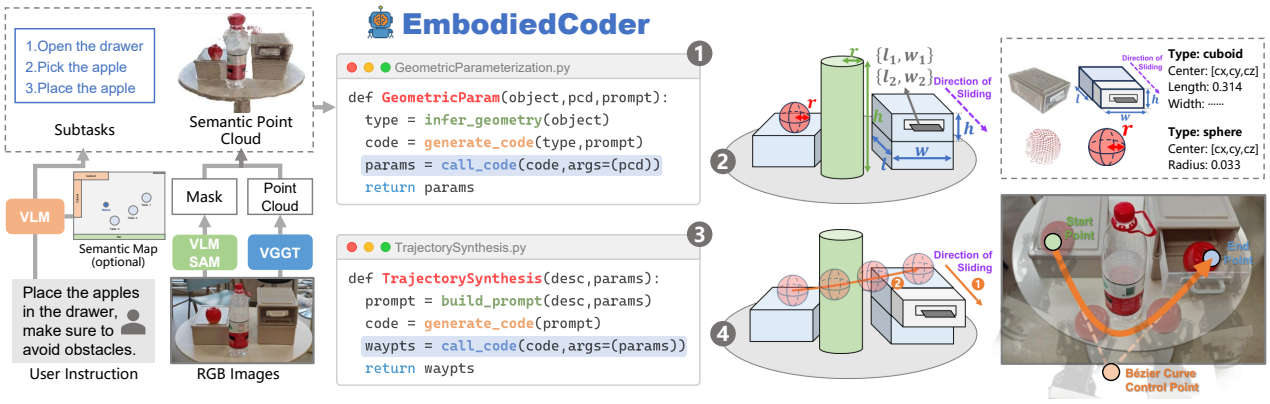

Fig. 2: **Overview of the proposed system pipeline.** The system consists of three modules: (i) ***Scene understanding and task decomposition***, which processes RGB-D images with VLM and VGGT to build semantic maps and decompose long-horizon instructions into subtasks; (ii) ***EmbodiedCoder***, which prompts an LLM to perform code-driven geometric parameterization of objects and trajectory synthesis under physical and environmental constraints; and (iii) ***Motion execution***, which samples waypoints from the synthesized trajectory and executes the manipulation with the robot arm.

system to infer implicit operations, such as opening a door, even when such actions are not explicitly stated in the instruction.

D. EmbodiedCoder

A central component of our framework is the EmbodiedCoder module, which leverages pre-trained large language models (LLMs) to transfer commonsense knowledge of manipulation into executable robot actions. This process is divided into two main stages:

(a) Code-Driven Geometric Parameterization In this step, the system aligns the point cloud of a target object to an appropriate geometric primitive such as a cuboid, cylinder, or sphere. For objects that cannot be accurately approximated by standard shapes, an irregular placeholder is used. Since LLMs cannot directly process raw point clouds, we prompt the model to generate code that fits parametric geometries to the observed data. For instance, fitting a cylinder requires estimating its radius, height, and center position, while fitting a cuboid requires determining the length, width, height, and centroid coordinates. This process transforms incomplete or occluded point clouds into compact geometric parameterizations, facilitating more robust reasoning. Fig. 3 presents an example of circular object parameterization.

```
def geoparamCircle(pts):
  main_pts = DBSCAN(pts) → largest_cluster
  plane, plane_pts = RANSAC(pts) → horizontal_plane
  pts_2d, trans, h = project_to_plane(plane_pts, plane)
  circA = boundary_fitting(pts_2d) → circle_params
  circB, inlier_ratio = RANSAC(pts_2d) → circle_params
  cx, cy, r = fuse_results(circA, circB, inlier_ratio)
  center = convert_to_world(cx, cy, h, trans)
  return {center, radius=r, height=h}
```

Fig. 3: LLM-generated cylinder parameterization code.

Moreover, parameterization enhances the subsequent trajectory synthesis by ensuring that critical features, such as symmetry axes or surface normals, are explicitly encoded. Our ablation studies confirm the necessity of this step for ensuring accuracy and stability in the overall process. The parameterization also implicitly encodes the affordances of the object, which are vital for functional manipulation. Beyond simply capturing the full structure of the object, we also parameterize the relevant parts that the robot will interact with, such as a drawer handle or a door knob. For irregular objects, such as a cloth, the system will parameterize the center of the cloth's point cloud, allowing the gripper to directly grasp the center and perform the manipulation. This process allows the robot to focus on the task-relevant parts of the object and ensures more effective and accurate interactions.

Formally, we define the parameterization process as

$$P : \{P_1, P_2, \ldots, P_n\} \rightarrow \{P'_1, P'_2, \ldots, P'_n\}, \tag{2}$$

where $P_1, P_2, \ldots, P_n$ represent the raw point clouds of the objects (such as a table and objects on it), and $P'_1, P'_2, \ldots, P'_n$ are the corresponding parameterized representations. Each object is parameterized differently depending on its shape, such as a circle and radius for a table, or center and dimensions for another object. This ensures that the manipulation trajectories generated later are well-defined and functionally relevant.

This parameterization step is crucial. Compared to directly reasoning from the point cloud, parameterizing the object leads to more accurate and stable trajectory generation, as validated in our ablation experiments. Fig. 4 shows some visualization results.

(b) Code-Driven Trajectory Synthesis After obtaining the geometric parameters of a task-specific object, we prompt the large language model (LLM) to generate code for synthesizing a trajectory $\tau = [\tau_1, \tau_2, \ldots, \tau_n]$ that aligns with the object's functional properties and the task requirements.

The trajectory generation process takes multiple constraints into account. For example, in the case of opening a door, the physical constraint is that the door must rotate around its hinge axis; the environmental constraint is whether the door should be pushed or pulled; and the robotic hardware constraint includes ensuring that, when rotating the door, if the robot does not release its grip, the door will block

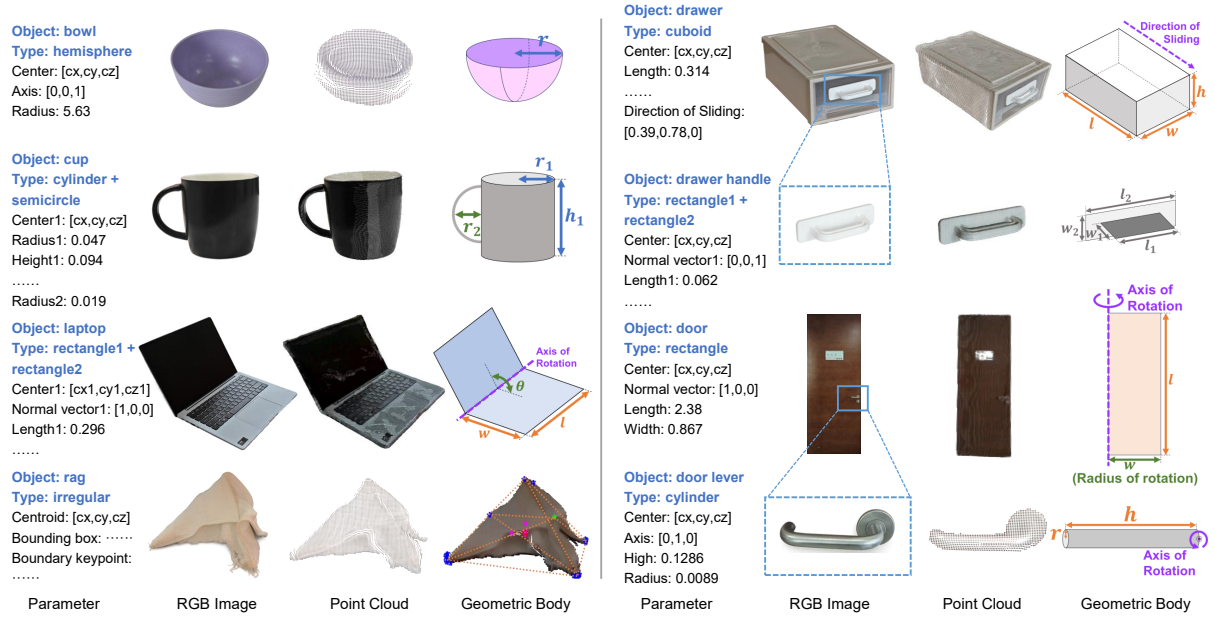

Fig. 4: Examples of Parametric Fitting for Common Objects.

the robot's arm once it opens past a certain angle, preventing further movement. Additionally, the door's opening gap must be wide enough to allow the robot to pass through, not just a small opening. This requires taking the robot's dimensions into consideration when planning the trajectory. In environments with obstacles, the system parametrizes the obstacles and incorporates them into the trajectory planning process. This ensures that the generated trajectory avoids these obstacles, adapting the robot's movements to navigate around them effectively. LLM generates the appropriate trajectory form based on the task requirements and the object's parameters, enabling the system to generalize across various tasks without manual design for each individual scenario.

Once the trajectory form is determined, it is expressed as a parametric curve, such as a line, arc, or Bézier curve, with the necessary parameters for the specific task. We formalize this process as follows

$$T : \{P, C\} \rightarrow \{\tau_1, \tau_2, \ldots, \tau_n\}, \tag{3}$$

where T represents the continuous trajectory function, P is the parameterized object geometry, and C represents the relevant constraints. The output is a series of waypoints, $\{\tau_1, \tau_2, \ldots, \tau_n\}$, which guide the robot's motion during task execution. Fig. 5 presents a code where the large coding model fits a Bézier curve.

```
def trajsynBezier(A, B, num, curvature):
  if A == B: return repeat(A, num)
  dx, dy, dz = B - A; dist = norm(B - A)
  if curvature == 0: return linear_interp(A, B, num)
  unit_dir = normalize(B - A)
  perp = find_perpendicular(unit_dir) → bias_upward
  P1 = A + (B - A)/3 + perp * dist * curvature
  P2 = B - (B - A)/3 + perp * dist * curvature
  waypoints = [cubic_bezier(A, P1, P2, B, t) for t in
linspace(0,1,num)]
  return waypoints
```

Fig. 5: LLM-generated code for Bézier curve fitting.

Using code generation for trajectory synthesis, instead of directly calculating the trajectory from the geometric parameters, improves the stability and interpretability of the process. This method provides the flexibility to dynamically adjust the trajectory in response to changing task conditions or environments, ensuring that the robot can generalize to new, previously unseen tasks.

E. Motion Execution

After scene understanding and task decomposition, the robot executes each subtask sequentially. For each subtask, EmbodiedCoder provides a trajectory τ. The robot samples a set of waypoints from τ, navigates to the relevant position, and executes the manipulation with its arm and gripper.

For tasks involving familiar object types or recurring subtasks, the system reuses previously generated code, reducing latency. For unseen objects or novel tasks, EmbodiedCoder is invoked to parameterize the object and synthesize new trajectories. This design achieves a balance between efficiency for known cases and generalization for open-world scenarios.

IV. EXPERIMENT

In this section, we evaluate the effectiveness of our proposed method in real-world scenarios. Our experiments focus on long-horizon tasks that involve complex and diverse subtasks beyond simple pick-and-place operations. We also compare our method with commonly used Vision-Language-Action (VLA) approaches on out-of-distribution (OOD) tasks to assess generalization. Finally, we conduct ablation studies to analyze the contributions of different components of our framework.

A. Experimental Setup

Our experiments are conducted on an AgileX Cobot S Kit mobile robotic platform equipped with a RealSense D455 RGB-D camera. The vision-language model (VLM) used in

TABLE II: **Success rates of long-horizon tasks and their subtasks**, averaged over 20 trials. S/U indicate performance in seen and zero-shot tasks, respectively. Compared with DovSG [2], our approach consistently yields higher success rates and is capable of completing operations such as door opening, which DovSG [2] cannot handle.

Bring the water bottle from the table by the door and pour it into the bowl.				
	Open Door(%)	Pick Bottle(%)	Pour Water(%)	Long Term(%)
	S/U	S/U	S/U	S/U
DovSG [2]	✗/✗	75/75	✗/✗	✗/✗
Ours	60/50	85/80	70/60	35/25
Take the apples from the white box and place them on the cutting board.				
	Open Box(%)	Pick Apple(%)	Place Apple(%)	Long Term(%)
	S/U	S/U	S/U	S/U
DovSG [2]	✗/✗	50/50	90/90	✗/✗
Ours	55/50	80/70	90/90	40/30
Place the apples in the drawer and make sure to avoid obstacles.				
	Open Drawer(%)	Pick Apple(%)	Place Apple(%)	Long Term(%)
	S/U	S/U	S/U	S/U
DovSG [2]	✗/✗	90/90	80/80	✗/✗
Ours	85/80	95/90	90/85	70/65
Move the tennis ball from the first table to the pink bowl on the third table.				
	Pick ball(%)	Place ball(%)	-	Long Term(%)
	S/U	S/U	-	S/U
DovSG [2]	85/85	90/90	-	75/75
Ours	95/95	95/95	-	90/90
Get a cloth and wipe the stains off the table.				
	Pick Cloth(%)	Wipe Table(%)	-	Long Term(%)
	S/U	S/U	-	S/U
DovSG [2]	60/60	✗/✗	-	✗/✗
Ours	85/85	75/70	-	65/60

our experiments is Qwen-2.5-VL [33] 7B, and the code-generation language model (LLM) is Claude Code [34]. In our ablation studies, we also explore how substituting different VLMs or LLMs affects overall performance.

B. Long-Horizon Task Evaluation

To evaluate performance on complex daily tasks, we design five long-horizon tasks that involve a combination of mobile navigation and dexterous manipulation: *1) Bring the water bottle from the table by the door and pour it into the bowl. 2) Pick up apples from the white box and place them on the cutting board. 3) Move a tennis ball from the first table to a pink bowl located on the third table. 4) Store the apples inside a drawer while avoiding surrounding obstacles. 5) Retrieve a cleaning cloth and wipe stains off the table surface.* These tasks require precise end-effector control, mobile base navigation, and sequential decision-making. Each experiment is repeated 20 times to ensure statistical reliability.

We compare our method with DovSG [2], as other code-generation-based methods such as Rekep [30] and VoxPoser [8] are not capable of executing long-horizon tasks. Table II summarizes the results. We define two evaluation settings: **Unseen**: The robot executes tasks without prior knowledge of the environment, objects, or task type. **Seen**: The robot has previously executed the same task and retains effective code generated by the model, which is reused directly without re-invoking the LLM.

Our method achieves similar performance between seen and unseen conditions, demonstrating strong generalization. In contrast, DovSG [2] is limited to simple pick-and-place actions and fails on tasks requiring more complex operations, such as opening doors or drawers. However, our method shows a lower success rate on the Open Door task. This is primarily due to two factors. First, when navigating close to the door, the limited field of view of the camera sometimes fails to capture the entire door, resulting in incomplete point clouds. This leads to errors in parameter estimation, particularly in computing the rotation radius, which propagates to trajectory generation and causes execution failure. Second, successful door-opening requires a two-stage motion: initially pulling the door slightly ajar, then moving around to push it fully open. In practice, we observed that even with detailed prompts, the LLM often outputs a single continuous arc trajectory rather than a two-phase plan, resulting in failure. These findings highlight the challenges of handling multi-stage tasks under current model capabilities.

C. Simple Task Evaluation

Since most existing VLA methods and code-generation approaches can only handle single-step tasks, we also compare performance on simpler tasks.

TABLE III: **Quantitative results on OOD tasks compared with VLA models.** We report success rates(%) over 20 trials, with the results of the VLA models on their original papers.

Task (%)	RT-1	RT-2	Octo	OpenVLA	RDT	Ours
Pick Pepsi Can	60	100	0	80	-	100
Pick Banana	100	100	60	100	-	80
Pick Green Cup	20	100	0	100	-	100
Place Apple on Plate	0	80	0	80	-	95
Place Banana in Pan	0	40	0	80	-	80
Pour Water	-	-	13	0	63	80
Move Orange near Brown Chip Bag	20	100	40	100	-	90

Table III reports success rates on OOD tasks for both VLA methods and our approach. Our method achieves performance comparable to or better than trained VLA models **without any additional training.** Notably, for tasks involving delicate operations, such as pouring water, our success rate significantly exceeds that of other methods. This demonstrates the strong and stable generalization ability of our framework, which stems from effectively leveraging the commonsense knowledge embedded in large models and grounding it in physical 3D operations through our parameterization strategy. By directly linking semantic reasoning to the physical world, we avoid the information loss associated with conventional feature extraction pipelines.

Additionally, Table IV presents comparisons with Rekep [30], Voxposer [8], and Code-as-Monitor [9], which adopt code-generation approaches for deriving actions. Our method consistently outperforms these approaches. Unlike methods that rely on LLM-generated constraint points for trajectory optimization, our parameterization approach serves as a direct bridge between LLMs and physical execution.

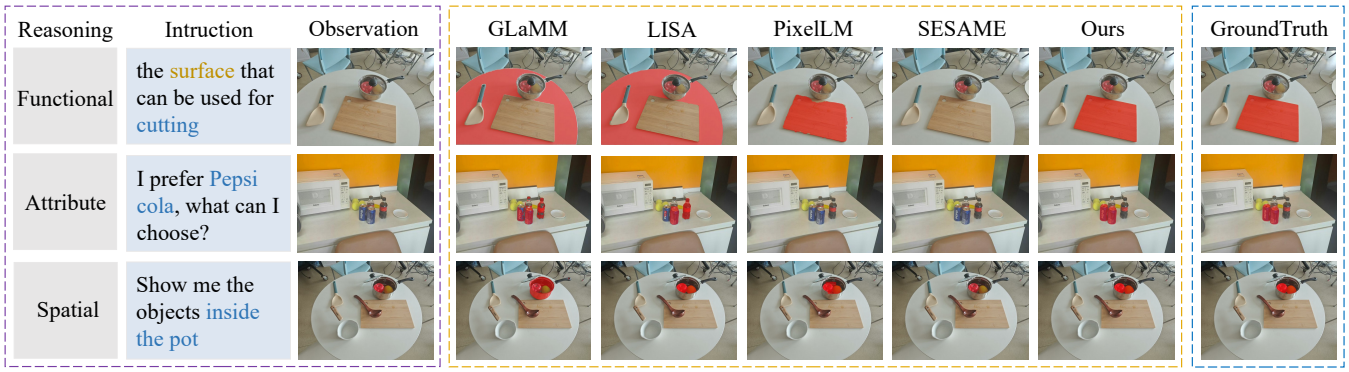

Fig. 6: Comparison of different large models on semantic grounding across functional, attribute, and spatial reasoning tasks.

TABLE IV: **Comparison with code-generation methods.** Results are reported from original papers.

Task	Pour Tea	Recycle Can	Stow Book
ReKep [30]	80	80	60
VoxPoser [8]	0	30	0
Code-as-Monitor [9]	50	-	70
Ours	80	100	80

This allows the model to incorporate both physical and environmental constraints into trajectory planning, resulting in more reliable and adaptable performance.

D. Ablation Study

We conduct ablation studies to validate the design and effectiveness of each module in our framework.

1) Effect of Object Morphology: We test grasping performance on a diverse set of objects, including bottles, oranges, and plastic bags, and compare our method with AnyGrasp [35].

TABLE V: Comparison with AnyGrasp [35] on grasp success rates(%) across different objects over 20 trials.

Task (%)	Bottle	Apple	Orange	Banana	Pepsi Can	Plastic Bag	Green Cup
Anygrasp [35]	95	70	95	80	40	90	60
Ours	100	95	100	90	75	100	80

As shown in Table V, our method achieves significantly higher success rates. By parameterizing objects to obtain accurate attributes such as center position and radius, we can align grasp poses with object morphology. For instance, spherical objects are grasped with gripper poses aligned along the radial direction, resulting in stable grasps. In contrast, AnyGrasp [35] outputs grasp poses without considering gripper constraints, leading to infeasible or unreachable configurations and reduced performance.

2) Robustness of Semantic Grounding: We evaluate the robustness of semantic grounding across three reasoning categories in Fig. 7: *1) Functional reasoning*, which identifies areas suitable for specific actions such as cutting; *2) Attribute reasoning*, which distinguishes between objects with similar appearances, for example different soda brands; and *3) Spatial reasoning*, which captures the spatial relationships between objects. Given image inputs and task instructions, the VLM generates corresponding masks. Our method consistently achieves the highest accuracy across all reasoning categories, demonstrating strong semantic grounding capability.

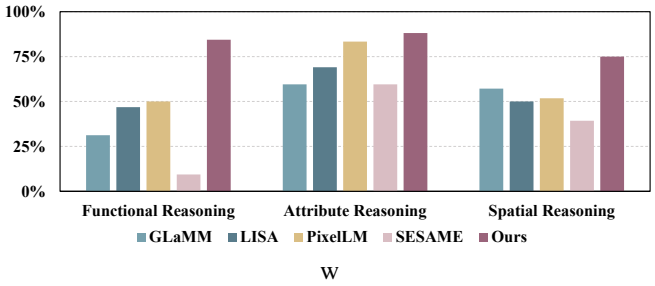

Fig. 7: Ablation study comparing the performance of five different segmentation models on functional reasoning, attribute reasoning, and spatial reasoning tasks.

In addition, we investigated the effect of providing the VLM with a 2D semantic map as input for subtask decomposition. As illustrated in Fig. VI, when supplied with a 2D map, the VLM is more likely to generate feasible decomposition strategies.

TABLE VI: Comparison of five different VLMs on subtask decomposition success rates(%) with and w/o map.

Models	PaliGemma	Qwen-3B	Qwen-7B	GPT-5	Gemini2.5 Pro
With Map	0	80	88	88	56
W/O Map	0	72	72	64	20

For example, a cross-room task may require navigating to the doorway followed by executing a door-opening action. Experimental results show that incorporating a 2D semantic map helps align subtask decomposition with the actual scene, thereby reducing planning hallucinations.

3) Impact of LLM Coding Capabilities: We investigate how the coding ability of different LLMs affects our method. Given the same prompts, we ask each model to generate code for object parameterization. We select several top-performing LLMs based on recent coding benchmarks. As shown in Fig. 8, Claude Code exhibits the highest valid rate, outperforming all other models. These results confirm that our pipeline is only feasible with the current generation of powerful LLMs. Earlier models lacked the coding

w

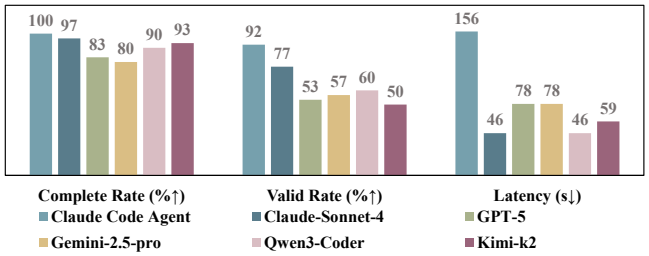

Fig. 8: Ablation study evaluating the ability of five coding models to fit the geometric parameters of objects.

proficiency required to support this approach.

V. CONCLUSION AND LIMITATIONS

We introduced EmbodiedCoder, a training-free framework that combines large language and coding models with structured geometric object representations to enable open-world mobile manipulation. By grounding semantic knowledge into parameterized affordances and generating executable trajectory code, the system allows robots to perform long-horizon, contact-rich tasks without predefined primitives or additional training. Experiments on real robots demonstrate strong generalization to unseen objects and environments, advancing the integration of high-level reasoning with low-level control. Nonetheless, several limitations remain. First, task success is highly sensitive to the quality of code generated by large models, and errors in logic or syntax can significantly reduce reliability. Second, the code synthesis process introduces latency, which may limit responsiveness in real-time applications. Addressing these issues will enhance robustness and scalability, moving closer to practical deployment of versatile robot intelligence.

REFERENCES

[1] L. P. Kaelbling, "The foundation of efficient robot learning," *Science*, vol. 369, no. 6506, pp. 915–916, 2020.

[2] Z. Yan, S. Li, Z. Wang, L. Wu, H. Wang, J. Zhu, L. Chen, and J. Liu, "Dynamic open-vocabulary 3d scene graphs for long-term language-guided mobile manipulation," *IEEE Robotics and Automation Letters*, 2025.

[3] P. Liu, Y. Orru, C. Paxton, N. M. M. Shafiullah, and L. Pinto, "Ok-robot: What really matters in integrating open-knowledge models for robotics," *arXiv preprint arXiv:2401.12202*, 2024.

[4] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," in *International conference on machine learning*. PmLR, 2021, pp. 8748–8763.

[5] J. Li, D. Li, S. Savarese, and S. Hoi, "Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models," in *International conference on machine learning*. PMLR, 2023, pp. 19 730–19 742.

[6] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, "Code as policies: Language model programs for embodied control," *ICRA*, 2023.

[7] Y. Mu, J. Chen, Q. Zhang, S. Chen, Q. Yu, C. Ge, R. Chen, Z. Liang, M. Hu, C. Tao *et al.*, "Robocodex: Multimodal code generation for robotic behavior synthesis," *arXiv*, 2024.

[8] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei, "Voxposer: Composable 3d value maps for robotic manipulation with language models," *arXiv preprint arXiv:2307.05973*, 2023.

[9] E. Zhou, Q. Su, C. Chi, Z. Zhang, Z. Wang, T. Huang, L. Sheng, and H. Wang, "Code-as-monitor: Constraint-aware visual programming for reactive and proactive robotic failure detection," in *Proceedings of the Computer Vision and Pattern Recognition Conference*, 2025, pp. 6919–6929.

[10] D. Shah, B. Osiński, B. Ichter, Y. Hu, A. Saddiqi *et al.*, "Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action," in *Conference on Robot Learning (CoRL)*, 2022.

[11] J. Wang, M. Chen, N. Karaev, A. Vedaldi, C. Rupprecht, and D. Novotny, "Vggt: Visual geometry grounded transformer," in *Proceedings of the Computer Vision and Pattern Recognition Conference*, 2025, pp. 5294–5306.

[12] M. Zawalski, W. Chen, K. Pertsch, C. Finn, and S. Levine, "Robotic control via embodied chain-of-thought reasoning," *arXiv preprint arXiv:2407.08693*, 2025.

[13] J. Hu, R. Hendrix, A. Farhadi, A. Kembhavi, R. Martín-Martín *et al.*, "Flare: Achieving masterful and adaptive robot policies with large-scale reinforcement learning fine-tuning," *arXiv preprint arXiv:2409.16578*, 2024.

[14] R. Mon-Williams, G. Li, R. Long, W. Du, and C. G. Lucas, "Embodied large language models enable robots to complete complex tasks in unpredictable environments," *Nature Machine Intelligence*, vol. 7, no. 7, pp. 592–601, 2025.

[15] P. Ding, H. Zhao, W. Zhang, W. Song, S. Huang *et al.*, "Quar-vla: Vision-language-action model for quadruped robots," in *European Conference on Computer Vision (ECCV)*, 2024, pp. 352–367.

[16] S. Reed, K. Zolna, E. Parisotto, S. Colmenarejo, A. Novikov *et al.*, "A generalist agent," *arXiv preprint arXiv:2205.06175*, 2022.

[17] A. Brohan, N. Brown, J. Carbajal, and et al., "Rt-2: Vision-language-action models transfer web knowledge to robotic control," 2023.

[18] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu *et al.*, "Rt-1: Robotics transformer for real-world control at scale," *arXiv*, 2022.

[19] J. Yang, R. Tan, Q. Wu, R. Zheng, Y. Liang *et al.*, "Magma: A foundation model for multimodal ai agents," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

[20] J. Liu, P. An, Z. Liu, R. Zhang, C. Gu *et al.*, "Robomamba: Efficient vision-language-action model for robotic reasoning and manipulation," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 37, pp. 40 085–40 110, 2024.

[21] G. Kang, J. Kim, K. Shim, J. Lee, and B.-T. Zhang, "Clip-rt: Learning language-conditioned robotic policies from natural language supervision," in *Robotics: Science and Systems (RSS)*, 2025.

[22] A. Padalkar, A. Pooley, A. Jain, K. Hausman, A. Rai *et al.*, "Open x-embodiment: Robotic learning datasets and rt-x models," in *Conference on Robot Learning (CoRL)*, 2023.

[23] J. Wen, Y. Zhu, J. Li, M. Zhu, Z. Tang, K. Wu, Z. Xu, N. Liu, R. Cheng, C. Shen *et al.*, "Tinyvla: Towards fast, data-efficient vision-language-action models for robotic manipulation," *IEEE Robotics and Automation Letters*, 2025.

[24] J. Bjorck, F. Castañeda, N. Cherniadev, X. Da, R. Ding, L. Fan, Y. Fang, D. Fox, F. Hu, S. Huang *et al.*, "Gr00t n1: An open foundation model for generalist humanoid robots," *arXiv preprint arXiv:2503.14734*, 2025.

[25] A. Stone, T. Xiao, Y. Lu, K. Gopalakrishnan, K. Hausman *et al.*, "Open-world object manipulation using pre-trained vision-language models," in *Conference on Robot Learning (CoRL)*, 2023.

[26] T. Xiao, H. Chan, P. Sermanet, A. Wahid, S. Levine *et al.*, "Robotic skill acquisition via instruction augmentation with vision-language models," in *Robotics: Science and Systems (RSS)*, 2023.

[27] Y. Jiang, A. Gupta, Z. Zhang, G. Wang, L. Fei-Fei *et al.*, "Vima: General robot manipulation with multimodal prompts," in *International Conference on Machine Learning (ICML)*, 2023.

[28] D. Driess, F. Xia, M. S. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter *et al.*, "Palm-e: An embodied multimodal language model," in *International Conference on Machine Learning (ICML)*, 2023.

[29] S. Nair, S. Khandelwal, M. Danielczuk, K. Xu, P. Florence *et al.*, "R3m: A universal visual representation for robot manipulation," in *Conference on Robot Learning (CoRL)*, 2022.

[30] W. Huang, C. Wang, Y. Li, R. Zhang, and L. Fei-Fei, "Rekep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation," *arXiv*, 2024.

[31] W. Huang, F. Xia, D. Shah, A. Zeng, P. Florence *et al.*, "Grounded decoding: Guiding text generation with grounded models for robot control," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[32] M. Ahn, A. Brohan, N. Brown, and et al., "Do as i can, not as i say: Grounding language in robotic affordances," 2022.

[33] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge,

Y. Han, F. Huang *et al.*, "Qwen technical report," *arXiv preprint arXiv:2309.16609*, 2023.
[34] Anthropic, "Claude sonnet 4 technical documentation," https://www.anthropic.com, 2025.
[35] H.-S. Fang, C. Wang, H. Fang, and et al., "Anygrasp: Robust and efficient grasp perception in spatial and temporal domains," 2023.