LLM-powered runtime code change in robots

Haixuan Xavier Tao
1ms.ai
Paris, France
tao.xavier@outlook.com

Philipp Oppermann
Freelancer
Karlsruhe, Germany
mail@phil-opp.com

Yong He
Futurewei Technologies
Dallas, Texas, USA
yhe@futurewei.com

***Abstract*—Artificial intelligence (AI) is evolving exponentially, from residing mainly in the realm of human imagination and advanced scientific research to becoming an undeniable part of day-to-day human reality. Many of the AI applications already revolutionizing society such as ChatGPT are generated using large language models (LLMs) trained on huge amounts of data to recognize and interpret human language. The rise of AI goes hand-in-hand with automation and robotics, fueling the need of more intelligent systems and robots. However, one of the main difficulties with robotic applications is understanding human instruction. A very natural progression of LLMs is to move towards robotic applications and enhance functionality.**

In this paper, we show how allowing LLMs to modify robotic codebase at runtime enables new human-machine interaction. To achieve this, we use dora-rs, a robotic framework capable of changing code at runtime while keeping state, also known as hot-reloading. By pairing dora-rs with LLMs, we demonstrate that robots can be controlled and instructed with natural language to modify any aspect of the robot codebase. This approach allows new human-robot interactions that were previously inaccessible due to the limitations posed by the need to use existing predefined interfaces, thus paving the way to more sophisticated and wider use of robotic applications that can better understand and respond to human needs. The code is available at: github.com/dora-rs/dora and huggingface.co/datasets/haixuantao/dora-robomaster

***Index Terms*—LLM, robotics, AI, RAG**

I. Introduction

LLMs make it possible to write quality code using natural language. With their understanding of natural language and coding syntax, LLMs can generate very powerful code assistants such as ChatGPT and Github Copilot.

The idea of using LLMs to generate code to control a robot from natural language has been popularized by Liang et al. where they demonstrate LLM capabilities to generate functional code for a robot to solve specified tasks [1]. Liang et al. showcased that some LLM has great coding and reasoning skills and that they can solve many concrete robotic challenges in zero-shot [1]. Liang et al. limited the coding capabilities of LLMs to only a subset of the codebase called the Language Model Program (LMP), but we want to extend to the entire robot codebase.

This approach has not been actively researched as the commonly used Robot Operating System (ROS) [2] requires stopping the robot, building the codebase, and then restarting the robot in order to implement any modification. The ROS/ROS2 compilation step is required as the message format is protobuf-encoded, making message compilation necessary even when using an interpreted language like Python.

These issues have pushed us to investigate hot-reloaded code change to provide a way to interact with a robot during runtime and use LLM as the driver of those changes, making it possible to use natural language to modify code at runtime.

Being able to modify and override a codebase at runtime provides access to instruction that would have been impossible to implement with predefined interface.

II. Methods

This section deep dives on how we build our LLM-powered hot-reloading robotic framework. It first describes dora-rs unique implementation details, and then dives in the code generation components.

A. dora-rs

1) Dataflow Paradigm: dora-rs was initially written as a framework to help build robotic applications. It implements a dataflow paradigm where tasks are split between nodes that are isolated as individual processes. Each node defines its inputs and outputs to connect with other nodes. The dataflow paradigm has the advantage to create an abstraction layer that makes robotic applications modular and easily configurable. Nodes can be sensor nodes, network nodes, as well as compute nodes. Splitting a robotic application into distributed nodes makes it possible to edit and reload specific nodes of the application.

2) Communication: Communication between nodes is handled with shared memory on a same machine and Transmission Control Protocol (TCP) on distributed machines. Our shared memory implementation tracks messages across processes and discards them when obsolete. Shared memory slots are cached to avoid new memory allocation.

3) Message Format: Nodes communicate with Apache Arrow Data Format as it defines a C data interface without any build-time or link-time dependency requirements [3]. This makes hot-reloading simpler as compiling and linking message formats at runtime can be slow and error-prone. Apache Arrow message format also makes sharing data zero-copy as data can be interpreted at runtime from all languages. Apache Arrow also uses the same memory layout as many scientific libraries, making it zero-copy to convert Apache Arrow Data to the format used by LLMs such as NumPy [4] or PyTorch [5].

4) Python: dora-rs nodes are written in Python removing the need for compilation. Hot-reloading compiled code is hard as changing a dynamic library at runtime is very error-prone

[6]. LLMs also perform better on Python according to Peng, Chai, and Li [7]. Dora also supports C, C++ and Rust nodes but those nodes are not hot-reloadable.

5) Hot-Reloading: The hot-reloading sequence is the following:

1) Check for code change from OS signal
2) Spawn a new node from the updated code
3) Catch error upon initialization and keep current running node in case of failure
4) Copy running node state into the new node
5) Swap the running node with the new node
6) Catch error on following input to allow runtime debugging

This hot-reloading sequence has the unique ability to keep the state of the running node. Any progress within the state will be copied in the reloaded node, thus avoiding the loss of running state.

B. Code Generation

To generate code, we provide an instruction and a source code to an LLM that is able to implement the code modification required to follow the instruction. This modification is then saved which triggers the hot-reloading process.

The code generation implementation follows a common Retrieval Augmented Generation (RAG) [8] pattern applied to the codebase of the robot.

Currently, dora-rs has been tested on two LLMs: Finetuned OpenHermes-2.5-Mistral-7B and ChatGPT-4-turbo.

1) ChatGPT: ChatGPT is a LLM developed by OpenAI [9]. It demonstrates advanced problem solving capabilities and is very instinctive when provided with code, without needing prompts to extensively explain the instruction.

ChatGPT cannot be embedded offline but is easily accessible via its API.

ChatGPT has a low generation speed of 22 token/s or about 16 words/s [10] which is considered slow as our average nodes source contains about 100 lines of codes, equivalent to about 350 tokens, which takes about 30 seconds to a minute to generate a response to our code change request.

Our system prompt using ChatGPT is “You are a helpful assistant.” and user prompt template is [11]:

````
this is a python code:
```python
{code}
```
{instruction}

Format your response by:
- Showing the whole modified code.
- No explanation is required.
- Only code.
````

The response is processed by copying the code block enclosed in triple backticks and pasting it over the original code.

2) Mistral 7B: Mistral 7B is an open-source LLM model developed by Mistral.ai [12]. Mistral 7B has the advantage of having a small number of parameters and can be embedded into applications within a laptop with sufficient GPU Memory. We use Mistral 7B OpenHermes 2.5: “TheBloke/OpenHermes-2.5-Mistral-7B-GGUF” as it has been finetuned on code generation using ChatGPT4 output. Mistral 7B can make minor code changes such as changing variables, modifying functions or adding simple if-statements. It will, however, struggle with grasping complex domain technicalities and major code changes.

Mistral 7B, being a smaller model, can generate at a faster speed of 94 token/s [13] or about 74 words/s which takes about couple of seconds making instant reactive response.

Our prompt template for Mistral 7B model is [14]:

````
<|im_start|>system
You're a python code expert.
Respond with only one line of code.
<|im_end|>
<|im_start|>user
```python
{code}
```

{instruction}
<|im_end|>
<|im_start|>assistant
````

The response is processed by extracting the generated line of code and searching for the most similar original line of codes using the Levenshtein distance and replacing it.

3) Instructions: Instructions are in natural languages and do not need to specify any code implementation, as LLMs are able to deduct the required code modification. Instructions can be made through speech recorded by a microphone and converted using OpenAI Whisper “speech-to-text”. [15]

4) Codebase embeddings: To retrieve the right source code to modify, we conduct a vector search of the codebase. We first map all nodes source code into a vector embedding database using the FlagEmbedding embedding model: “BAAI/bge-large-en-v1.5” described by Xiao et al. as it is open source and easy to integrate using Python [16]. We then map the instruction into the same embedding space and search for the most similar source code using cosine similarity [17]. Being able to match the embedding of an instruction in natural language with code relies on the embedding model to correctly encode the code documentation such as docstrings and comments. We currently do not embed imported libraries; rather, we rely on the code to be self-explanatory enough for the LLMs to do the modification.

The following schema is a summary of the different steps in the reloading process (Fig. 1)

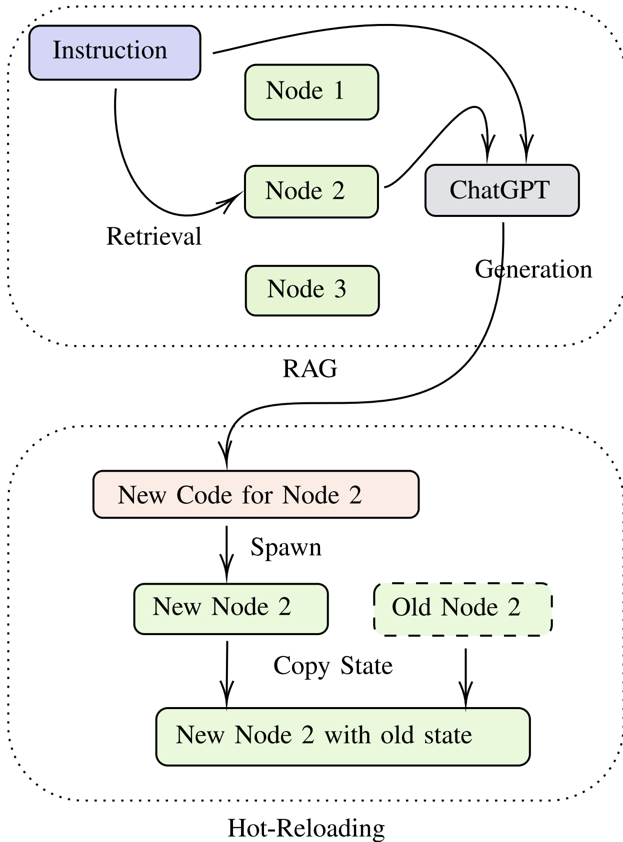

Fig. 1. Schema of the LLM-powered hot-reloading process

III. Experiment

1) DJI Robomaster: To demonstrate how using a LLM within a robotic codebase improves human-machine interaction, we tested our method on a DJI robomaster S1 [18].

The DJI Robomaster is a rover programmable using Python, costing approximately 500 US dollars. This relatively affordable platform has very few dependencies and is very easy to build. It can be moved in two dimensions and has a rotating gimbal that is attached to a webcam that we can use for object detection. We connected dora-rs with the DJI robomaster using DJI Robomaster Python SDK.

All code and recorded data can be found at huggingface.co/datasets/haixuantao/dora-robomaster.

2) Scenarios: We tested our LLM-powered hot-reloading functionality in multiple scenarios:

- Modifying static variables such as changing speed or the destination objectives. This scenario requires retrieving the right code and making the right modification. It can be challenging as the instruction wording might not be directly translatable to the code syntax.
- Changing function arguments such as adding, removing or modifying a parameter in a function call. This scenario requires understanding the underlying function and expressing the required change in the proper way.
- Adding control flow such as adding if-statement. This scenario requires the generation of basic programming skills and being able to integrate within an existing code.
- Adding new functionalities such as modifying the entire behavior of a robot. This scenario requires an implementation of a functionality from scratch. (Fig. 2)

We tested each scenario 20 times with different example inputs on the robomaster.

3) Metrics: We break down the success rate at three different stages: Retrieval, Generation and Code Logic. We measure the success rate by manually annotating the outcome and resetting the robot using hot-reloading in case of failure.

```
# yaw-axis angle in degrees(int): [-55, 55]
rotation = 0

# ...

bboxs = dora_event["value"].to_numpy()
self.bboxs = np.reshape(
    bboxs, (-1, 6)
)  # [min_x, min_y, max_x, max_y, conf, label]

+ # ChatGPT Addition
+ # Find the bbox with the highest confidence
+ target = max(self.bboxs, key=lambda x: x[4])
+ bbox_center_x = (target[0] + target[2]) / 2.0
+ rotation = np.clip(
+     int((bbox_center_x - CAMERA_WIDTH / 2)
+     * 55 / (CAMERA_WIDTH / 2)),
+     -55,
+     55,
+ )
```

Fig. 2. To illustrate code generation, the following code snippet is ChatGPT's response to the instructions: "rotate the robot according to bounding box". In this example, we asked ChatGPT to override a static variable to rotate a robot by setting the rotation of the robot according to the bounding box sent by an object detection node. ChatGPT is able to generate a linear function of the rotation proportional to the bounding box center. Video can be found here: youtu.be/NvvTEP8Jak8?t=229

IV. Results

Our LLM-powered hot-reloading functionality was able to pass most of the scenarios we set up, although most required multiple trials (Table I).

The failures were equally present in the three stages of generation (Fig. 3):

- Retrieval stage: Errors occurred when the expressed instruction did not provide enough context and was not similar enough to the source file that we wanted to modify. Being very precise in the instruction and having a well documented code with docstring and comments gave the best result.
- Generation stage: Generation failed when the LLMs assumed a wrong variable type or assumed the wrong definition of a function.
- Code logic stage: Implementation failed when the task was too complex or when there was not enough information about the required inputs.

TABLE I
FAILURE RATES DEPENDING ON SCENARIOS AND MODELS

Response type	Mistral 7B	ChatGPT
Modifying static variable	82%	98%
Changing function arguments	77%	90%
Adding control flow	N/A	57%
Adding new functionalities	N/A	43%

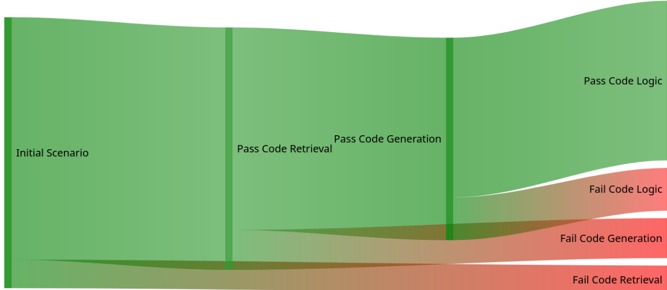

Fig. 3. Failure rates at different steps in generating code using the dora-robomaster

V. Discussion

Our method sets a new precedent in human-machine interaction as it provides the possibility to pass many new instructions to robots on any part of its codebase at runtime, from changing simple configuration to requesting whole new functionality.

VI. Future Work

Our methods still need refinement as our current failure rate is still quite high, but we have already identified many improvements to be made.

For the retrieval failure rate, we want to:

- Improve our retrieval and generation by extensively documenting each source code in order to limit retrieval and generation error.
- Benchmark additional embedding models.
- Reduce retrieval error by using a bigger context model such as Gemini Pro 1.5 [19] and feed more than one source code to the LLM and let the LLM decide which source code to modify.

For the generation and code generation failure rate, we want to:

- Benchmark alternative LLMs such as Claude 3 [20]
- Use LLM Agents to split the code generation task into several LLM Agents and condition each agent to a specific part of the problem [21].
- Implement an auto-debugging mechanism to feed errors trace-back into the LLMs and enable it to auto-correct coding syntax error [22].
- Fine-tune ChatGPT and Mistral 7B on existing validated data [11].
- Investigate prompt engineer and its influence on our failure rate [23].

In addition to what we have described above, we believe that there are clear next steps that can further improve interactions between human and machine:

- Integrate vision into the LLMs in order to be able to integrate visual element into the code [24].
- Improve the latency of the LLMs making the interaction faster using alternative inference engine provider such as groq.com [25] can reach speed of 500 tokens/s. Being able to interact faster also means that there can be more trials.
- Explore LLM capacity to self-discover problems and auto-implement solutions in the likes of Wang et al. self-driscovering Minecraft bot [26].
- Build and curate a dataset with all modal data (audio, text, code, and image) and ground truth, to optimize each step of the generation.
- Add function calling LLMs to interact with code at runtime.

VII. Conclusion

In this paper, we explained how using LLMs in conjunction with dora-rs, a hot-reloading robotic framework, can provide new ways to interact with a robot. By hot-reloading code changes and implementing a retrieval augmented code generation, we create the opportunity for users to implement instructions previously inaccessible. This method is still fairly error-prone but has yielded promising initial results, and many ways of improvement have already been identified.

References

[1] Jacky Liang et al. *Code as Policies: Language Model Programs for Embodied Control*. arXiv:2209.07753 [cs]. May 2023. DOI: 10.48550/arXiv.2209.07753. URL: http://arxiv.org/abs/2209.07753 (visited on 03/10/2024).

[2] *ROS: Home*. URL: https://www.ros.org/ (visited on 03/11/2024).

[3] Tanveer Ahmad, Zaid Al Ars, and H. Peter Hofstee. *Benchmarking Apache Arrow Flight – A wire-speed protocol for data transfer, querying and microservices*. arXiv:2204.03032 [cs]. Apr. 2022. DOI: 10.48550/arXiv.2204.03032. URL: http://arxiv.org/abs/2204.03032 (visited on 03/10/2024).

[4] *pyarrow.Array — Apache Arrow v15.0.1*. URL: https://arrow.apache.org/docs/python/generated/pyarrow.Array.html#pyarrow.Array.to_numpy (visited on 03/11/2024).

[5] *torch.Tensor.numpy — PyTorch 2.2 documentation*. URL: https://pytorch.org/docs/stable/generated/torch.Tensor.numpy.html (visited on 03/11/2024).

[6] Amos Wenger. *So you want to live-reload Rust*. en. Sept. 2020. URL: https://fasterthanli.me/articles/so-you-want-to-live-reload-rust (visited on 03/10/2024).

[7] Qiwei Peng, Yekun Chai, and Xuhong Li. *HumanEval-XL: A Multilingual Code Generation Benchmark for Cross-lingual Natural Language Generalization*. arXiv:2402.16694 [cs] version: 1. Feb. 2024. URL: http://arxiv.org/abs/2402.16694 (visited on 03/11/2024).

[8] Patrick Lewis et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. arXiv:2005.11401 [cs]. Apr. 2021. DOI: 10.48550/arXiv.2005.11401. URL: http://arxiv.org/abs/2005.11401 (visited on 03/10/2024).
[9] OpenAI et al. *GPT-4 Technical Report*. arXiv:2303.08774 [cs]. Mar. 2024. DOI: 10.48550/arXiv.2303.08774. URL: http://arxiv.org/abs/2303.08774 (visited on 03/10/2024).
[10] *GPT-4 - API Providers Performance & Price Analysis — Artificial Analysis*. en. URL: https://artificialanalysis.ai/models/gpt-4/providers (visited on 03/11/2024).
[11] *OpenAI Platform*. en. URL: https://platform.openai.com (visited on 03/11/2024).
[12] Albert Q. Jiang et al. *Mistral 7B*. arXiv:2310.06825 [cs]. Oct. 2023. DOI: 10.48550/arXiv.2310.06825. URL: http://arxiv.org/abs/2310.06825 (visited on 03/10/2024).
[13] *Mistral 7B - Quality, Performance & Price Analysis — Artificial Analysis*. en. URL: https://artificialanalysis.ai/models/mistral-7b-instruct (visited on 03/11/2024).
[14] *TheBloke/OpenHermes-2.5-Mistral-7B-GGUF · Hugging Face*. URL: https://huggingface.co/TheBloke/OpenHermes-2.5-Mistral-7B-GGUF (visited on 03/11/2024).
[15] Alec Radford et al. *Robust Speech Recognition via Large-Scale Weak Supervision*. arXiv:2212.04356 [cs, eess]. Dec. 2022. DOI: 10.48550/arXiv.2212.04356. URL: http://arxiv.org/abs/2212.04356 (visited on 03/11/2024).
[16] Shitao Xiao et al. *C-Pack: Packaged Resources To Advance General Chinese Embedding*. arXiv:2309.07597 [cs]. Dec. 2023. URL: http://arxiv.org/abs/2309.07597 (visited on 03/10/2024).
[17] Pinky Sitikhu et al. “A Comparison of Semantic Similarity Methods for Maximum Human Interpretability”. In: *2019 Artificial Intelligence for Transforming Business and Society (AITB)*. Vol. 1. Nov. 2019, pp. 1–4. DOI: 10.1109/AITB48515.2019.8947433. URL: https://ieeexplore.ieee.org/document/8947433 (visited on 03/11/2024).
[18] *RoboMaster S1 - DJI*. fr. URL: https://www.dji.com/fr/robomaster-s1 (visited on 03/10/2024).
[19] Machel Reid et al. *Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context*. arXiv:2403.05530 [cs]. Mar. 2024. URL: http://arxiv.org/abs/2403.05530 (visited on 03/11/2024).
[20] *Introducing the next generation of Claude*. en. URL: https://www.anthropic.com/news/claude-3-family (visited on 03/11/2024).
[21] Zhiheng Xi et al. *The Rise and Potential of Large Language Model Based Agents: A Survey*. arXiv:2309.07864 [cs]. Sept. 2023. DOI: 10.48550/arXiv.2309.07864. URL: http://arxiv.org/abs/2309.07864 (visited on 03/10/2024).
[22] Jialun Cao et al. *A study on Prompt Design, Advantages and Limitations of ChatGPT for Deep Learning Program Repair*. arXiv:2304.08191 [cs]. Apr. 2023. URL: http://arxiv.org/abs/2304.08191 (visited on 03/11/2024).
[23] Banghao Chen et al. *Unleashing the potential of prompt engineering in Large Language Models: a comprehensive review*. arXiv:2310.14735 [cs]. Oct. 2023. URL: http://arxiv.org/abs/2310.14735 (visited on 03/11/2024).
[24] Chenfei Wu et al. *Visual ChatGPT: Talking, Drawing and Editing with Visual Foundation Models*. arXiv:2303.04671 [cs]. Mar. 2023. URL: http://arxiv.org/abs/2303.04671 (visited on 03/11/2024).
[25] *GroqChat*. URL: https://groq.com/ (visited on 03/11/2024).
[26] Guanzhi Wang et al. *Voyager: An Open-Ended Embodied Agent with Large Language Models*. arXiv:2305.16291 [cs]. Oct. 2023. DOI: 10.48550/arXiv.2305.16291. URL: http://arxiv.org/abs/2305.16291 (visited on 03/11/2024).