

Robotic Programmer: Video Instructed Policy Code Generation for Robotic Manipulation

Senwei Xie, Hongyu Wang, Zhanqi Xiao, Ruiping Wang and Xilin Chen

Abstract—Zero-shot generalization across various robots, tasks and environments remains a significant challenge in robotic manipulation. Policy code generation methods use executable code to connect high-level task descriptions and low-level action sequences, leveraging the generalization capabilities of large language models and atomic skill libraries. In this work, we propose Robotic Programmer (RoboPro), a robotic foundation model, enabling the capability of perceiving visual information and following free-form instructions to perform robotic manipulation with policy code in a zero-shot manner. To address low efficiency and high cost in collecting runtime code data for robotic tasks, we devise Video2Code to synthesize executable code from extensive videos in-the-wild with off-the-shelf vision-language model and code-domain large language model. Extensive experiments show that RoboPro achieves the state-of-the-art zero-shot performance on robotic manipulation in both simulators and real-world environments. Specifically, the zero-shot success rate of RoboPro on RLBench surpasses Code-as-Policies equipped with the state-of-the-art model GPT-4o by 11.6%. Furthermore, RoboPro is robust to variations on API formats and skill sets. Our website can be found at <https://video2code.github.io/RoboPro-website/>.

I. INTRODUCTION

A long-term goal of embodied intelligence research is to develop a single model capable of solving any task defined by the user. Recent years have witnessed a trend towards large-scale foundation models on natural language processing tasks [1], [2]. Scaling up these language models in terms of model size and training tokens significantly improves the few-shot performance on a wide range of downstream tasks, even achieving performance comparable to previous state-of-the-art fine-tuning methods. However, for robotic tasks, we have yet to see large-scale pre-trained models that can directly transfer across different robots, tasks and environments without additional fine-tuning.

To improve the zero-shot generalization ability of robotic models, one common approach is to unify different tasks as the next action prediction. This paradigm requires the model to directly generate low-level actions. [3], [4], [5] collected large amount of trajectories across various robots, tasks and environments. They trained vision-language-action (VLA) models derived from LLMs to map images and task instructions into discrete action tokens. Despite these models

The authors are with the Key Laboratory of AI Safety of CAS, Institute of Computing Technology, Chinese Academy of Sciences (CAS), Beijing, 100190, China, and also with the University of Chinese Academy of Sciences, Beijing 100049, China. {senwei.xie,hongyu.wang,zhanqi.xiao}@vipl.ict.ac.cn, {wangruiping,xlchen}@ict.ac.cn

Corresponding author: Ruiping Wang.

achieve better performance and show the capacity to transfer on novel objects, fine-tuning is still required when deploying on new robots and environments. Besides, it is extremely expensive to collect trajectories through real-world robots, while using human-built simulators often leads to lack of diversity and introduces additional gap between simulation platform and real-world usages.

Another line of research aims to use code as compromise solution for bridging high-level instructions and low-level execution, leveraging generalization capabilities of large language models (LLMs) and atomic skills. RoboCodeX [6] utilizes large vision-language model (VLM) to generate tree-of-thought plans and grasp preference. However, it also relies on manually-built simulation environments and human-annotated code for data curation, which is expensive and not friendly for scaling up in terms of training data.

In this work, we introduce **Robotic Programmer (RoboPro)**, a robotic foundation model, enabling the capability of perceiving visual information and following free-form user instructions to perform manipulation tasks without additional fine-tuning. The design of RoboPro introduces a unified architecture that seamlessly integrates visual perception, instruction following, and code generation by leveraging end-to-end vision-language models (VLMs). To address low efficiency and high cost in collecting runtime code data for robotic tasks, we devise Video2Code, an automatic data curation pipeline for multimodal code generation.

We draw our inspiration from the extensive amount of operational videos in-the-wild that implicitly contain necessary procedural knowledge about how to finish operational tasks. Previous research has focused on utilizing videos for large-scale supervised learning [3], [5] or extracting relevant knowledge (e.g., affordance [7]), while extracting executable policy code from videos is still under-explored. Our data curation pipeline uses the off-the-shelf VLM and Code LLM to synthesize code execution data from videos, which is much more efficient and scalable compared with generating code data from manually-built simulation environments. With Video2Code, we synthesize 115k robot execution code data along with the corresponding scene information and task descriptions from DROID [8]. Extensive experiments (examples depicted in Fig. 1) show that RoboPro achieves the state-of-the-art zero-shot performance on robotic manipulation tasks in both simulators and real-world environments. Specifically, the zero-shot success rate of RoboPro on RLBench outperforms the state-of-the-art model GPT-4o by a gain of 11.6%. It is even comparable to a strong supervised training method PerAct [9]. Furthermore, we make an early attempt

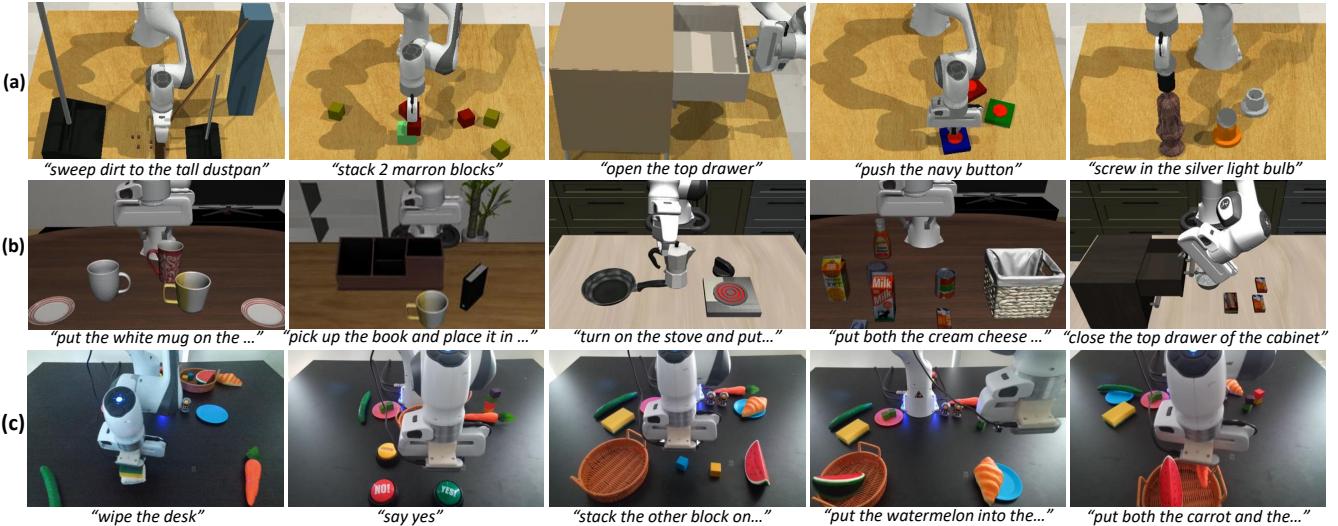


Fig. 1. Visualization of evaluation tasks and execution results. RoboPro shows impressive zero-shot performance on novel and compositional tasks in RLBench (a), long-termed manipulation tasks in LIBERO (b), and real-world tasks (c). Video demos can be found in our supplementary materials.

to discuss the adaptability of RoboPro across variations on API formats and unseen skill sets.

II. RELATED WORKS

A. Language-Guided Robot Manipulation.

Language-conditioned robot manipulation refers to the use of natural language instructions to guide robotic actions. Natural language instructions allow non-experts to interact with robots through intuitive commands and enable robots to generalize to various tasks based on natural language input [10]. Recent advancements in language-conditioned embodied agents have leveraged Transformers [11] to enhance performance on multi-task settings. One category of recent approaches is language-conditioned behavior cloning (BC), where models learn to mimic demonstrated language-conditioned actions and output dense action sequences directly. 3D BC methods [9], [12] trained from scratch perform well on specific environment, while lacking of generalization ability across environments. Vision-language-action (VLA) models [3], [5], [13] built on pre-trained large language models (LLMs) show capacity to transfer on different task settings, but need additional fine-tuning when being deployed on new environments. Another line is to create high-level planners based on LLMs [14], [15], [16], which output step-by-step natural language plans according to human instructions and environmental information. These methods show better generalization ability across environments, leveraging the reasoning and generalization ability of LLMs on language instructions and environments. However, there is still a gap between generated natural language plans and low-level robotic execution, requiring an extra step to score potential actions or decompose plans into relevant policies [17].

B. Robot-Centric Policy Code Generation.

Code-as-Policies [18] proposes that executable code can serve as a more expressive way to bridge high-level task

descriptions and low-level execution. Atomic skills to perceive 3D environments and plan primitive tasks are provided in predefined API libraries. LLMs process textual inputs and generate executable policy code conditioned on the API libraries [17], [18], [19], [20]. However, these methods rely solely on linguistic inputs, requiring detailed descriptions of environments and instructions as textual inputs, which limits their generalization and visual reasoning ability across environments. RoboCodeX [6] utilizes large vision-language model (VLM) to decompose multimodal information into object-centric units in a tree-of-thought format. Nevertheless, it relies on manually-built simulation environments and human-annotated data, which lacks environmental richness and is expensive for scaling up. Different from previous works using language-only LLMs, RoboPro enables visual reasoning ability and follows free-form instructions in a zero-shot manner. Furthermore, an automatic and scalable data curation pipeline Video2Code is developed to synthesize runtime code data from extensive videos in-the-wild in a quite efficient and low-cost fashion.

III. METHOD

A. Problem Statement

We consider language-guided robotic manipulation where each task is described with a free-form language instruction I . Given RGBD data from the wrist camera as the observation space O_t and the robotic state s_t (e.g., gripper pose at current time t), the central problem investigated in this work is how to generate motion trajectories T , where T denotes a sequence of end-effector waypoints to be executed by an Operational Space Controller [21]. However, generating dense motion trajectories at once according to the free-form instruction I is quite challenging, as I can be arbitrarily long-horizon and would require comprehensive contextual understanding. Policy code generation methods map long-horizon instructions to a set of atomic skills, leading to rapid

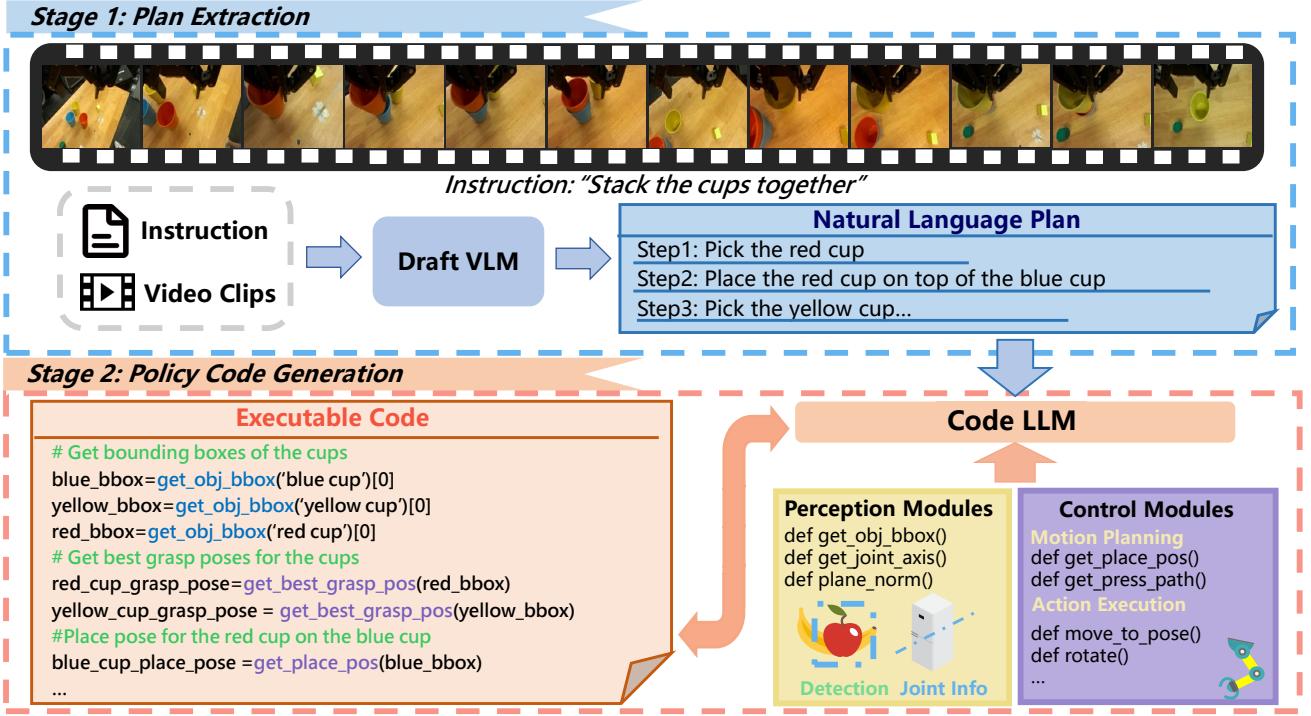


Fig. 2. The data curation pipeline of Video2Code. We first use the Draft VLM to extract a brief natural language plan for execution of the user instruction. After that, the Code LLM generates robot-centric code using the provided API library and natural language plan from the first stage.

adaptation capabilities across various robotic platforms. With comprehensive contextual understanding and advanced visual grounding capabilities, large VLMs can function as intelligent planners, translating the task execution process into generated programs due to their robust emergent capabilities.

To prompt vision-language models (VLMs) to generate policy code, we assume a set of parameterized skills with unified interface, which is defined as the API library L_{API} . L_{API} can be categorized into perception module L_{per} and control module L_{con} based on the API's role in task execution process. L_{per} is tasked with segmenting the task-relevant part point cloud Π_I and predicting the physical property ϕ_I of relevant objects, while L_{con} predicts the contact pose of the gripper and generates the motion trajectory T based on the output of L_{per} and the current robot state s_t :

$$L_{API} = \{L_{per}, L_{con}\} \quad (1)$$

$$\{\Pi_I, \phi_I\} = L_{per}(O_t, I) \quad (2)$$

$$T = L_{con}(s_t, \{\Pi_I, \phi_I\}). \quad (3)$$

With the visual observation and the language instruction, VLMs generate executable policy code $\{\pi_i, p_i\}_{i=1}^N$ conditioned on the API library L_{API} , where π_i denotes the i -th L_{per} or L_{con} calls and p_i represents corresponding parameters for API calls. Each API call generates a sub-trajectory sequence τ_i of arbitrary length (the length is ≥ 0). All sub-trajectory sequences $\{\tau_i\}_{i=1}^N$ are then concatenated to form the final complete motion trajectory T . This process is formulated as:

$$(O_t, I) \xrightarrow{VLM} \{\pi_i, p_i\}_{i=1}^N \implies \{\tau_i\}_{i=1}^N. \quad (4)$$

Explainable API calls generated by VLMs connect the observation and high-level instructions to low-level execution, enabling the capacity of zero-shot generalization in free-form language instructions and across different environments. Obviously, training such VLMs to perceive environments, follow instructions and generate executable code will inevitably require a vast amount of diverse and well-aligned robot-centric multimodal runtime code data, which poses a significant challenge.

B. Video2Code: Synthesize Runtime Code From Videos

Videos are widely available raw data sources for runtime code data synthesis. Extensive operational videos naturally provide low-level details of performing tasks such as “*how to pour tea into a cup*”, which inherently contain necessary procedural knowledge for runtime code data. Despite their favorable diversity and considerable quantity, it is still an under-explored and challenging problem how to collect executable policy code from demonstration videos efficiently. To this end, we devise Video2Code, a low-cost and automatic data curation pipeline to synthesize high-quality runtime code data from videos in an efficient way. Although open-source or lightweight vision-language models exhibit promising performance on video understanding tasks, a performance gap remains when compared to code-domain large language models in handling complex code generation tasks. As depicted in Fig. 2, to combine the visual reasoning ability of VLM and coding proficiency of code-domain LLM, Video2Code adopts a two-stage strategy.

Plan extraction. The first stage is to extract robot-centric

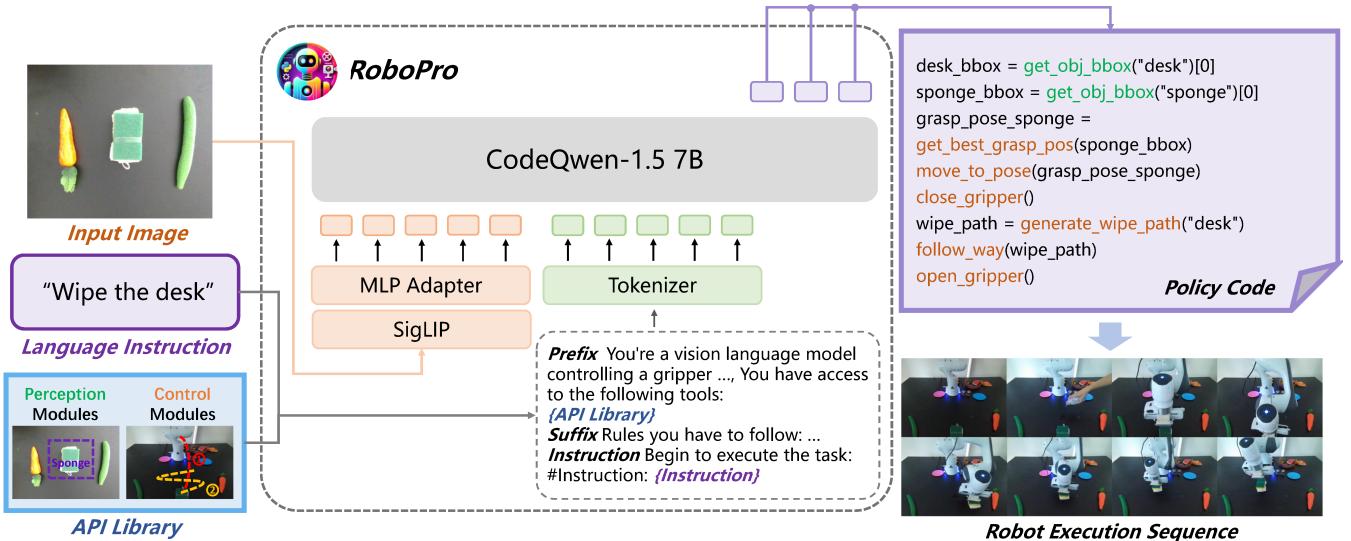


Fig. 3. The overview of RoboPro. RoboPro utilizes environmental observation and natural language instruction as multimodal input, then outputs executable policy code. Extensible API library plays a role in mapping policy code into low-level execution sequences.

plans from instructional videos. These instructional videos are filtered from DROID [8], a large-scale robot manipulation dataset with 350 hours of interaction data across 564 scenes, 86 tasks, and 52 buildings. We extract 50k independent instructional videos with at least one free-form human instruction and further clip each video into 16 key frames uniformly sampled across its duration. After that, we use Gemini-1.5-Flash [22] as the Draft VLM to generate a brief list of actions for human instruction with these key frames as reference. As shown in Fig. 2, the Draft VLM generates a step-by-step robot-centric plan from an instructional video to “*stack the cups together*”. The generated natural language plans contain knowledge and habit of human to follow free-form embodied instructions, and key visual information is extracted automatically from the instructional video.

Policy code generation. After plan extraction, we use Code LLM DeepSeek-Coder-V2 [23] to “translate” these natural language plans into executable code. A complete prompt fed into the Code LLM includes API definitions, the natural language plan, and auxiliary part containing rules to follow. In the API definitions part, parameterized API functions are classified into two categories as formulated in Sec. III-A: perception module, and control module. For each of these API functions, we provide API definitions and descriptions to demonstrate their usage. Auxiliary part contains prefix, third party tools, and rules to follow, similar to previous practices in RoboCodeX [6]. Natural language plans accompanied with original human instructions are attached at the end of the prompt. As shown in Fig. 2, step-by-step decomposed natural language plan guides the Code LLM to generate high-quality policy code in a Chain-of-Thought format. As for API implementation, we use GroundingDINO [24] and AnyGrasp [25] to get the bounding boxes and grasp preferences, respectively. Besides, we provide heuristic implementation for compositional skills on the control level. We finally collect 115k runtime code data

with task descriptions and environmental observations using Video2Code for supervised fine-tuning.

C. RoboPro: Robotic Foundation Model

RoboPro introduces a unified architecture that seamlessly integrates visual perception, instruction following, and code generation by leveraging end-to-end vision-language models (VLMs). The unified pipeline eliminates the potential loss of critical information during intermediate steps and enhances computational efficiency during inference. Powered by well-aligned image-instruction-code pairs from Video2Code, RoboPro demonstrates strong capabilities in executing free-form instructions grounded in visual observations.

Model architecture. As shown in Fig. 3, RoboPro has a vision encoder and a pre-trained LLM. They are connected with a lightweight adaptor layer consisting of a two-layer MLP. Specifically, the vision backbone first encodes the image into a sequence of visual tokens. After that, the lightweight adaptor is designed to project visual tokens onto embedding space of the LLM. In addition, we provide the API definitions and the user instruction as the text inputs. The visual and text tokens are directly concatenated and then fed into the LLM. The LLM are trained to generate the runtime code based on the visual inputs and task description. RoboPro is designed to reason on multimodal inputs and generate executable policy code for robotic manipulation. Thus, two key factors for the choice of its components are the ability of visual reasoning and the quality of code generation. RoboPro adopts SigLIP-L [26] as the vision encoder, which yields favorable performance on visual reasoning tasks. For the base LLM, a code-domain LLM, CodeQwen-1.5 [27], is utilized, which shows state-of-the-art performance among open-source code models. The model architecture and working process of RoboPro are illustrated in Fig. 3.

Training. The training procedure of RoboPro consists of three stages: visual alignment, pre-training, and supervised

fine-tuning (SFT). We first train a lightweight adaptor layer while freezing the vision encoder and LLM with LLaVA-Pretrain [28]. Then we pre-train the lightweight adaptor and the LLM on a corpus of high-quality image-text pairs [29]. For supervised fine-tuning, the 115k runtime code data generated by Video2Code (as noted in Sec. III-B) are used. To avoid overfitting and enhance visual reasoning ability, a general vision language fine-tuning dataset (LLaVA-1.5 [28]) is also involved during the SFT process. Thus, RoboPro is trained to follow free-form language instructions and perceive visual information to generate executable policy code for robotic manipulation.

IV. EXPERIMENTS

A. Zero-Shot Generalization across Tasks and Environments

Zero-shot generalization across instructions, tasks and environments is a significant challenge for robotic learning. Policy code generation methods leverage adaptability of large language models to generate code plan across tasks and scenarios in a zero-shot manner. RoboPro, a multimodal policy code generation model, is trained on real-world data. To validate zero-shot generalization of RoboPro across environments, we evaluate the performance on two different simulation environments (RLBench, LIBERO), which keep unseen during training procedure. To further guarantee fairness in comparisons, we also carefully verified that scenes and instructions in training stage are completely unseen during the test phase.

The baselines can be categorized into two groups. Our primary comparison targets are other code generation methods. They first output robot-centric policy code, then execute it with provided APIs. We evaluate their zero-shot performance on RLBench and LIBERO. CaP [18] equips large language model with ground-truth textual scene descriptions, containing object names, attributes, and instructions, to generate executable code. Following their paper, we implement CaP with GPT-3.5-Turbo [31]. GPT-4o [30] is the state-of-the-art multimodal model for various vision-language tasks. For RoboPro and GPT-4o, we require the model to directly generate the executable code given the image from the wrist camera, user instructions and API definitions. For a fair comparison, we adopt the same API library for these methods (i.e., CaP, GPT-4o, and RoboPro). Our API library shares similar design formulation as RoboCodeX [6]. The methods from another group directly output actions while require supervised training when implement on different tasks and environments, e.g., behavior cloning methods, including PerAct [9] and OpenVLA [5]. We use popular training-based methods primarily as an upper bound for reference.

1) *RLBench*: Following PerAct [9], we select 9 tasks with the requirement of novel instruction understanding or long-horizon reasoning in RLBench [32] for evaluation. Each task is evaluated with 25 episodes scored either 0 or 100 for failure or success in task execution. Code generation methods are evaluated in a zero-shot manner with same API implementation. PerAct is trained on 100 episodes in a multi-task setting for a training-based reference.

We report the average success rate on 25 episodes for each task. As shown in Table I, the zero-shot result of RoboPro surpasses language-only generation method (CaP) by 19.1%. Besides, our model significantly outperforms the state-of-the-art VLM GPT-4o by 11.6% on average success rate. To thoroughly analyze the factors contributing to the performance gap between different methods, we conducted an error breakdown for the policy code generation approaches. In the context of policy code generation methods, the successful execution of manipulation tasks relies on both the accuracy of the policy code and the capabilities of the API library. The main types of errors impacting the quality of robot-centric policy code are logical errors, functional errors, and grounding errors. These errors are associated with challenges in the appropriate selection and utility of APIs, as well as issues related to visual grounding. As depicted in Fig. 4, the results show that all these methods perform well on following functional definition of API library, causing a low occupancy of functional error. Compared with linguistic only method CaP, GPT-4o and RoboPro show a noticeable improvement in target object grounding. The main failure cases of CaP and GPT-4o fall in logical error, including API selection and proper order of API calls. In contrast, RoboPro effectively reduces this margin, mainly owing to the procedural knowledge about long-term execution learned in Video2Code. Execution errors maintain a consistent proportional relationship with successful cases, which result from API limitations rather than inaccuracies in the policy code.

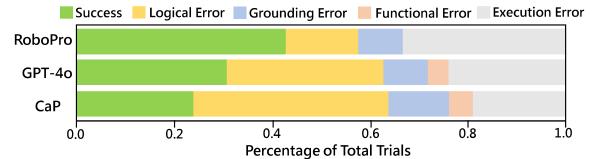


Fig. 4. Error breakdown on RLBench.

2) *LIBERO*: We choose 8 representative tasks from LIBERO [33] as the evaluation set. These tasks include short-horizon tasks which need scene understanding, and long-horizon tasks which require multi-step implementation. Similar with RLBench, each task is evaluated with 30 episodes scored either 0 or 100 for failure or success execution. We fine-tuned OpenVLA-7B [5] using Low-Rank Adaptation (LoRA) across 8 evaluation tasks as reference of training-based methods. Code generation approaches keep zero-shot setting during evaluation.

As reported in Table II, RoboPro significantly outperforms GPT-4o by a gain of 17.4% average success rate on 8 LIBERO tasks, which is aligned with the observations from the experiments on RLBench. Compared with GPT-4o, RoboPro executes more accurate sequences of actions to complete various manipulation tasks. For instance, when given the task “Turn on the stove”, RoboPro consistently approaches the stove knob, grasps it, and rotates it clockwise. In contrast, GPT-4o sometimes misinterprets the knob’s affordance, attempting to press it rather than rotate.

TABLE I

SUCCESS RATE (%) ON RLBNCH MULTI-TASK SETTING. PERACT GREYED ON NEED SUPERVISED TRAINING ON THE SIMULATION PLATFORM.

Models	Push Buttons	Stack Blocks	Open Drawer	Close Jar	Stack Cups	Sweep Dirt	Slide Block	Screw Bulb	Put in Board	Avg.
PerAct [9]	48	36	80	60	0	56	72	24	16	43.6
CaP [18]	72	4	24	40	0	36	4	20	12	23.6
GPT-4o [30]	72	20	56	36	4	40	20	20	12	31.1
RoboPro (ours)	68	48	68	44	4	48	60	32	12	42.7
w/ API Renaming	68	40	60	48	4	48	68	36	12	42.7
w/ API Refactoring	68	36	72	44	8	16	80	28	12	40.4

TABLE II

SUCCESS RATE (%) ON 8 TASKS ON LIBERO. OPENVLA GREYED ON IS FINE-TUNED ON THIS SIMULATION PLATFORM.

Models	Turn on Stove	Close Cabinet	Put in Sauce	Put in Butter	Put in Cheese	Place Book	Boil Water	Identify Plate	Avg.
OpenVLA [5]	97	97	37	60	53	93	43	40	65.0
CaP [18]	0	37	17	13	7	30	7	7	14.8
GPT-4o [30]	37	17	63	43	57	43	17	3	35.0
RoboPro (ours)	97	60	67	53	63	43	23	13	52.4

B. Zero-Shot Generalization across APIs and Skills

Another challenge for code generation methods lies in generalization across different robotic configurations, which often manifests as variations in the format and implementation of API libraries. Additionally, real-world tasks may require robots to perform unseen skills with user-specific preferences. Despite this issue has been largely under-explored in prior works, enhancing adaptability to diverse API implementations and novel skills is crucial for making code generation methods more scalable in real-world applications. We take an early step toward examining the robustness of code generation methods across varying API formats and user-specified skills.

1) *Generalization across different API formats:* We first evaluate the robustness of RoboPro to the changes of API formats, which implies that the model can internalize the atomic skills under the API interface. To assess the generalization of RoboPro under different level of changes in API library, we designed two representative sets of experiments: the *API Renaming* set and the *API Refactoring* set. For renamed APIs, we only change in their names and keep consistent in functional structure. For refactored APIs, we change in functional structure but keep their names. Take the control API “`get_best_grasp_pose()`” as an example. In the API Renaming set, it is renamed as “`generate_obj_grasp_pos()`” without changes on functionality, and in the API Refactoring set, the inputs, outputs and comments are all changed (e.g., the input format changes from “`bbox`” to “`np.ndarray`”). As shown in Table I, the performance of RoboPro on RLBench is robust to the changes in API formation, which originates from RoboPro’s ability as a generalist code model to comprehend different variations of API formats.

TABLE III
SUCCESS RATE (%) ON THREE COMPOSITIONAL TASKS FROM RLBNCH
BASED ON A NEW SET OF TASK-SPECIFIC APIs.

Models	Water Plants	Hit Ball	Scoop Cube	Avg.
CaP [18]	4	16	0	6.7
GPT-4o [30]	40	12	24	25.3
RoboPro (ours)	40	44	48	44.0

2) *Generalization across unseen skills:* To further evaluate RoboPro’s adaptability to newly defined or task-specific APIs, we select three compositional tasks from RLBNch that involve multi-step execution: Water Plants, Hit Ball, and Scoop Cube. For each task, we design a new set of task-specific APIs encompassing skills not included in RoboPro’s training phase, while follow similar functional structure with the original implementation of action modules. Under this setting, the performance of RoboPro consistently outperforms CaP and GPT-4o in a zero-shot manner. We observe that on unseen skill sets, RoboPro exhibits preferences and behaviors similar to those observed in the training skill set. Compared with methods using proprietary models, RoboPro trained with video demonstrations tends to grasp tools with appropriate affordance before performing compositional skills, and matches visual observation with vague language instructions to perform in a comprehensive manner. This robustness implies that sequential action knowledge and preferences learned from Video2Code is transferable and beneficial to perform with unseen skill sets.

C. Real-World Experiments

To evaluate the performance of RoboPro in real-world scenarios, we conduct realistic experiments on a Franka

TABLE IV

THE ZERO-SHOT SUCCESS RATE (%) OF ROBOPRO AND GPT-4O ACROSS 8 REAL-WORLD MANIPULATION TASKS.

Task	GPT-4o	RoboPro (ours)
Move in Direction	60	80
Setup Food	80	90
Distinct Base	80	70
Prepare Meal	60	60
Tidy Table	40	70
Express Words	50	60
Stack on Color	10	50
Wipe Desk	100	100
Average	60.0	72.5

Emika robot arm equipped with an Intel RealSense D435i wrist camera. As emphasized in Sec. III-A, long-horizon task decomposition and visual understanding capabilities are crucial for zero-shot generalization in language-guided robotic manipulation. To assess RoboPro’s performance in these aspects, we carefully designed 8 tasks, ranging from short-horizon to long-horizon tasks, as well as tasks that require visual comprehension. For instance, RoboPro is required to select the object with “wipe” affordance from the scene given instruction “*wipe the desk*”. Additionally, to rigorously validate RoboPro’s generalization capability across different real-world scenarios, we ensure that each task consists of at least two variations in terms of object categories and physical properties (10 tests are run for each task). We select GPT-4o as an extra baseline on real-world environments.

As shown in Table IV, RoboPro is able to achieve 72.5% success rate on average among all 8 tasks, which verifies RoboPro’s strong generalization ability in real-world scenarios without any specific fine-tuning. We also observe RoboPro exhibits impressive emergent ability in visual reasoning. For example, when asked to wipe the desk, RoboPro will choose the appropriate tool (the sponge) among irrelevant objects, and grasp it to wipe water on the desk. On directional moving and one-turn tasks, GPT-4o shows comparable performance with RoboPro (Move in Direction, Setup Food), while RoboPro shows better performance on tasks requiring visual understanding or target identification (Stack on Color, Tidy Table), which yields conclusions consistent with the experimental results on simulation platforms.

D. Ablation Study

We conduct extensive ablation studies to analyze how Video2Code and the scaling of training data impact performance on manipulation tasks. Additionally, we explore the effect of the choice of base LLM within RoboPro.

Effectiveness of Video2Code. We compare our model trained with and without Video2Code on manipulation tasks. For a fair comparison, we only remove Video2Code from the fine-tuning stage for the baseline, that is, the 115k runtime code data are excluded and only the general vision language fine-tuning dataset is used during the SFT process, as described in Sec. III-C. The first two rows of Table V show the comparison of the two settings. It is found that the Video2Code generated data have significantly improved

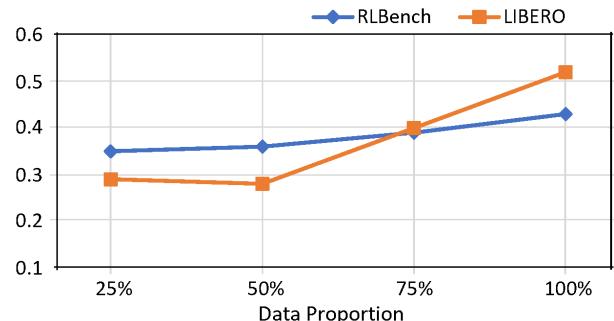


Fig. 5. Success rate on manipulation tasks across varying data proportions.

TABLE V
ABLATIONS OF VIDEO2CODE AND DIFFERENT BASE LLMs.

LLM	Video2Code	Manipulation	
		RLBench	LIBERO
CodeQwen-1.5-7B	✗	0.4	7.0
CodeQwen-1.5-7B	✓	42.7	52.4
DeepSeek-Coder-6.7B	✓	41.3	48.8

the performance on both RLBench and LIBERO by a gain of 42.3% and 45.4%, respectively, which indicate Video2Code’s efficacy in enhancing skill utility and instruction following.

Scaling of training data. We further conducted an ablation study on the dataset size. Specifically, we trained RoboPro using 115k runtime code data collected from DROID, varying the dataset proportion of SFT stage to 25%, 50%, 75%, and 100%. We evaluated the models trained with different sizes of dataset on RLBench and LIBERO. As shown in Fig. 5, results indicate that RoboPro adheres to the scaling law: training with just 25% of the data already yields a well-performing model, while its performance continues to improve as the dataset size increases. Video2Code is efficient for scaling up of runtime code data, which deserves further exploration to involve in more robotic demonstrations.

Choice of base LLM. For the components of the RoboPro framework, we evaluate its performance using different code-domain base LLMs, specifically DeepSeek-Coder-6.7B-Instruct [34] and CodeQwen-1.5-7B-Chat [27]. As shown in Table V, the version of RoboPro trained with CodeQwen-1.5-7B-Chat consistently outperforms the one trained with DeepSeek-Coder-6.7B-Instruct across both manipulation tasks. These results demonstrate that employing a more powerful base LLM for code generation task can consequently enhance performance in both tasks.

V. CONCLUSION AND FUTURE WORK

In this work, we propose RoboPro, a robotic foundation model, which perceives visual information and follows free-form instructions to perform robotic manipulation in a zero-shot manner. To address low efficiency and high cost for runtime code data synthesis, we propose Video2Code, a scalable and automatic data curation pipeline. Through extensive experiments, RoboPro achieves impressive generalization and significant performance improvement compared with

other policy code generation methods. These results indicate that incorporating procedural knowledge within operational videos into training process will bring substantially enhanced understanding of skills (i.e., API libraries) and free-form instructions. For more complex and dexterous tasks, building robust low-level skills is as important as composing them well. In the future, we would like to extract skill-relevant information from a broader range of real-world data to avoid manual efforts on skill design. A more hierarchical way to organize and implement skill is also important for generalization across skill variants.

ACKNOWLEDGMENT

This work is partially supported by Natural Science Foundation of China under contracts Nos. 62495082, 62461160331, U21B2025, and National Key R&D Program of China No. 2021ZD0111901, 2023YFF1105104. Senwei Xie, Hongyu Wang, and Zhanqi Xiao contributed equally to this work. Senwei Xie was responsible for the implementation of Video2Code, evaluation on RLBench and real-world tasks. Hongyu Wang provided support on the training of RoboPro. Zhanqi Xiao was responsible for evaluation on LIBERO and setup of the real-world environment.

REFERENCES

- [1] 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.
- [2] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, Goyal, *et al.*, “Llama: Open and efficient foundation language models,” *arXiv preprint arXiv:2302.13971*, 2023.
- [3] A. Brohan, N. Brown, J. Carbalal, Y. Chebotar, X. Chen, Choromanski, *et al.*, “Rt-2: Vision-language-action models transfer web knowledge to robotic control,” *arXiv preprint arXiv:2307.15818*, 2023.
- [4] A. Padalkar, A. Pooley, A. Jain, A. Bewley, A. Herzog, A. Irpan, Khazatsky, *et al.*, “Open x-embodiment: Robotic learning datasets and rt-x models,” *arXiv preprint arXiv:2310.08864*, 2023.
- [5] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, Rafailov, *et al.*, “Openvln: An open-source vision-language-action model,” *arXiv preprint arXiv:2406.09246*, 2024.
- [6] 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 preprint arXiv:2402.16117*, 2024.
- [7] S. Bahl, R. Mendonca, L. Chen, U. Jain, and D. Pathak, “Affordances from human videos as a versatile representation for robotics,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 13 778–13 790.
- [8] A. Khazatsky, K. Pertsch, S. Nair, A. Balakrishna, S. Dasari, S. Karamcheti, S. Nasiriany, M. K. Srivama, L. Y. Chen, K. Ellis, *et al.*, “Droid: A large-scale in-the-wild robot manipulation dataset,” *arXiv preprint arXiv:2403.12945*, 2024.
- [9] M. Shridhar, L. Manuelli, and D. Fox, “Perceiver-actor: A multi-task transformer for robotic manipulation,” in *Conference on Robot Learning*. PMLR, 2023, pp. 785–799.
- [10] T. Winograd, “Procedures as a representation for data in a computer program for understanding natural language,” 1971.
- [11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in Neural Information Processing Systems*, vol. 30, 2017.
- [12] J. Zhang, C. Bai, H. He, Z. Wang, B. Zhao, X. Li, and X. Li, “SAM-E: leveraging visual foundation model with sequence imitation for embodied manipulation,” in *International Conference on Machine Learning*, 2024.
- [13] D. Niu, Y. Sharma, G. Biamby, J. Quenum, Y. Bai, B. Shi, T. Darrell, and R. Herzig, “Llarva: Vision-action instruction tuning enhances robot learning,” *arXiv preprint arXiv:2406.11815*, 2024.
- [14] A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, *et al.*, “Do as i can, not as i say: Grounding language in robotic affordances,” in *Conference on robot learning*. PMLR, 2023, pp. 287–318.
- [15] D. Driess, F. Xia, M. S. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, Wahid, *et al.*, “Palm-e: An embodied multimodal language model,” in *International Conference on Machine Learning*. PMLR, 2023, pp. 8469–8488.
- [16] 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.
- [17] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, “Progprompt: Generating situated robot task plans using large language models,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 11 523–11 530.
- [18] 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.
- [19] S. Huang, Z. Jiang, H. Dong, Y. Qiao, P. Gao, and H. Li, “Instruct2act: Mapping multi-modality instructions to robotic actions with large language model,” *arXiv preprint arXiv:2305.11176*, 2023.
- [20] 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,” in *Conference on Robot Learning*. PMLR, 2023, pp. 540–562.
- [21] O. Khatib, “A unified approach for motion and force control of robot manipulators: The operational space formulation,” *IEEE Journal on Robotics and Automation*, vol. 3, no. 1, pp. 43–53, 1987.
- [22] G. Team, R. Amil, S. Borgaeud, Y. Wu, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, *et al.*, “Gemini: a family of highly capable multimodal models,” *arXiv preprint arXiv:2312.11805*, 2023.
- [23] Q. Zhu, D. Guo, Z. Shao, D. Yang, P. Wang, R. Xu, Y. Wu, Li, *et al.*, “Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence,” *arXiv preprint arXiv:2406.11931*, 2024.
- [24] S. Liu, Z. Zeng, T. Ren, F. Li, H. Zhang, J. Yang, C. Li, Yang, *et al.*, “Grounding dino: Marrying dino with grounded pre-training for open-set object detection,” *arXiv preprint arXiv:2303.05499*, 2023.
- [25] H.-S. Fang, C. Wang, H. Fang, M. Gou, J. Liu, H. Yan, W. Liu, Y. Xie, and C. Lu, “Anygrasp: Robust and efficient grasp perception in spatial and temporal domains,” *IEEE Transactions on Robotics*, 2023.
- [26] X. Zhai, B. Mustafa, A. Kolesnikov, and L. Beyer, “Sigmoid loss for language image pre-training,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 11 975–11 986.
- [27] 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.
- [28] H. Liu, C. Li, Y. Li, and Y. J. Lee, “Improved baselines with visual instruction tuning,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 26 296–26 306.
- [29] L. Chen, J. Li, X. Dong, P. Zhang, C. He, J. Wang, F. Zhao, and D. Lin, “Sharegpt4v: Improving large multi-modal models with better captions,” *arXiv preprint arXiv:2311.12793*, 2023.
- [30] OpenAI, “Hello gpt-4o,” May 2024. [Online]. Available: <https://openai.com/index/hello-gpt-4o/>
- [31] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, *et al.*, “Training language models to follow instructions with human feedback,” *Advances in neural information processing systems*, vol. 35, pp. 27 730–27 744, 2022.
- [32] S. James, Z. Ma, D. Rovick Arrojo, and A. J. Davison, “Rlbench: The robot learning benchmark & learning environment,” *IEEE Robotics and Automation Letters*, 2020.
- [33] B. Liu, Y. Zhu, C. Gao, Y. Feng, Q. Liu, Y. Zhu, and P. Stone, “Libero: Benchmarking knowledge transfer for lifelong robot learning,” *Advances in Neural Information Processing Systems*, vol. 36, 2024.
- [34] D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, Bi, *et al.*, “Deepseek-coder: When the large language model meets programming—the rise of code intelligence,” *arXiv preprint arXiv:2401.14196*, 2024.