Graph-Guided API Sequencing with Reinforcement Learning

Lakshmi Mandal*
Indian Institute of Science Bangalore, India
lmandal@iisc.ac.in

Balaji Ganesan
IBM Research, India
bganesa1@in.ibm.com

Avirup Saha
IBM Research, India
avirup.saha2@ibm.com

Renuka Sindhgatta
IBM Research, India
renuka.sindhgatta.rajan@ibm.com

Abstract

API sequencing involves the selection and execution of APIs, which can include REST APIs or function signatures, to achieve specific goals. Recent efforts have explored generating these sequences of API calls using Large Language Models in response to natural language utterances. In this work, we present a novel approach to API sequencing using Reinforcement Learning to determine the next API to add to an evolving sequence. Our approach leverages an API Graph that represents API endpoints as nodes and all possible call sequences as edges, allowing the RL agent to navigate and choose the next API based on the current sequence state. The agent repeats this process until a decision maker determines that no further additions are required or that the problem has become intractable. Our graph-based RL framework is designed to automate complex multi-step tasks through API sequencing.

1 Introduction

Representational State Transfer (REST) APIs are widely used in application development, but they often involve complex multi-step sequences to achieve specific goals. This knowledge of sequence of APIs to call is typically provided by human developers. But with the increasing use of Retrieval Augmented Generation (RAG) and tool calling systems, automated API sequencing has become a more relevant task.

Large Language Models (LLMs) have been explored for generating API sequences in response to natural language requests but they often suffer from inefficiencies in sequence generation. These models can produce incorrect or incomplete sequences. To address this, we propose a multi-agent system based on Reinforcement Learning (RL) that uses an API Graph to navigate and generate the optimal API sequence. Our approach is designed to automatically solve complex multi-step tasks by guiding the RL agent through a graph of API endpoints, selecting the most appropriate API at each step. By leveraging a graph-based structure, our approach offers enhanced control over the sequencing process, compared to methods that only rely on LLMs to do the sequencing.

Previous works have explored different aspects of API sequencing, but few have combined this task with RL. Tool use and function calling in language models has been extensively studied. [1] presented a survey on augmented LLMs, and works like [2] focused on supplementing LLMs with information retrieval for API function calling. Toolformer [3] fine-tunes an LLM on the task of function calling with some custom built tools. [4] teaches LLMs to use such tools with self-instruction. TaskMatrix [5] studied the problem of task completion using a large number of APIs. ToolLLM [6] is a general tool-use framework encompassing data construction, model training, and evaluation over 16,000 APIs from RapidAPI Hub. However, unlike these methods, our focus is on the sequential decision-making process using RL, which is less explored in the context of API sequencing.

L. Mandal et al., Graph-Guided API Sequencing with Reinforcement Learning (Extended Abstract). Presented at the Third Learning on Graphs Conference (LoG 2024), Virtual Event, November 26–29, 2024.

In order to experiment on API sequencing we need appropriate datasets. Fortunately, there are a number of works in this area. APIBench from Gorilla [2] consists of HuggingFace, TorchHub, and TensorHub APIs. RestBench from RestGPT [7] consists of APIs from TMDB movie database and Spotify music player. ToolBench from ToolLLM [6] consists of 16,464 real-world RESTful APIs spanning 49 categories from RapidAPI Hub. AnyToolBench from AnyTool [8] is similar to ToolBench but with a different evaluation protocol.

Agent-based frameworks have also been explored in this area. ReAct [9] studied the integration of reasoning and acting (by means of function calls) in LLM agents. Inspired by ReAct, RestGPT [7] proposes a dual-agent planner-executor approach to connect LLMs with real-world RESTful APIs. [10] introduced Autogen, an LLM based multi-agent framework for executing function calls. AnyTool [8] introduced self-reflective, hierarchical agents for API calling using the function calling ability of GPT-4 [11]. [12] introduced agents for infusing knowledge and reasoning with foundation models. These works inspire our multi-agent RL-based API sequencing system.

Graph Convolutional Networks [13] have been proposed to learn the reward function in Reinforcement Learning. [14] use GCNs for reward shaping by propagating messages from rewarding states to accelerate learning in reinforcement learning tasks. [15] proposed using reinforcement learning for goal-oriented next best activity recommendation which has a similar objective as our proposed method, except we are concerned with incorporating API information and Graph knowledge into the next state prediction using RL.

The contributions of our work are as follows:

- We propose a multi-agent system to have better and more accurate multi-hop API sequence generation and to execute further downstream tasks such as schema generation.
- We introduce an RL agent to orchestrate with other agents and optimize the interactions to achieve an optimal API sequence.

2 Graph Guided API Sequencing

Our approach relies on constructing an API Graph that serves as the foundational structure for sequencing APIs using Reinforcement Learning (RL). The graph-based RL system operates in a multi-agent environment to generate optimal API sequences for complex multi-step tasks.

2.1 API Graph Construction

In our approach, the core of the API sequencing problem is represented as a graph, where API endpoints are nodes, and the possible sequences of API calls form directed edges.

Initial API Graph Population We construct an initial API Graph for Spotify and TMDB by parsing the sequences of API calls from RestBench. Each API endpoint in these sequences becomes a node in the graph, and the connections between consecutive API calls form the directed edges. This forms a base-level graph where the existing API sequences are captured. However, this graph may not fully represent all possible API interactions or sequences due to the limited scope of the dataset.

Link Prediction To further populate the API Graph and infer potential API sequences that are missing from the dataset, we apply Graph Neural Network (GNN)-based link prediction. GNNs are effective for learning graph-structured data and can predict likely edges between nodes based on the structure of the existing graph. By training a GNN model on the initial API Graph, we can predict additional edges (i.e., API call sequences) that are not explicitly present in the RestBench ground truth. We use node embeddings generated from the GNN to encode the structural and semantic properties of each API endpoint. The GNN is trained to predict links between APIs that are likely to be sequenced together, effectively expanding the API Graph with potential new edges. This more complete graph serves as the foundation for the Reinforcement Learning (RL) agent, which will use this API Graph to explore and generate optimal API sequences for complex tasks.

API Graph in RL System Once the API Graph is fully populated, it is incorporated into our RL framework. The RL agent treats this graph as a navigation space, where the nodes represent the current state of the API sequence, and the edges represent possible next actions (API calls). The RL agent traverses this graph to sequence APIs in an optimal manner, selecting the next API based on the state of the current sequence and the structure of the graph.

Steps	Input	Output
1. Foundation Model Agent +RAG Agent	NLU	Obs1: Initial API, Feasible Actions: call RAG+ Foundation Model (FM), Oracle, Code agent
2. RL Agent	Obs1, Feasible Actions	Chosen Action: call FM+RAG Agent
3. Foundation Model Agent +RAG Agent	NLU, Obs1	Obs2: Next API, Feasible Actions: call RAG+ FM, Oracle, Code agent
4. RL Agent	Obs2, Feasible Actions	Chosen Action: call FM+RAG Agent
5. Foundation Model Agent +RAG Agent	NLU, Obs1+Obs2	Obs3: Next API, Feasible Actions: call RAG+ FM, Oracle, Code agent
6. RL Agent	Obs3, Feasible Actions	Chosen Action: call Oracle Agent
7. Oracle Agent	NLU, Obs1+Obs2+Obs3	Obs1+Obs2+Obs3 as sequence of API, Feasible Actions: call RAG+FM, Oracle, Code agent
8. RL Agent	sequence of API, Feasible Actions	Chosen Action: call Code Agent
9. Code Agent	NLU, sequence of API	Return Error or Result

Figure 1: Flow between agents in our Multi-Agent system for API sequencing.

2.2 Multi-Agent System

In addition to the API Graph, we employ a multi-agent system where the RL agent interacts with other specialized agents to generate API sequences:

- **RAG Agent:** Responsible for retrieving APIs based on natural language utterances, starting with an initial set of APIs from a vector database and later querying the API Graph.
- **LLM Agent:** Also known as the Foundation Model agent, as a retriever of relevant API information and assists the RL agent in interpreting the current sequence based on natural language instructions.
- **RL Agent:** The central agent that navigates the API Graph, choosing the next API in the sequence based on the current state of the graph. The RL agent uses GNN embeddings to inform its decisions, ensuring that the selected API transitions are contextually appropriate.
- **Code Agent:** Responsible for taking the generated optimal API sequence as input and producing a result or an error message which can be used by the RL Agent to decide the next step.
- **Oracle Agent:** Acting as decision maker, provides a feedback signal on whether the optimal API sequence generation is completed or not.

Figure 1 shows the flow of information between agents in our framework. The RL agent orchestrates the interactions between these agents, using the API Graph to determine the optimal path for API sequencing. The input to our system is a natural language utternace like *Make me a playlist containing three songs of Mariah Carey and name it 'Love Mariah'*. The expected output is a sequence of APIs which when executed will accomplish this task. Our multi-agent system starts by retrieving potential candidate APIs for this utterance. These APIs could be obtained based on their description using a vector database.

Now given the natural language utterance and one or more initial APIs, the task becomes one of identifying the next API to add to the sequence. The search space is constrained by the API Graph. An API can be called only if all its parameter values are available from the natural language utterance or the outputs of earlier APIs. An oracle agent or a reasoning based decision maker agent like the one described in [12] can assist in determining if we have reached the end state. The main challenge then is determining what is the next best API to call given the current state. We propose an RL solution for this which we describe next.

2.3 Reinforcement Learning for API Sequencing

We describe the RL environment for API sequencing followed by our policy optimization solution.

- **State Representation:** The state is represented by sentence embeddings of the natural language utterance combined with the GNN embeddings of the APIs that have been sequenced so far. This combined representation captures both the intent of the task and the structure of the API Graph, ensuring that the state evolves as new APIs are added to the sequence.
- **Action Representation:** Our action space consists of available APIs, constrained by the current state of the API Graph. An API can be selected only if all its required parameters are available from the natural language utterance or previous API outputs.
- **Reward Structure:** A positive reward is provided if the current state is the goal state; otherwise, a reward of zero is given for non-goal states. In this framework, the Oracle Agent provides feedback on whether the task is complete, indicating if the goal state has been achieved.

We view generating an optimal API sequence for a given natural language utterance, as a sequential decision-making task and formulate it as a reinforcement learning (RL) problem [16]. A standard RL setting requires the agent to interact with the RL environment by observing its state and selecting an action. The agent then observes the next state and receives a reward. The goal of the agent is to select actions that maximize its long-term rewards.

The agent-environment interaction is modeled as a Markov Decision Process (MDP) [17] that can be represented via a five-tuple $(\mathcal{S}, \mathcal{A}, r, P, \gamma)$, where $\mathcal{S}, \mathcal{A}, r, P, \gamma$ stand for the state space, the action space, the reward function, the probability transition matrix, and the discount factor $(0 < \gamma < 1)$, respectively. In our formulation, the state space $\mathcal{S}$ consists of a combination of sentence embeddings derived from the natural language utterance and GNN embeddings of the API Graph. The action space, $\mathcal{A}$, consists of the available APIs in the domain, constrained by the API Graph. Each action corresponds to selecting the next API from the graph, where the selection is constrained by the availability of required parameters. The reward function $r(s_t, a_t)$ provides a positive reward when the selected API sequence progresses toward the task described by the natural language utterance.

We assume a finite action space, i.e., the total number of actions is finite. Entries of the transition probability matrix are in the form of $P(s' \mid s, a)$, the probability of transition from the current state s to the next state s' when action a is chosen according to a policy π, where $\pi : S \to \Delta(\mathcal{A})$ denotes a mapping from the state space to distributions over the set of actions feasible in state s. The discount factor plays the role of providing more weight to the current reward in comparison to rewards that are likely to come in the future.

We assume a model-free RL setting, where the system lacks explicit model information, relying on (state, action, reward, next state) data samples, denoted as $(s_t, a_t, r(s_t, a_t), s_{t+1})$. We use Proximal Policy Optimization (PPO) [18] to compute the optimal value and optimal policy, respectively. In PPO, policy evaluation (Eq. (1)) is performed by the critic and policy improvement (Eq. (2)) is performed by the actor. In Eq. (1), v_π is the long-term reward for a given policy π, and the initial state s_0 is drawn from the initial distribution ψ_0. In Eq. (2), π^* represents the optimal policy and Π is the set of all policies.

$$v_\pi = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r\left(s_t, a_t\right) \mid s_0 \sim \psi_0, a_t \sim \pi, \forall t\right], \tag{1}$$

$$\pi^* = \arg\max_{\pi \in \Pi} v_\pi. \tag{2}$$

Conclusion

In this work, we propose a novel approach for API sequencing using a reinforcement learning (RL) framework guided by an API Graph. We propose an API Graph constructed from API benchmarks like RestBench, using Graph Neural Networks (GNNs) to predict missing links and complete the API Graph, and a multi-agent system to generate and execute optimal API sequences. The RL agent uses GNN embeddings and natural language input to make informed decisions at each step of the sequence. By leveraging Proximal Policy Optimization (PPO), we train the RL agent to navigate the API Graph and find the most appropriate API call based on the current sequence state. Our method is expected to enhance the accuracy and efficiency of API sequencing, providing a scalable solution for automating complex multi-step tasks.

References

[1] Grégoire Mialon, Roberto Dessi, Maria Lomeli, Christoforos Nalmpantis, Ramakanth Pasunuru, Roberta Raileanu, Baptiste Roziere, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. Augmented language models: a survey. *Transactions on Machine Learning Research*, 2023. 1

[2] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*, 2023. 1, 2

[3] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36, 2024. 1

[4] Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. Gpt4tools: Teaching large language model to use tools via self-instruction. *Advances in Neural Information Processing Systems*, 36, 2024. 1

[5] Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, et al. Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis. *arXiv preprint arXiv:2303.16434*, 2023. 1

[6] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023. 1, 2

[7] Yifan Song, Weimin Xiong, Dawei Zhu, Wenhao Wu, Han Qian, Mingbo Song, Hailiang Huang, Cheng Li, Ke Wang, Rong Yao, et al. Restgpt: Connecting large language models with real-world restful apis. *arXiv preprint arXiv:2306.06624*, 2023. 2

[8] Yu Du, Fangyun Wei, and Hongyang Zhang. Anytool: Self-reflective, hierarchical agents for large-scale api calls. *arXiv preprint arXiv:2402.04253*, 2024. 2

[9] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022. 2

[10] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*, 2024. 2

[11] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 2

[12] Debarun Bhattacharjya, Junkyu Lee, Don Joven Agravante, Balaji Ganesan, and Radu Marinescu. Foundation model sherpas: Guiding foundation models through knowledge and reasoning. *arXiv preprint arXiv:2402.01602*, 2024. 2, 3

[13] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2022. 2

[14] Martin Klissarov and Doina Precup. Reward propagation using graph convolutional networks. *Advances in Neural Information Processing Systems*, 33:12895–12908, 2020. 2

[15] Prerna Agarwal, Avani Gupta, Renuka Sindhgatta, and Sampath Dechu. Goal-oriented next best activity recommendation using reinforcement learning. *arXiv preprint arXiv:2205.03219*, 2022. 2

[16] Richard S. Sutton and Andrew G. Barto. Reinforcement learning : An introduction, 2'nd ed. *Cambridge, MA: MIT Press*, 2:526, 2018. 4

[17] Martin L Puterman. Markov decision processes: discrete stochastic dynamic programming. *John Wiley & Sons*, 2014. 4

[18] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. 4