2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

EmbodiedCoder: Parameterized Embodied Mobile Manipulation via Modern Coding Model

Zefu Lin[2,3] **Rongxu Cui**[5] **Chen Hanning**[1] **Xiangyu Wang**[1,2,3] **Junjia Xu**[5] **Xiaojuan Jin**[2,3]
Chen Wenbo[2,3] **Hui Zhou**[6] **Lue Fan**[2,3] ✉ **Wenling Li**[5] **Zhaoxiang Zhang**[1,2,3,4] ✉

[1] University of Chinese Academy of Sciences (UCAS)
[2] Institute of Automation, Chinese Academy of Sciences (CASIA)
[3] New Laboratory of Pattern Recognition (NLPR)
[4] State Key Laboratory of Multimodal Artificial Intelligence Systems (MAIS)
[5] Beihang University [6] Chinese University of Hong Kong
{linzefu2022, lue.fan}@ia.ac.cn

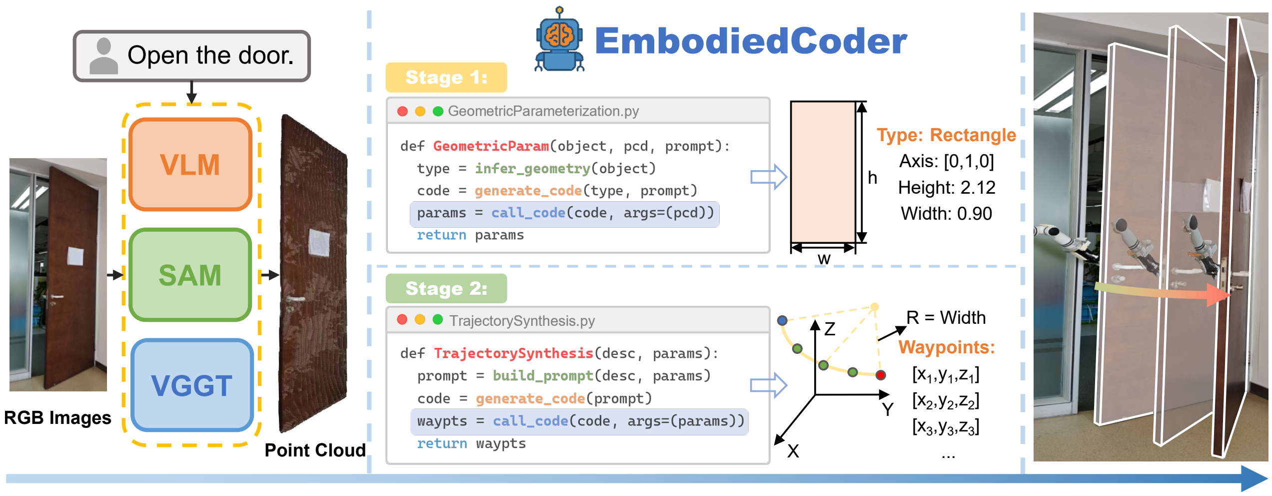

Fig. 1: **EmbodiedCoder** employs code generation to bridge perception and manipulation by parameterizing objects and synthesizing task-specific trajectories. The figure shows a subtask ***Open the door*** derived from a long-term instruction. Through code-driven geometric parameterization, the door is represented as a parametric model with a hinge axis, and the system generates code that synthesizes a semicircular trajectory consistent with this geometry. The robotic arm then executes the opening motion by following waypoints sampled from the generated trajectory, demonstrating how coding enables functional manipulation without additional training.

Abstract— **Recent advances in control robot methods, from end-to-end vision-language-action frameworks to modular systems with predefined primitives, have advanced robots' ability to follow natural language instructions. Nonetheless, many approaches still struggle to scale to diverse environments, as they often rely on large annotated datasets and offer limited interpretability. In this work, we introduce *EmbodiedCoder*, a training-free framework for open-world mobile robot manipulation that leverages coding models to directly generate executable robot trajectories. By grounding high-level instructions in code, *EmbodiedCoder* enables flexible object geometry parameterization and manipulation trajectory synthesis without additional data collection or fine-tuning. This coding-based paradigm provides a transparent and generalizable way to connect perception with manipulation. Experiments on real mobile robots show that *EmbodiedCoder* achieves robust performance across diverse long-term tasks and generalizes effectively to novel objects and environments. Our results demonstrate an interpretable approach for bridging high-level reasoning and low-level control, moving beyond fixed primitives toward versatile robot intelligence. See the project page at https://embodiedcoder.github.io/EmbodiedCoder/.**

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 manipulation primitives. Many real-world tasks, such as opening doors or drawers, require nuanced interactions that cannot be reduced to a finite set of predefined primitives.

Recent work has attempted to extend beyond this primitive-based architecture by generating executable code for manipulation. 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 propose **EmbodiedCoder**, a code-driven framework for open-world mobile robot manipulation. Unlike traditional training-intensive approaches, our method leverages the expressive power of coding models to generate executable code that directly encodes manipulation strategies. This design transforms high-level instructions into programmatic representations of geometric parameterization and trajectory synthesis. By grounding the reasoning process in code, the system benefits from both interpretability and flexibility, enabling robots to adapt to novel objects and environments without additional training or fine-tuning.

At the core of our framework, code serves as the medium that bridges perception and manipulation. The process begins with scene understanding, where VGGT [11] and a vision-language model capture RGB-D observations and ground semantic information into 3D point representations. Based on this input, **EmbodiedCoder** prompts the coding models to generate code for two critical stages. First, in *code-driven geometric parameterization*, the system fits point clouds of task-relevant objects to geometry parametric primitives that encode functional affordances, such as approximating a drawer as a cuboid with a pulling axis. Second, in *code-driven trajectory synthesis*, our method produces programmatic descriptions of feasible motion trajectories that satisfy physical, environmental, and task-specific constraints. The trajectories are first represented as parameterized curves, from which discrete waypoints are sampled and subsequently executed by the robot. By this approach, the system not only achieves robust performance in novel environments but also provides a transparent and generalizable mechanism for linking perception with real-world manipulation.

In summary, our method 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 coding models with embodied agents, enabling complex long-term manipulations in real-world environments.
- We propose a novel method for parameterizing objects into functional geometric abstractions, allowing pretrained knowledge to be grounded into executable trajectories for sophisticated.
- 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]. RT-2 [13], for example, extends multi-task policies with web-scale vision-language pretraining to enhance zero-shot understanding. However, such models require massive robot demonstration datasets [14] and often fail to generalize under distribution shifts in lighting, appearance, or object variation. Recent efforts improve efficiency [15], [**?**]. TinyVLA [16] achieves faster inference and greater data efficiency with a compact design, while GR00T N1 [17] employs dual-system reasoning for humanoid control. These advances broaden VLA capabilities, but even state-of-the-art policies still depend on curated datasets and lack systematic generalization in novel scenarios [18], [19], [20].

B. LLM-Driven Trajectory and Code Generation

Another line of research employs large language models (LLMs) as high-level planners or code generators for robotics [10], [21]. Code-as-Policies [6] showed that LLMs can compose robot-executable code from natural language, but flexibility is limited by fixed APIs. VoxPoser [8] generates 3D affordance maps for motion planning and zero-shot execution of tasks like drawer opening, though only for relatively simple manipulations. ReKep [22] detects object keypoints and prompts LLMs to produce relational cost functions, enabling multi-stage manipulation without task-specific training, though tasks poorly described by sparse keypoints remain difficult. RoboCodeX [7] decomposes high-level instructions into object-centric units with affordances and safety constraints, generating structured code that achieves strong results across simulation and real robots. Yet, its reliance on curated multimodal data restricts adaptability. Complementary to planning, Code-as-Monitor [9] compiles natural language constraints into runtime monitors for failure detection and recovery. In summary, these LLM-driven methods [23] illustrate the potential of language models for behavior composition and robustness, but they still struggle with contact-rich interactions and depend heavily on prior data or predefined skills.

C. Modular Systems with Predefined Skills

A third strategy integrates high-level reasoning with a fixed library of skills. SayCan [24] combines language models with affordance-based value functions to choose among predefined behaviors, producing interpretable long-term plans but constrained by a finite repertoire. OK-Robot [3] integrates open-vocabulary object recognition with grasping and navigation, achieving strong zero-shot pick-and-place performance without extra training. DovSG [2] leverages dynamic open-vocabulary 3D scene graphs to update the world model during execution, supporting adaptive planning in changing environments. Despite robustness and interpretability, these modular systems remain fundamentally bounded by predefined skills, limiting their ability to handle novel behaviors or tool use without manual extension.

In contrast, we introduces **EmbodiedCoder**, a training-free, code-driven framework that bridges perception and action by generating executable programs for geometric parameterization and trajectory synthesis, thereby overcoming the limitations of data-hungry policies, predefined skill libraries, and narrow keypoint-based reasoning.

III. METHOD

A. Problem Setup

We consider a mobile manipulation robot in everyday environments, tasked with executing complex instructions involving both navigation and 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 coding models to leverage their commonsense knowledge and strong code generation ability for object parameterization and trajectory resolution. These models can qualitatively determine how objects should be parameterized and which trajectories are appropriate for accomplishing a given task, and they further provide quantitative solutions that transform these insights into executable motion plans. Building on this capability, we propose **EmbodiedCoder**, a zero-shot framework that integrates coding models 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**. The module takes RGB-D images and task instructions as input, performs semantic grounding of objects, decomposes the instruction into subtasks, and outputs task-related semantic point clouds. (ii) **EmbodiedCoder**. This core module generates the robot's operation trajectories. Given a subtask and a semantic point cloud, EmbodiedCoder prompts the coding models to first parameterize objects by fitting point clouds to geometric primitives, and then to synthesize trajectories that conform to object geometry. Reasoning over object geometry allows the system to capture contact surfaces, spatial relations, and kinematic feasibility, which in turn ensures that the resulting trajectories satisfy environmental and physical constraints. (iii) **Motion Execution**. This module carries out the planned motion by navigating to the target location and executing task-oriented manipulation. The comparison with other methods is shown in the Table I. We next describe each component in detail.

TABLE I: Comparison of code-generation methods.

Method	Training-Free	Code Type	Skill Library	Long-term Task
Code as Policies [6]	✓	motion planing	✓	
Code as Monitor [9]		constraints	✓	
RoboCodeX [7]		motion planing	✓	
VoxPoser [8]		voxel value map		
ReKep [22]		constraints		
CodeDiffuser [25]		perception		
RoboScript [26]	✓	motion planing	✓	
Ours	✓	geometry & trajectory		✓

C. Semantic Scene Understanding and Task Decomposition

1) Semantic Mapping for Scene Understanding: In the preparation phase, the robot processes a sequence of RGB images using VGGT [11] to reconstruct a dense point cloud of the entire scene. Relying solely on the RGB-D camera is insufficient due to depth noise and range limitations, so we align its depth maps with the VGGT reconstruction to achieve a reliable metric-scale point cloud. A vision-language model (VLM) provides semantic grounding of the scene in the form of bounding boxes, which are then passed to SAM [27] to generate 2D semantic masks of all objects. These masks are projected onto the reconstructed point cloud, resulting in a semantic point cloud map. To facilitate subsequent task planning, this map is converted into a bird's-eye-view semantic representation stored in 2D image form.

2) Task Decomposition and Object-centric Semantic Understanding: In this stage, the system takes RGB-D observations and a complete task instruction as input. The instruction, together with the semantic map, is processed by the VLM to decompose the task into a sequence of subtasks, each associated with specific objects. For instance, the system may deduce that interacting with a door is a prerequisite before moving into an adjacent room. For each subtask, the current RGB observation is fed into the VLM

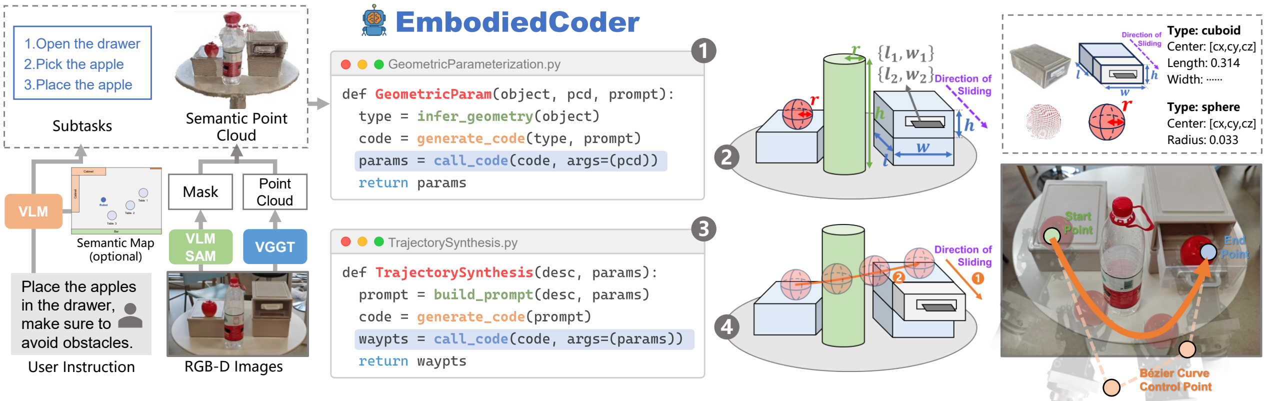

Fig. 2: **Overview of the proposed system pipeline.** The system consists of three modules: (i) ***Scene understanding and task decomposition***, which processes RGB-D images with VLM and VGGT to build semantic maps and decompose instructions into subtasks; (ii) ***EmbodiedCoder***, which prompts an coding model 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.

to identify and ground the relevant objects. In addition, the VLM infers the most suitable geometric shape of the objects and performs functional reasoning to determine which object parts must be manipulated to achieve the intended function. The grounding results are passed to SAM [27] to generate 2D semantic masks of these objects. The masks are then projected into the point cloud, producing a semantic point cloud representation that captures only the objects of interest for the given subtask. This object-centric representation is stored and later consumed by EmbodiedCoder for geometric parameterization and trajectory synthesis.

D. *EmbodiedCoder*

A central component of our framework is the Embodied-Coder 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 takes the task-relevant objects and their point clouds, and prompts the coding model to generate code that fits them to the geometric primitives identified in *Task Decomposition and Object-centric Semantic Understanding*. 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. For deformable objects, extreme points are selected to construct bounding envelopes instead of rigid parameterizations. Fig. 3 presents the code-driven parameterization of a door.

Extracting geometric parameters for all task-relevant objects provides a structured basis for trajectory planning, enabling the system to satisfy constraints such as obstacle avoidance and motion feasibility. Consequently, we parameterize not only the overall structure but also the functional components that support interaction, such as drawer handles or door knobs. This process directs the robot's attention to task-relevant regions of the object and leads to more accurate and effective manipulation. After this step, unstructured point cloud data are converted into structured representations. For example, the point cloud of an apple is reduced to a sphere defined by its center and radius, while a door is represented as a combination of a cuboid for the panel, a rotational axis, and a cylinder for the handle. Such compact parameterizations make the subsequent generation of task-specific trajectory code feasible. Fig. 4 shows some visualization results.

(b) Code-driven Trajectory Synthesis. After obtaining the geometric parameters of a task-specific object, we prompt

```
Prompt: ... The current subtask of the robot is {open the door}.
Please use a {rectangle} to fit the point cloud of the {door},
with the fitting method being {the prompt for the specific
rectangle fitting method}. Use a {cylinder} to fit the point
cloud of the {door handle}, with the fitting method being {the
prompts for the specific cylinder fitting method}. At the same
time, compute the functional parameters related to the task.
The input and output are respectivel ...

def geoparamDoor(door_pcd, handle_pcd):
  # 1.Door geometry extraction
  # denoise & keep main cluster
  door_main = DBSCAN(door_pcd)
  # door plane orientation from PCA
  door_ctr, door_norm, dir_w, dir_h = PCA_plane(door_main)
  door_2d = project_to_2d(door_main, door_ctr, dir_w, dir_h)
  # Fit the rectangle using quantiles
  rect2d = quantile_rectangle(door_2d)
  door_geom = project_to_3d(rect2d, door_ctr, dir_w, dir_h)
  # 2.Handle geometry extraction
  handle_main = DBSCAN(handle_pcd)
  # principal axis of the handle
  h_axis = PCA_axis(handle_main)
  # circle fit on handle cross-section
  h_ctr, h_r = LSQ_circle_section(handle_main, h_axis)
  handle_geom = fit_cylinder(h_ctr, h_r, h_axis, handle_main)
  # 3.Functional parameters of the door
  # locate hinge opposite to handle
  hinge_axis = infer_hinge(door_geom, handle_geom, door_norm,
dir_h)
  rot_radius = dist(handle_geom.center, hinge_axis)}
  return {door_geom, handle_geom, hinge_axis, rotation_radius}
```

Fig. 3: **Example of door parameterization.** The task-relevant content in prompt is from *Task Decomposition and Object-centric Semantic Understanding*.

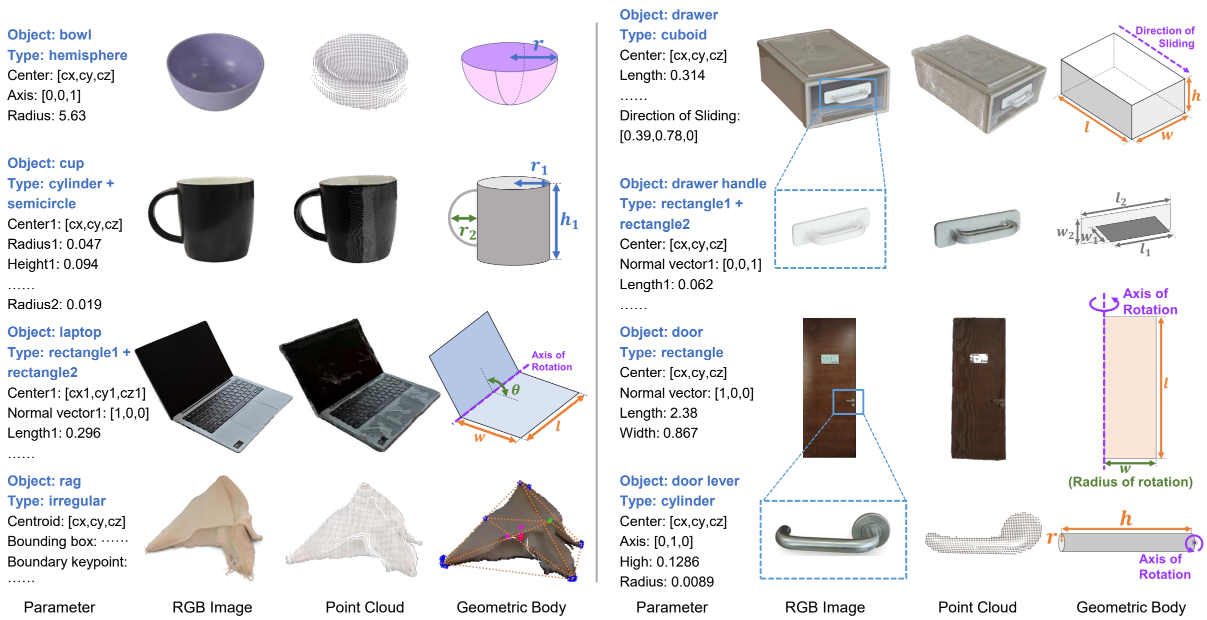

Fig. 4: Examples of Parameterization Result for Common Objects.

the coding model to generate code for synthesizing a trajectory that aligns with the object's functional properties and the task requirements.

The trajectory generation process accounts for multiple constraints, which are inferred directly by the coding model. 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 concerns whether the door should be pushed or pulled; and the hardware constraint of the robot arises from the limited range of motion of its joints. 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. Coding model generates the appropriate trajectory generation code 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. Fig. 5 shows the code generated from the prompt for placing an apple while avoiding obstacles. Using code generation for trajectory synthesis, instead of relying on a VLA that directly maps visual inputs to actions, improves the 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 novel tasks.

(c) Code Caching. For tasks involving familiar object types or recurring subtasks, the system reuses previously generated code, which can reduce latency and prevent code generation failures caused by unsuccessful model reasoning. For novel objects or novel tasks, EmbodiedCoder is invoked to parameterize the object and synthesize new trajectories. As the system successfully executes more tasks, it gradually builds a growing library of versatile skills that can be applied to future problems. This design achieves a balance between efficiency for known cases and generalization for open-world scenarios.

```
Prompt: ...The current subtask of the robot is {move the apple
into the drawer}. The geometric and functional parameters of
the task-related objects are {object={object_geoparam,
object_funcparam}}. The current gripper pose is {tcp0}. Please
plan an end-effector trajectory for the gripper to accomplish
the current subtask. The trajectory must ensure reachability
and task requirements, satisfy kinematic/dynamic feasibility
(avoid singularities, maintain smoothness), and guarantee
environmental safety (collision avoidance). The inputs and
outputs are respectivel...

def planAppleToDrawer (apple_param, drawer_param, bottle_param,
current_pose, n_points):
  # 1. extract key geometry
  apple_c, apple_r = apple_param.center, apple_param.radius
  drawer_c, drawer_n = drawer_param.top_center,
drawer_param.normal
  bottle_c, bottle_r = bottle_param.center, bottle_param.radius
  # 2. define key poses
  lift_pose = apple_c + drawer_n * lift_height
  # lateral avoidance on the drawer plane
  offset_dir = normalize((apple_c - bottle_c) - proj(apple_c -
bottle_c, drawer_n))
  # control points for obstacle avoidance with safety margin
  ctrl1 = lift_pose + offset_dir * (bottle_r + safety_margin)
  place_pose = drawer_c + drawer_n * place_height
  ctrl2 = place_pose + offset_dir * (bottle_r + safety_margin)
  # 3. generate smooth trajectory with cubic Bezier
  waypts = cubic_bezier_path([current_pose, lift_pose, ctrl1,
ctrl2, place_pose], n_points)
  return waypts
```

Fig. 5: **Example of apple placement with obstacle avoidance**. The task-relevant content in prompt is obtained through subtask decomposition and geometric parameterization.

E. Motion Execution

After scene understanding and task decomposition, the robot executes each subtask sequentially. EmbodiedCoder generates a trajectory for each subtask, from which the robot samples waypoints to navigate and perform the required manipulation. At this point, the entire process from vision-language input to action output has been completed.

IV. EXPERIMENT

We evaluate the proposed system in real-world environments on long-term mobile-manipulation tasks. The experiments test (i) whether code-driven geometric parameterization and trajectory synthesis translate language goals into executable motions, (ii) how well the system generalizes to novel tasks compared with VLA and other code-generation methods, and (iii) the contribution of each module through ablations.

A. Experimental Setup

All experiments are conducted on an AgileX Cobot S Kit with a RealSense D455 RGB-D camera. The Scene Understanding and Task Decomposition module uses Qwen-2.5-VL [28] (7B) for grounding and instruction decomposition, SAM [27] for masks, and VGGT [11] for reconstruction to metric point clouds. EmbodiedCoder employs Claude-Sonnet-4 [29] to generate parameter-fitting and trajectory synthesis code. Unless otherwise stated, these models are used throughout; ablations replace either the VLM or the coding model to assess sensitivity.

B. Long-term Task Evaluation

We design five multi-step tasks that couple navigation with contact-rich 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. Each task is repeated 20 times.

We report results in Table II and compare against DovSG [2]. ReKep [22] and VoxPoser [8] cannot execute long-term procedures end-to-end and are therefore omitted from this setting. We evaluate two conditions: **cached**, where previously generated code for the same task or familiar object can be directly reused, and **non-cached**, where no prior code is available.

Our method attains comparable success rates between cached and non-cached conditions. This indicates that EmbodiedCoder does not rely on task-specific templates and that the two-stage pipeline—geometric parameterization followed by constraint-aware trajectory synthesis—supports zero-shot generalization. In particular, success rates in the cached setting are slightly higher because execution can proceed by directly reusing code that has already been verified in previous trials, thereby avoiding potential failures caused by incorrect code generated by the coding model. In contrast, DovSG [2] performs well on short pick-and-place segments but fails on tasks requiring additional structure, such as door or drawer operation. This gap is consistent with our method's ability to (i) parameterize articulated and functional parts (e.g., door handle as a cylinder) and (ii) synthesize trajectories constrained by those parameters and by robot kinematics. Additionally, the door-opening success rate is lower than other subtasks. One primary failure mode was observed and is consistent with the method's assumptions. 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. These findings highlight the challenges of handling multi-stage tasks under current model capabilities.

TABLE II: **Success rates of long-horizon tasks and their subtasks**, averaged over 20 trials. For entries under 'Ours', the first and second values denote success rates under *cached* and *non-cached* conditions, 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(%)
DovSG [2]	✗	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(%)
DovSG [2]	✗	50	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(%)
DovSG [2]	✗	90	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(%)
DovSG [2]	85	90	-	75
Ours	95/95	95/95	-	90/90
Get a cloth and wipe the stains off the table.				
	Pick Cloth(%)	Wipe Table(%)	-	Long Term(%)
DovSG [2]	60	✗	-	✗
Ours	85/85	75/70	-	65/60

C. Simple Task Evaluation

TABLE III: **Comparison with code-generation methods.** Success rates (%) are reported using the results of the other methods from their original papers.

Task (%)	Pour Tea	Recycle Can	Stow Book
ReKep [22]	80	80	60
VoxPoser [8]	0	30	0
Code-as-Monitor [9]	50	-	70
Ours	80	100	80

To compare with methods designed for single-step tasks, we evaluate simple tasks drawn from the out-of-distribution benchmarks used in the VLA papers, as shown in Table IV.

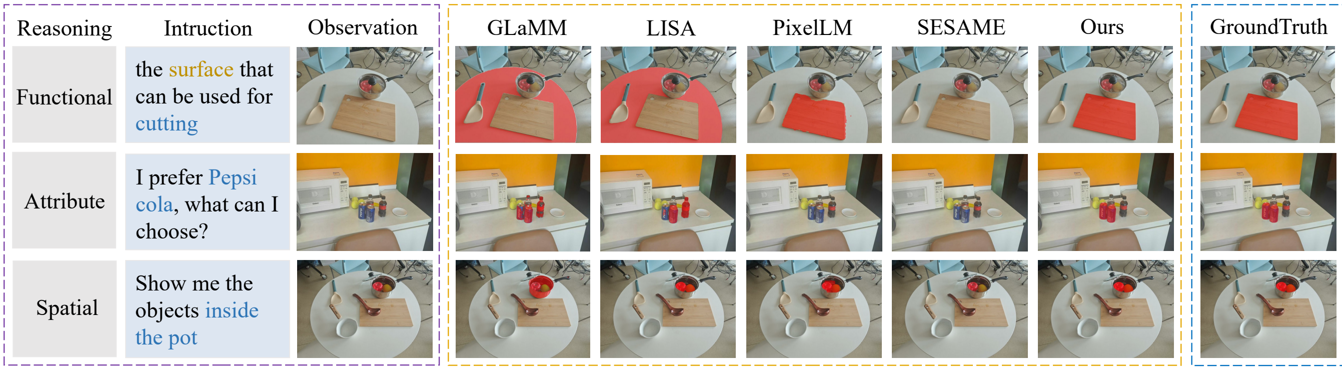

Fig. 6: Comparison of different large models on semantic grounding across functional, attribute, and spatial reasoning tasks.

TABLE IV: **Quantitative results on simple tasks compared with VLA models.** Success rates (%) are reported using the results of the other methods from 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
Average	36	84	12.2	73.3	-	89.2

Our method matches or surpasses trained VLAs without **additional training**. Tasks involving delicate flow control (e.g., pouring) show a clear advantage. This outcome is aligned with the method: the parameterization stage exposes stable surface normals, contact lines, and symmetry axes, and the trajectory code can explicitly encode motion constraints such as tilt angle limits and collision margins. By grounding semantics in the metric scene through parameterized geometry, the system avoids the ambiguity that can arise when control is derived only from learned visual features.

We also compare to code-generation approaches that predict constraint points or action scripts such as ReKep [22], VoxPoser [8], and Code-as-Monitor [9] in Table III. Our method attains consistently higher success. Unlike other approaches that infer control points or directly output trajectories, EmbodiedCoder first fits parameterized representations, which simplifies following trajectory planning and improves task success rate and stability.

D. Ablation Study

1) Effect of Object Shape: We assess grasping across objects with distinct morphology (bottles, oranges, plastic bags) and compare with AnyGrasp [30] (Table V). Our method achieves higher success rates. The improvement aligns with our parameterization design: for spheres, the planner aligns the gripper along the radial direction inferred from the fitted center and radius; for cylinders, it selects contact along the principal axis while respecting gripper aperture. AnyGrasp predicts grasps directly on point clouds without encoding gripper and kinematic constraints, which can yield unreachable or unstable poses on our platform.

TABLE V: Comparison with AnyGrasp [30] on grasp success rates (%) across different objects over 20 trials.

Task (%)	Bottle	Apple	Orange	Banana	Pepsi Can	Plastic Bag	Green Cup
Anygrasp [30]	95	70	95	80	40	90	60
Ours	100	95	100	90	75	100	80

2) Robustness of Semantic Grounding: We qualitatively assess semantic grounding across three categories of reasoning: functional, attribute, and spatial, as illustrated in Fig. 6. The visualizations reveal that different models show considerable variation in their reasoning performance. Reliable grounding and accurate segmentation of task-relevant objects are essential, since errors in location or size can directly compromise subsequent geometric parameterization and trajectory execution.

We further examined the effect of providing a two-dimensional semantic map as input to the VLM. As shown in Table VI, with this representation, the model produced more feasible task decompositions, such as navigating to a doorway before opening it for cross-room tasks. The results indicate that a 2D semantic map helps align task decomposition with real environments and reduces planning hallucinations.

TABLE VI: Comparison of five different VLMs on subtask decomposition success rates (%) with and without map.

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

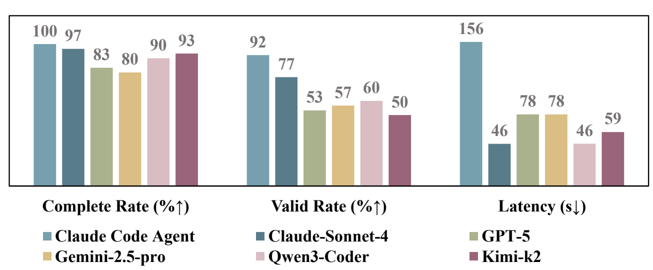

Fig. 7: Comparison of different coding models in their ability to perform geometric parameterization and trajectory synthesis under the same prompt.

3) Impact of Coding Models Capabilities: As shown in Fig. 7, we compare different coding models on their success rates in object geometric parameterization and task-oriented trajectory synthesis. The complete rate measures whether the model can generate executable code, while the valid rate measures whether the generated code successfully accomplishes the intended task, which includes both parameterization and trajectory synthesis. Claude-Sonnet-4 [29] achieves the highest success rates but also exhibits the largest latency. Other models perform significantly worse, indicating that only recent coding models possess sufficient reasoning ability for these tasks. Consequently, our paradigm is feasible only when coding models are strong enough to support reliable task reasoning and execution.

V. CONCLUSION AND LIMITATIONS

We introduce 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-term, contact-rich tasks without predefined primitives or additional training. Experiments on real robots demonstrate strong generalization to novel 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.

ACKNOWLEDGEMENTS

This work was supported by Beijing Natural Science Foundation (No. L257004), the National Natural Science Foundation of China (No. 62320106010).

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] W. Huang, F. Xia, J. Tompson, A. Zeng, B. Ichter *et al.*, "Inner monologue: Embodied reasoning through planning with language models," in *Conference on Robot Learning (CoRL)*, 2022.

[22] 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.

[23] 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.

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

[25] G. Yin, Y. Li, Y. Wang, D. McConachie, P. Shah, K. Hashimoto, H. Zhang, K. Liu, and Y. Li, "Codediffuser: Attention-enhanced diffusion policy via vlm-generated code for instruction ambiguity," *arXiv preprint arXiv:2506.16652*, 2025.

[26] J. Chen, Y. Mu, Q. Yu, T. Wei, S. Wu, Z. Yuan, Z. Liang, C. Yang, K. Zhang, W. Shao *et al.*, "Roboscript: Code generation for free-form manipulation tasks across real and simulation," *arXiv preprint arXiv:2402.14623*, 2024.

[27] N. Ravi, V. Gabeur, Y.-T. Hu, and et al., "Sam 2: Segment anything in images and videos," 2024.

[28] 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.

[29] Anthropic, "Claude sonnet 4 technical documentation," https://www.anthropic.com, 2025.

[30] H.-S. Fang, C. Wang, H. Fang, and et al., "Anygrasp: Robust and efficient grasp perception in spatial and temporal domains," 2023.