Long-Horizon Robotic Manipulation with Progressive In-Context Code Generation and Episodic Feedback

Yuan Meng[1], Xiangtong Yao[1], Haihui Ye[1], Yirui Zhou[1], Shengqiang Zhang[2], Zhenguo Sun[3], Xukun Li[1], Zhenshan Bing[4,†], and Alois Knoll[1] *IEEE fellow*

***Abstract*— Long-horizon robotic manipulation requires robots to compose perception, reasoning, and action over extended task sequences, yet existing language-conditioned frameworks often rely on dense step-wise feedback, learned action policies, or unstructured prompting, which limits robustness and generalization. We propose DAHLIA, a data-agnostic code-generation framework that treats long-horizon manipulation as episodic task planning and evaluation. DAHLIA uses progressively organized in-context examples with chain-of-thought reasoning to guide an LLM in synthesizing executable code plans from language instructions. To improve robustness under partial observability, a VLM-based reporter evaluates task outcomes only after each execution episode and provides structured feedback for replanning, avoiding the overhead of per-step inference. Experiments on LoHoRavens, CALVIN, Franka Kitchen, and real-world manipulation tasks show that DAHLIA achieves strong performance on over 30 long-horizon tasks and improves generalization to unseen scenarios, including cluttered scenes and occluded objects.**

I. INTRODUCTION

Long-horizon robotic manipulation requires a robot to interpret high-level language instructions, reason over object relations, and execute temporally extended action sequences. Recent vision-language-action models (VLAs) have shown promising progress in language-conditioned manipulation, but they typically require large-scale robot data and extensive training, while their performance often degrades when tasks become longer, more compositional, or substantially different from the training distribution [1], [2], [3]. This limitation is particularly critical in practical manipulation scenarios, where robots must solve diverse tasks under partial observability, clutter, and changing object configurations. Large language models provide an alternative route by serving as high-level task planners. Existing hierarchical approaches decompose instructions into subgoals and execute them with pretrained low-level policies or manipulation skills [4], [5], [6], [7]. Although such methods improve semantic task decomposition, they often depend on the reliability of the underlying action policy and require frequent feedback or step-wise reasoning. These requirements increase inference cost and make the system vulnerable to cumulative errors, especially in long-horizon tasks where a single incorrect intermediate decision may lead to task failure.

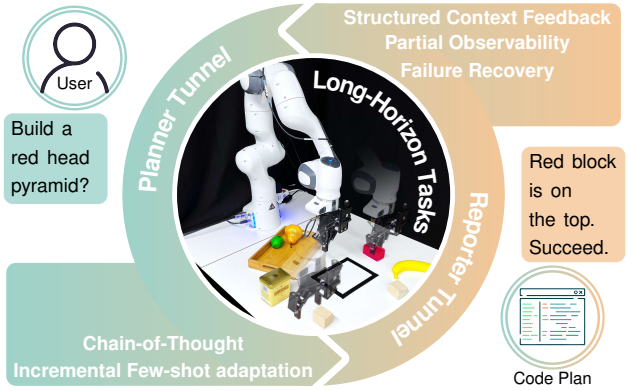


Fig. 1: Overview of our proposed framework. The dual-tunnel structure generates executable task plans and enables task-level closed-loop feedback.

Code-generation-based manipulation has recently emerged as a promising paradigm, where an LLM directly synthesizes executable programs that compose predefined robot APIs [8]. Compared with natural-language plans, executable code provides a more structured interface for long-horizon task execution and can reduce dependence on task-specific policy learning. However, existing code-generation methods still face two main challenges. First, many approaches operate in an open-loop manner or use only coarse feedback, making it difficult to recover from failures caused by occlusion, spatial misalignment, or incomplete execution. Second, their few-shot prompts are often arranged without an explicit learning structure, which can cause the model to imitate surface-level code patterns rather than infer reusable task-planning abstractions. As a result, generalization to unseen long-horizon tasks remains limited.

To address these issues, we propose DAHLIA, a **D**ata-**A**gnostic **H**ierarchical Closed-loop Framework for **L**ong-horizon **I**ncremental In-context **A**daptation. DAHLIA formulates manipulation as an episodic code-generation process. In each episode, a planner generates a complete executable task plan by composing predefined perception and manipulation APIs. The plan is executed as a sequence of primitives without per-step replanning. After execution, a reporter evaluates the task outcome from paired observations and the original instruction, then returns structured task-level feedback for the next planning episode if the goal is not yet achieved. This design introduces closed-loop recovery at the task level while avoiding continuous inference during execution.

[1]The School of Computation, Information and Technology, Technical University of Munich, Germany.
[2]The Center for Information and Language Processing, Ludwig Maximilian University of Munich, Germany.
[3]Beijing Academy of Artificial Intelligence (BAAI)
[4]The State Key Laboratory for Novel Software Technology, Nanjing University, China.
† Corresponding author: bing@nju.edu.cn
Videos and data are available at https://ghiara.github.io/DAHLIA/

A central component of DAHLIA is its structured in-context learning strategy. Instead of using randomly selected demonstrations, we organize few-shot examples in progressively increasing functional and structural complexity, and combine them with chain-of-thought reasoning cues. The examples therefore serve as a curriculum that guides the LLM to extract reusable program structures, task decomposition logic, and spatial reasoning patterns. This progressive prompting strategy improves long-horizon code synthesis and supports out-of-distribution generalization without task-specific training. The main contributions of this work are summarized as follows:

- We propose an episodic code-generation framework for long-horizon robotic manipulation, enabling an LLM to synthesize executable task plans from language instructions without task-specific training or pretrained low-level action policies.
- We introduce a progressive in-context learning strategy with chain-of-thought reasoning, which organizes examples from simple to complex and improves compositional generalization in long-horizon planning.
- We design a VLM-based episodic reporter that provides structured task-level feedback for replanning under partial observability, while avoiding the computational overhead of per-step feedback.
- We evaluate DAHLIA on more than 30 long-horizon tasks across LoHoRavens, CALVIN, Franka Kitchen, and real-world scenarios, demonstrating improved robustness and generalization over strong language- and code-generation baselines.

II. RELATED WORK

Task planning with language guidance uses LLMs as high-level planners for task decomposition via natural language, combined with pretrained low-level controllers (e.g., RL-based) to execute long-horizon tasks. SayCan [4] introduced open-loop planning with LLMs, while closed-loop variants like Inner Monologue [5], DoReMi [6], and LoHoRavens [7] incorporate various feedback mechanisms to enhance robustness. However, these approaches often struggle in unseen scenarios and degrade under environment perturbations due to the limitations of their low-level policies [7], [9]. While feedback has been well studied in language-generation-based frameworks, its potential remains underexplored in the context of code-generation-based planning.

Manipulation with code generation leverages LLMs to directly control robots through code, bypassing the need for fine-tuned low-level policies [10], [7], [8]. Recent works fall into categories such as direct code generation [8], [11], [12], task decomposition [13], and constraint-based planning [14], [15], with Code-as-Policy (CaP) [8] being a notable example. However, few studies explore both the role of structured few-shot prompting on robustness and generalization, and closed-loop feedback. Our work aims to address this gap by incorporating a closed-loop feedback pipeline, combining examples with progressively increasing functionality and difficulty.

III. METHOD

A. Problem setting

We consider embodied long-horizon manipulation at the task-planning level, where a robot interprets a natural language instruction and executes a sequence of actions to achieve a semantic goal. Our focus is on the generation, evaluation, and revision of task-level executable plans, rather than continuous control or low-level policy learning.

A **task** is specified by a language instruction l and an initial environment state s_0 obtained through perception. The objective is to generate an executable code plan composed of mid-level manipulation primitives that transforms the system into a goal-satisfying terminal state. We assume access to a library of predefined Application Programming Interfaces (APIs) (e.g., pick-and-place, pose queries), which encapsulate temporal constant low-level motion execution with fixed interfaces.

We adopt an **episodic execution model with temporal abstraction** to describe the process logic of our framework. In each episode, the system (i) generates a complete task plan as executable code, (ii) executes the plan open-loop as a sequence of primitives, and (iii) evaluates the outcome after the entire plan finishes. No per-step or per-skill feedback is used within an episode; feedback is introduced only **between episodes**. This design is motivated by the observation that, under code-based execution, primitive actions are typically executed in a predictable and repeatable manner via deterministic APIs, while task failures more often arise from environment-level perturbations such as occlusions that can only be meaningfully assessed after execution. Moreover, avoiding per-step feedback reduces redundant inference overhead, which reduces the efficiency of real-time robotic operation as observed in prior work [16], [6], [7].

Based on this execution model, the system operates in a **task-level feedback loop**. After each episode, the task outcome is assessed with respect to the original instruction. If the task is incomplete or misaligned, structured task-level feedback is provided to guide the generation of a revised plan in the next episode. This feedback does not modify low-level control, but supports episodic task recovery and replanning under partial observability, distinguishing our approach from closed-loop control or policy-learning-based methods.

B. System interfaces and language conditioning

Under the above task-level execution model, the key challenge is to ensure that a language model produces reliable and executable task plans that are consistent with the robot's available capabilities and the environment's spatial conventions. To this end, we instantiate the system by explicitly specifying (i) the manipulation and sensing interfaces, (ii) a shared spatial reference, (iii) execution constraints that enforce stable code generation, and (iv) progressively organized in-context plan templates that provide structured examples for long-horizon reasoning (Fig. 2, left).

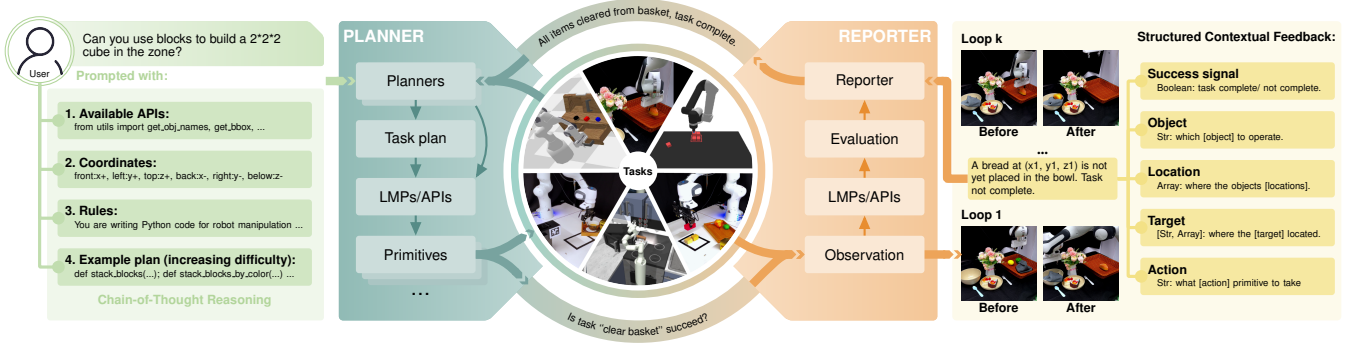


Fig. 2: Framework of DAHLIA. Our framework utilizes a dual-tunnel pipeline to implement a closed-loop feedback at the task level for various long-horizon manipulation tasks. The planner, powered by LLMs, converts human instructions into executable code plans, leveraging few-shot incremental adaptation in CoT to generate and perform multi-step primitives at once. The reporter, powered by a VLM, evaluates task outcomes based on paired visual observations and task instructions, then provides structured contextual feedback to the planner, enabling re-planning and robust performance in unstructured environments.

1) Manipulation and state interfaces: We provide the framework with a library of predefined manipulation and scene-query APIs that serve as the only executable building blocks for task plans. These interfaces encapsulate low-level motion execution (e.g., `put_first_on_second`) as well as state access (e.g., object identities, poses, bounding boxes, and free-space queries). A generated plan is considered valid only if it calls these APIs with correct argument types and returns code in the prescribed format. Table I summarizes the available APIs and their intended functionality.

TABLE I: Instructions of APIs.

API Name	Instruction
get_obj_names()	Retrieves all objects in the environment with attributes such as color, shape, and unique IDs.
get_obj_pos(obj), get_obj_rot(obj)	Obtains the 3D position and rotation (Euler angle or quaternion) of a specific object (**obj**).
get_bbox(obj)	Provides the axis-aligned bounding box of an object (**obj**) to aid in spatial inference.
denormalize(pos)	Converts normalized coordinates (**pos**) into actual environment coordinates via linear transformation.
is_target_occupied(targ)	Checks if a target position or object (**targ**) is occupied and returns the names of occupying objects.
get_random_free_pos(targ, area)	Returns a random free position within a specified area (**area**) that does not occupy the target (**targ**).
put_first_on_second(obj1, obj2)	Executes a pick-and-place primitive to position (**obj1**) on (**obj2**) or a specified pose.

2) Spatial reference and coordinate system: Long-horizon tasks often require spatial reasoning over relative descriptions (e.g., "left of", "in front of", "on top of"), while the underlying environments differ in their coordinate conventions. To standardize spatial grounding across tasks, we provide an explicit scene reference frame and axis semantics (e.g., "front: $x+$", "left: $y-$", "top: $z+$"). The framework can perform intermediate reasoning in a normalized coordinate system and convert to environment coordinates using `denormalize(pos)` when issuing primitive calls.

3) Execution rules and constraints: To improve generation stability and prevent invalid code, we supply a small set of explicit execution constraints as natural-language rules embedded as comments. These constraints specify the role of each component (e.g., writing executable Python plans), restrict the action space to the provided APIs (e.g., no external imports), and encourage safe resolution of ambiguous references via dedicated parsing utilities (e.g., `parse_obj_name()` and `parse_position()`). The following snippet illustrates the type of constraints used:

```
You are writing Python code for robot manipulation.
Refer to the code style in the examples below. You
can use the existing APIs above; you must NOT
import other packages...
```

4) In-context plan examples with progressive complexity: Beyond interface specifications, we condition the framework with a small set of example plans to provide templates for composing primitives in long-horizon tasks. Naively appending randomly selected examples is often insufficient when the target task differs substantially from the demonstrations. We therefore organize examples in a progressively increasing order of functional and structural complexity, and pair them with CoT-style reasoning cues to emphasize **decomposition and plan logic** rather than surface-form imitation. Concretely, earlier examples demonstrate basic compositions (e.g., `stack_blocks(objs, pos)`), while later ones introduce richer constraints (e.g., color-conditioned stacking via `stack_blocks_by_color(objs, pos, target_color)`), and may further include alignment requirements such as rotation-aware placements. This structured conditioning encourages the framework to reuse reusable program structures and improves robustness when scaling to longer horizons and novel task configurations.

With the above interfaces and contextual conditioning in place, DAHLIA instantiates a dual-tunnel task-level loop consisting of (i) a planner that generates executable code plans and (ii) a reporter that evaluates outcomes and triggers plan revision (Fig. 2). Inspired by previous work [8], we treat the language model as a program synthesizer that composes predefined primitives into a coherent task plan. For reproducibility, we provide more prompting detailed information in Appendix A.

C. Task-level planner

The planner (Fig. 2 darkgreen part) operates at the task level and is responsible for synthesizing executable code plans that compose manipulation primitives to achieve the given instruction. Given the system interfaces defined in Sec. III-B, the planner generates a complete task plan as Python code, where each plan corresponds to a single execution episode. The planner is instantiated as a primary LLM agent, supported by a set of auxiliary language model

TABLE II: Instructions of LMPs.

LMP Name	Instruction
parse_obj_name(dsc, ctxt)	Filters and returns object names from the context (**ctxt**) that match the target description (**dsc**) provided by the planner.
parse_position(dsc)	Converts a language description (**dsc**) of a location into one or more position coordinates or poses.
parse_function(dsc)	Implements new APIs by parsing the planner's description (**dsc**) and generating their functionality automatically.
parse_completion(dsc, ctxt)	Evaluates task completion from the context (**ctxt**) that match the target description (**dsc**).

programs (LMPs) that assist with localized reasoning. With direct access to structured scene information through APIs, the primary planner does not rely on end-to-end multimodal inference. Instead, it focuses on high-level task decomposition and plan composition, mitigating the accumulation of reasoning errors commonly observed in previous per-step decision pipelines.

Natural language task instructions often involve complex relational or attribute-based references, such as "the third block from the left that is different in color from the rightmost block but has the same size." While the primary planner is responsible for generating the overall task structure, resolving such detailed references can be non-trivial. To address this, the planner delegates specific sub-problems to dedicated LMPs, such as parsing object descriptions, grounding spatial expressions, or synthesizing auxiliary functions. Each LMP performs a narrowly scoped transformation (e.g., parsing object names or positions) and returns structured results to the primary planner. This modular delegation allows the planner to reason compositionally without entangling global task logic with low-level parsing details. The set of LMPs used in our framework is summarized in Table II. Together, these components enable the planner to construct coherent long-horizon task plans by composing reusable primitives and intermediate reasoning utilities.

Finally, following the execution model introduced in Sec. III-A, the planner produces plans that are executed open-loop within an episode. All primitives in a plan are executed sequentially without intermediate replanning or per-step feedback. This episodic planning strategy leverages temporal abstraction to reduce computational overhead and avoids excessive sensitivity to transient execution noise, while still allowing corrections through inter-episode replanning.

D. Episodic feedback with reporter

While the planner generates and executes task plans, it does not directly observe whether the resulting environment state satisfies the task goal, particularly under partial observability or execution imperfections. To close the task-level feedback loop, we introduce a reporter (Fig. 2 orange part) that evaluates episodic outcomes after each execution episode. The reporter is implemented as a VLM whose role is strictly limited to assessing task outcomes rather than generating actions or modifying control. Operating only at episode boundaries, the reporter evaluates the post-execution state with respect to the original task instruction and determines whether the goal has been achieved.

Concretely, the reporter receives (i) the task instruction, (ii) the current execution task plan generated by the planner, and (iii) paired RGB-D observations captured before and after the execution episode. Using these inputs, it evaluates task completion via the `parse_completion` interface and produces structured task-level feedback. As explained in Sec. III-A, unlike step-wise monitoring approaches, the reporter does not intervene during primitive execution and does not reason over intermediate control signals.

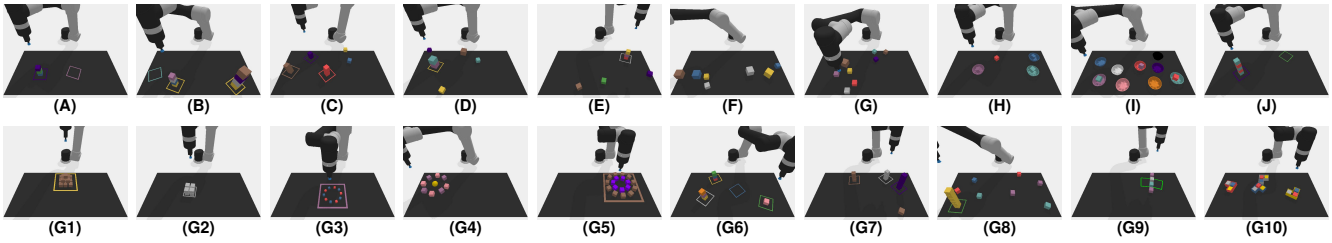
The reporter output consists of two components: (i) a binary success flag indicating whether the task is complete, and (ii) a structured description of the residual task state when completion fails. This structured feedback encodes four key elements: the relevant object, its current location, the intended target, and the missing or incorrect action required to resolve the mismatch. For example, the reporter may return: *"a red block [object] at* (x,y,z) *[location] is not yet stacked [action] on the target block [target]."* This explicit, task-oriented representation enables the planner to revise its next plan without relying on implicit or free-form language feedback. By constraining the reporter output to a structured schema, we reduce ambiguity in failure diagnosis and facilitate consistent plan revision across episodes.

Importantly, we do not require the reporter to have perfect perception, nor does it guarantee correctness in all cases. Its judgments may be affected by occlusions or perceptual uncertainty, and incorrect assessments can lead to redundant replanning. Nevertheless, within our episodic execution model, such errors manifest as additional planning iterations rather than unsafe control behavior. Empirically, we observe that this evaluation-driven replanning mechanism improves robustness in long-horizon tasks where partial observability and delayed failure detection are common.

The overall planner–reporter interaction iterates until the reporter confirms task completion, forming a task-level episodic feedback loop that supports recovery from execution errors without requiring continuous control feedback.

IV. EXPERIMENT

Experiment setup: To evaluate our framework's long-horizon manipulation performance and scalability across diverse scenarios, we utilize 22 long-horizon tasks from three benchmarks: LoHoRavens [7], CALVIN [17], and Franka Kitchen [18]. LoHoRavens, built on the Ravens, focuses on tabletop manipulation using a UR5e robot with a suction gripper. The environment features objects like blocks, bowls, and zones with variations in color, size, and texture. As shown in Fig. 3, we expand the LoHoRavens task pool to 20 tasks using GenSim [19], including 10 tasks from the original pool (A-J) and 10 newly generated, more challenging out-of-distribution tasks (G1-G10) for generalization evaluation, which are unseen in prompts or the training set. CALVIN offers four indoor setups featuring a Franka robot interacting with objects like drawers, sliding doors, buttons, and colored blocks. Similarly, the Franka Kitchen provides a kitchen scene where a Franka operates microwaves, burners, lights, and cabinets, etc. For both CALVIN and Franka Kitchen, we modify their base code setup (e.g., action space) and provide



Label	Instructions of Original Tasks	Label	Instructions of Generated Tasks
A	"Stack all blocks in the [COLOR] zone."	G1	"Construct a 9-4-1 rectangular pyramid structure in the zone using 14 blocks of the same color."
B	"Stack blocks of the same size in the [COLOR1] zone and [COLOR2] zone respectively."	G2	"Construct a 2*2*2 cube structure in the zone using 8 blocks of the same color."
C	"Stack all the blocks of the same color together in the same colored zone."	G3	"Construct a circle with suitable radius with alternating [COLOR1] and [COLOR2] blocks in the zone."
D	"Stack only the [SIZE] blocks of [COLOR_TYPE] color in the [COLOR] zone."	G4	"Construct a circle with suitable radius with alternating [COLOR1] and [COLOR2] blocks around the ball."
E	"Stack all the blocks, which are to the [REL_POS] of the [COLOR1] block with [POS_TYPE] distance larger than 0.05 unit, in the [COLOR2] zone."	G5	"Construct two concentric circles in the zone using [NUM] [COLOR1] and [NUM + 4] [COLOR2] blocks."
F	"Move all the blocks in the [POS1] area to [POS2] area."	G6	"Divide the blocks into groups of [NUM] and stack each group (also including the group with block number less than [NUM]) in a different zone."
G	"Move all the [SIZE] blocks in the [POS1] area to [POS2] area."	G7	"Place the maximal odd number of blocks of the same color in each correspondingly colored zone."
H	"Put the blocks in the bowls with matching colors."	G8	"Stack blocks of the same color that has the largest quantity in the zone."
I	"Put the blocks in the bowls with mismatching colors."	G9	"Arrange all blocks on the zone bisector line between two symmetrically placed zones evenly on the tabletop, and the gap between two adjacent blocks' edges should be near the block size, and the line connecting the center of the zones also bisects these blocks."
J	"Stack blocks with alternate colors on the [COLOR1] zone, starting with the [COLOR2] color."	G10	"Each L-shaped fixture can hold three blocks, suppose the block size is (a,a,a), then in fixture's local coordinate system, the three places that can hold blocks are [(0,0,0),(a,0,0),(0,a,0)]. Fill in all the fixtures which have random position and rotation with blocks, and make sure in the end in every fixture there are three blocks with different colors."

Fig. 3: Completion snapshots of LoHoRavens tasks. A-J are original tasks from LoHoRavens, and G1-G10 demonstrate generated tasks. The table shows the corresponding task instructions.

the necessary interfaces. For real-world experiments, we evaluate our framework on a Franka robot with a RealSense D456 camera mounted at the table edge. To acquire the object identities and positions, we use Grounded SAM2 [20].

We adopt OpenAI's GPT-4o-mini [21] as both the planner and reporter in our framework. For comparison in the code-generation domain, we evaluate against the representative open-source pipeline, Code as Policy (CaP). By introducing randomly ordered in-context examples from the original task pool (Fig. 3 A-J) into CaP, we assess the impact of our structured few-shot examples arranged in progressively increasing difficulty. In the language-generation domain, we compare against state-of-the-art baselines from LoHoRavens, including the oracle CLIPort [9]. Key distinctions between these baselines and our method include: (1) Baselines perform task decomposition using natural language and rely on pretrained CLIPort [9] as an action expert whose action space is end-effector Cartesian space. In contrast, our framework does not depend on pretrained action policies; instead, it treats executable code generated by the LLM as the action space. (2) Baselines rely on per-step reasoning, making them more prone to noise and cumulative errors, and less effective in long-horizon tasks. In contrast, our method executes all primitives in an episodic loop, and feedbacks are provided at inter-loop, improving both efficiency and robustness. (3) The baselines use randomly selected in-context examples from the task pool without structured reasoning guidance. Our method, however, organizes examples with progressively increasing complexity and employs CoT prompting to help the LLM extract underlying task logic and common knowledge, enabling better generalization and more reliable planning. These baseline variations are thoroughly compared with our framework in the given tasks shown in Fig. 3. Additionally, we conduct ablation studies to examine the individual contributions of task-level episodic feedback and progressively structured in-context examples with CoT reasoning. Specifically, we compare the full DAHLIA framework with (i) a planner-only variant that operates in an open-loop manner (DAHLIA (planner-only)) but with the same in-context examples setup as the original framework, and (ii) a variant that replaces progressive example ordering with randomly selected in-context examples and omits CoT guidance (DAHLIA (random)). All tasks are tested with 50 random seeds, and average success rates (SR, %) are reported. Results show our closed-loop setup achieves the highest success rates across both seen and unseen tasks.

Long-horizon manipulation: We first evaluate all models on the original LoHoRavens tasks (Fig. 3 A-J). As shown in Table III, our proposed framework, DAHLIA, outperforms all baselines in long-horizon manipulation tasks, achieving the highest SR, including 100% accuracy in five out of ten tasks. Compared to its planner-only setup and open-loop CaP, our dual-tunnel setup shows clear advantages, especially in tasks requiring precise coordination and iterative re-planning. While CaP performs well on simple tasks like stacking blocks (86% SR), its performance drops substantially on more challenging long-horizon tasks due to its use of randomly ordered in-context examples, which limits the LLM's

TABLE III: SR (%) for tasks from the original LoHoRavens task pool.

	Long-horizon Task Average Success Rate (%)									
Frameworks	A	B	C	D	E	F	G	H	I	J
CLIPort (oracle)	22	2	8	2	2	16	10	20	18	2
*Llama2^P + Flamingo^R $\rightarrow$ CLIPort	20	10	4	10	16	28	32	32	28	14
*Llama3^P + CogVLM2^R $\rightarrow$ CLIPort	20	20	24	22	14	20	20	32	30	22
*GPT-4o-mini^P + CogVLM2^R $\rightarrow$ CLIPort	26	22	30	24	20	34	28	36	36	30
*GPT-4o-mini^P + GPT-4o-mini^R $\rightarrow$ CLIPort	30	32	32	30	18	34	30	46	36	32
CaP (GPT-4o-mini)	86	32	42	36	10	60	38	30	22	26
DAHLIA (planner-only)	100	88	96	60	42	76	80	90	90	82
DAHLIA (random)	98	96	98	50	46	68	80	92	94	80
DAHLIA (Ours)	100	100	100	48	42	88	72	100	90	80

*Adopted from the LoHoRavens [7], where all baselines use planner for per-step inference, reporter for feedback, and CLIPort for action execution. ^{P}Planner model. ^{R}Reporter model.

ability to generalize and reason over spatial and contextual relationships. To isolate the effect of structured in-context learning, we further compare DAHLIA against DAHLIA (random). Although DAHLIA (random) consistently outperforms CaP and the planner-only setup, it underperforms the original DAHLIA framework on several long-horizon tasks, particularly those involving complex spatial constraints and multi-step dependencies (e.g., Tasks D, F, and G). This performance gap highlights the importance of progressively structured few-shot examples and CoT reasoning, which enable the planner to internalize reusable task abstractions and maintain coherent plan structure over long horizons. Other language generation baselines using per-step inference with CLIPort also struggle. For instance, GPT-4o-mini^{P} + GPT-4o-mini^{R} $\rightarrow$ CLIPort achieves relatively higher SR than others but suffers from cumulative errors, resulting in an average SR below 32%. In contrast, DAHLIA leverages CoT-guided in-context learning with progressively arranged examples and integrates closed-loop feedback, leading to up to 12% performance gains and effectively reducing error accumulation and ensuring robust long-horizon task execution.

Out-of-distribution performance: To further assess the generalization performance in more challenging out-of-distribution scenarios, we evaluate models models on a newly generated set of long-horizon tasks that are unseen during prompting (Fig. 3, G1-G10). No task-specific examples from this set are included in the in-context prompts or dataset, ensuring that these scenarios are novel to all models. The results in Table IV demonstrate that DAHLIA consistently achieves the highest SR, further validating its robust generalization in challenging unseen scenarios. In contrast, baseline models experience a notable decline in SR when generalized to unseen scenarios. Comparing DAHLIA with its planner-only variant highlights the importance of episodic closed-loop feedback. Incorporating outcome evaluation and re-planning leads to substantial improvements on several tasks, including G2 (+24%), G3 (+8%), G5 (+8%), and G7 (+12%), underscoring the role of iterative evaluation in recovering from execution errors. More importantly, the comparison between DAHLIA and DAHLIA (random) isolates the effect of progressively structured few-shot examples and CoT-guided reasoning. Although DAHLIA (random) performs competitively on some tasks that closely resemble those seen during prompting (e.g., G4 and G6), its performance degrades sharply on more diverse or compositional unseen tasks (e.g., G2, G3, G7, G9, and G10). This degradation indicates that, without incremental example ordering and explicit reasoning guidance, the model tends to rely on superficial alignment with previously seen patterns and lacks a deeper understanding of task structure.

In the absence of progressive few-shot conditioning, DAHLIA (random) primarily learns to match examples at a surface level, which suffices for familiar tasks but fails to transfer to genuinely novel configurations. When confronted with OOD scenarios that require recombining known skills in unfamiliar ways, the planner struggles to adapt due to insufficient internalization of reusable planning abstractions. In contrast, progressively ordered examples combined with CoT reasoning encourage the model to focus on task decomposition and compositional logic, enabling effective generalization across unseen task distributions.

Finally, closed-loop episodic feedback further enhances robustness by allowing the system to detect and correct failures after execution, leading to consistently higher SR in DAHLIA. Together, these results confirm that incremental few-shot learning and CoT-guided reasoning are critical for robust generalization in long-horizon code generation, particularly in out-of-distribution robotic long-horizon manipulation tasks.

Scalability: To assess the scalability and generalization of our framework across different embodiments and task setups, we conduct additional experiments in CALVIN and Franka Kitchen. We wrapped the underlying interface to make the naming and functionality consistent with previous APIs. The average SR for each subtask is shown in Table V. The results show that our default dual-tunnel DAHLIA setup consistently outperforms its planner-only variant, demonstrating the effectiveness of its closed-loop feedback in improving execution reliability. Our framework achieves 100% success in most tasks, including lift block, open drawer, open slide door, kettle, and slider cabinet, while also surpassing the planner-only model in complex tasks requiring precise coordination, such as rotate block (+4%) and turn on switch (+4%) in CALVIN, as well as top burner (+6%) and bottom burner (+8%) in Franka Kitchen. Notably, the planner-only model

TABLE IV: SR (%) for unseen out-of-distribution long-horizon tasks.

	Long-horizon Task Average Success Rate (%)									
Frameworks	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
CLIPort (oracle)	0	32	4	0	0	0	0	8	0	0
*GPT-4o-mini^{P} + GPT-4o-mini^{R} $\rightarrow$ CLIPort	8	36	34	30	4	28	30	18	0	0
CaP (GPT-4o-mini)	6	40	32	38	0	0	44	46	0	0
DAHLIA (planner-only)	58	76	92	100	42	98	88	88	12	8
DAHLIA (random)	36	38	40	90	50	90	72	66	0	0
DAHLIA (Ours)	50	100	100	100	50	100	100	90	18	10

*Adopted from the LoHoRavens [7], where all baselines use planner for per-step inference, reporter for feedback, and CLIPort for action execution. ^{P}Planner model. ^{R}Reporter model.

TABLE V: SR (%) for subtasks in CALVIN and Franka Kitchen

CALVIN Tasks	DAHLIA (planner-only)	DAHLIA (Ours)
lift block	100	100
rotate block	94	98
turn on switch	96	100
open slide door	100	100
open drawer	100	100
overall	98.00	99.60
Franka Kitchen Tasks	DAHLIA (planner-only)	DAHLIA (Ours)
microwave	98	100
kettle	100	100
light	98	100
top burner	90	96
bottom burner	90	98
slider cabinet	100	100
hinge cabinet	96	98
overall	96.00	98.86

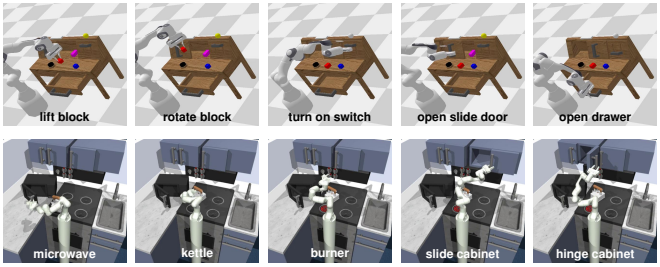


Fig. 4: Completion snapshots of CALVIN and Franka Kitchen benchmarks.

may struggle in tasks requiring highly accurate dexterous manipulation. For example, in the "top burner" and "bottom burner" tasks in Franka Kitchen, the robot must accurately reach the correct rotary knob and rotate it by 45 degrees to open the burner. Errors in task planning or dexterous execution, such as selecting the wrong switch or rotating in the wrong direction, can lead to complete task failure. In such cases, refined structured feedback plays a critical role by detecting and correcting execution errors, enabling the planner to refine its actions (e.g., correct the contact pose or rotation direction). The completion snapshots in Fig. 4 showcase the effectiveness of our framework across diverse long-horizon tasks, from structured tabletop interactions in CALVIN to articulated object handling in Franka Kitchen. These results further demonstrate our pipeline's ability to generalize across diverse task scenarios and embodiments. For more details, refer to our attached videos.

Real-world performance: Fig. 5 presents real-world demonstrations of our framework on four long-horizon tabletop manipulation tasks: (1) picking all fruits and placing them on a plate; (2) stacking blocks into a red head pyramid; (3) making coffee; and (4) retrieving all items from a basket, including a bread hidden under a towel, and placing them in a bowl without colliding with a nearby vase and flower. To increase scene reasoning complexity, we introduced distractor objects such as cucumbers and pumpkins in the first scenario, requiring accurate object identification. In the fourth scenario, an occluded bread item challenges the system's ability to recover from partial observations. Before execution, the agent scans the scene using open-world foundation models, extracts object attributes and targets from user instructions, and generates task plans. The primitives are then executed, followed by scene evaluation from the reporter, which determines task completion. As shown in the snapshots, our framework completes the first three tasks successfully in a single loop. In the fourth task, the hidden bread initially goes undetected. Upon exposure (green bbox), the reporter issues a re-planning request (orange bbox), prompting the planner with "*A bread at (x,y,z) in the basket is not yet put into the bowl.*" to update the task plan and complete the goal. This highlights the framework's robust closed-loop feedback and adaptability in real-world long-horizon scenarios. Videos are available in the attached files.

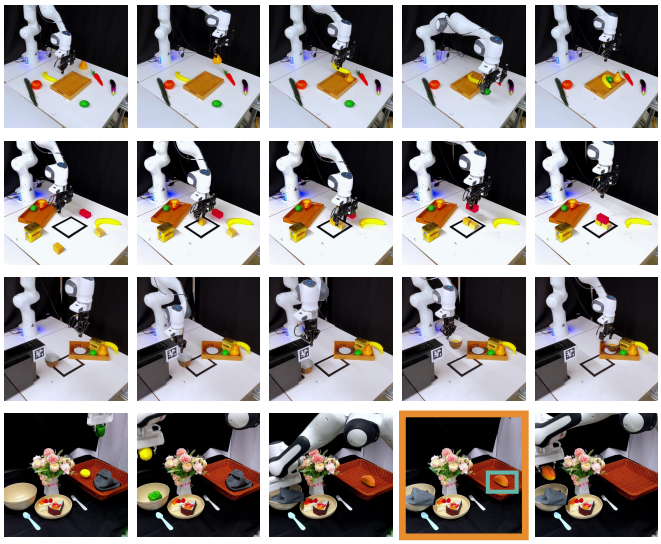


Fig. 5: Real-world demonstrations. Pick and place all the fruits on the plate (1st row). Stack a red head pyramid (2nd row). Make the coffee (3rd row). Detect (orange bbox) and re-pick-place occluded bread (green bbox) while avoiding the obstacle through closed-loop feedback recovery (4th row).

V. Conclusions

This paper presented DAHLIA, a data-agnostic framework for long-horizon robotic manipulation based on episodic code generation and task-level feedback. By organizing in-context examples with progressively increasing complexity and chain-of-thought reasoning, DAHLIA enables LLMs to synthesize reusable and executable task plans for complex manipulation tasks. A VLM-based reporter further provides structured outcome feedback for inter-episode replanning, improving robustness under partial observability without per-step inference. Experiments across LoHoRavens, CALVIN, Franka Kitchen, and real-world tasks demonstrate strong performance and generalization in both seen and unseen scenarios. Future work will explore adaptive example selection, stronger spatial grounding under severe occlusion, and tighter integration with open-world vision-language models.

APPENDIX

A. Prompt implementation details

For more information, refer to our code that will be open-sourced later. The planner prompt is basically structured as below:

```
# ------------------------------
# Existing Method Explanations
# ------------------------------
put_first_on_second(arg1 , arg2) # let robot put
arg1 to arg2
get_obj_pos(obj) -> [list] # return a list of
len(obj) of 3d position-vectors of obj, even when
obj is just one object not a list of objects
get_obj_rot(obj) -> [list] # return a list of
len(obj) of 4d quaternion orientation-vectors of
obj, even when obj is just one object not a list of
objects
parse_obj_name(query=str1, context=str2) -> list or
str # usually return a list of str or just one str
that is chosen out from str2 according to str1
which performs as description. Possible to demand
for wanted return format in str1
```

```
...(For more APIs refer to our code
implementation)...
# ------------------------------------
# Orientations in the Coordinate System
# ------------------------------------
left: y-    right: y+   front: x+
rear: x-    top: z+     bottom: z-
# ---------------------
# General Requirements
# ---------------------
You are writing Python code for robot manipulation,
refer to the code style in the examples below.
You can use the existing APIs above, you must NOT
import other packages. Our coordinate system is 3D
cartesian system, but still pay attention to the
orientations. Object sizes are important in some
tasks as they determine object positions, so
remember to get them in advance. When you are not
sure about objects or positions, you had better use
parse_obj_name() and parse_position(), and clarify
your return format demand. The examples below are
ordered from simple to complex; use them as a
curriculum to infer reusable planning patterns, and
reason step-by-step (in comments) to adapt these
patterns to the current task rather than copying
code verbatim.
# --------------
# Task Examples
# --------------
...(examples with incremental difficulty)...
# 1. example with block placement at the target
position
objects = ['yellow block with obj_id 1', 'blue bowl
with obj_id 3', 'green bowl with obj_id 6']
blocks = parse_obj_name('movable yellow blocks,
return result as list of object names', f'objects =
{get_obj_names()}')
area = parse_position('the 2d bounding box of the
top right area, return result as list of [x_min,
y_min, x_max, y_max]')
for block in blocks:
    pos = get_random_free_pos(targ=None,
    search_area=area)
    put_first_on_second(block, pos)
# 2. example with multiple block place in a target
zone.
...<code example>...
# i. example with block stack on each other.
...<code example>...
...(For more details refer to our code
implementation)...
```

The prompt for the reporter has basically the same structure, except that the inputs and requirements:

```
You are judging whether the task is completed based
on the given snapshots before and after planning
execution. Refer to the code examples below.
...(same instruction as planner)...
If you find the task is not complete, provide
additional feedback for replanning. This should
involve: objects to operate, the positions, the
actions to take, and the target to achieve.
```

REFERENCES

[1] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi *et al.*, "Openvla: An open-source vision-language-action model," *arXiv preprint arXiv:2406.09246*, 2024.

[2] P. Intelligence, K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai *et al.*, "$\pi_{0.5}$: a vision-language-action model with open-world generalization," *arXiv preprint arXiv:2504.16054*, 2025.

[3] Y. Meng, Z. Bing, X. Yao, K. Chen, K. Huang, Y. Gao, F. Sun, and A. Knoll, "Preserving and combining knowledge in robotic lifelong reinforcement learning," *Nature Machine Intelligence*, vol. 7, no. 2, pp. 256–269, 2025.

[4] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," *arXiv preprint arXiv:2204.01691*, 2022.

[5] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar *et al.*, "Inner monologue: Embodied reasoning through planning with language models," in *Conference on Robot Learning*. PMLR, 2023, pp. 1769–1782.

[6] Y. Guo, Y.-J. Wang, L. Zha, and J. Chen, "Doremi: Grounding language model by detecting and recovering from plan-execution misalignment," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024, pp. 12 124–12 131.

[7] S. Zhang, P. Wicke, L. K. Şenel, L. Figueredo, A. Naceri, S. Haddadin, B. Plank, and H. Schütze, "Lohoravens: A long-horizon language-conditioned benchmark for robotic tabletop manipulation," *arXiv preprint arXiv:2310.12020*, 2023.

[8] 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," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 9493–9500.

[9] M. Shridhar, L. Manuelli, and D. Fox, "Cliport: What and where pathways for robotic manipulation," in *Conference on robot learning*. PMLR, 2022, pp. 894–906.

[10] K. Chen, Z. Shen, Y. Zhang, L. Chen, F. Wu, Z. Bing, S. Haddadin, and A. Knoll, "Lemmo-plan: Llm-enhanced learning from multimodal demonstration for planning sequential contact-rich manipulation tasks," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2025, pp. 11 972–11 978.

[11] 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," *CoRR*, 2024.

[12] M. G. Arenas, T. Xiao, S. Singh, V. Jain, A. Ren, Q. Vuong, J. Varley, A. Herzog, I. Leal, S. Kirmani *et al.*, "How to prompt your robot: A promptbook for manipulation skills with code as policies," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 4340–4348.

[13] 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," in *Proceedings of the 41st International Conference on Machine Learning*, 2024, pp. 36 434–36 454.

[14] 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," *Proceedings of Machine Learning Research*, vol. 229, 2023.

[15] W. Huang, C. Wang, Y. Li, R. Zhang, and L. Fei-Fei, "Rekep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation," in *Conference on Robot Learning*. PMLR, 2025, pp. 4573–4602.

[16] S. Belkhale, T. Ding, T. Xiao, P. Sermanet, Q. Vuong, J. Tompson, Y. Chebotar, D. Dwibedi, and D. Sadigh, "Rt-h: Action hierarchies using language," *arXiv preprint arXiv:2403.01823*, 2024.

[17] O. Mees, L. Hermann, E. Rosete-Beas, and W. Burgard, "Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 7327–7334, 2022.

[18] A. Gupta *et al.*, "Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning," in *Conference on Robot Learning*. PMLR, 2020, pp. 1025–1037.

[19] L. Wang, Y. Ling, Z. Yuan, M. Shridhar, C. Bao, Y. Qin, B. Wang, H. Xu, and X. Wang, "Gensim: Generating robotic simulation tasks via large language models," *arXiv preprint arXiv:2310.01361*, 2023.

[20] T. Ren, S. Liu, A. Zeng, J. Lin, K. Li, H. Cao, J. Chen, X. Huang, Y. Chen, F. Yan *et al.*, "Grounded sam: Assembling open-world models for diverse visual tasks," *arXiv preprint arXiv:2401.14159*, 2024.

[21] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.