Large Language Models Can Learn Representation in Natural Language

Yiduo Guo[1], **Yaobo Liang**[2], **Dongyan Zhao**[1,3,4], **Nan Duan**[2]
[1]Wangxuan Institute of Computer Technology, Peking University
[2]Microsoft Research Asia
[3]State Key Laboratory of Media Convergence Production Technology and Systems
[4]Artificial Intelligence Institute of Peking University
`yiduo@stu.pku.edu.cn`, `justinzzk@stu.pku.edu.cn`, `yaobo.liang@microsoft.com`
`zhaody@pku.edu.cn`, `nanduan@microsoft.com`

Abstract

One major challenge for Large Language Models (LLMs) is completing complex tasks involving multiple entities, such as tool APIs. To tackle this, one approach is to retrieve relevant entities to enhance LLMs in task completion. A crucial issue here is obtaining accurate natural language representations for each entity to aid in retriever precision. In this paper, we propose the Natural Language Representation Optimization Problem, which aims to refine entity descriptions for improved retrieval and LLM utilization. We introduce the Learning to Represent with Natural Language method, which utilizes LLMs to optimize entity representations consisting of text patterns based on environmental feedback. We iteratively prompt LLMs to enhance or adjust patterns based on entity samples and evaluate their effectiveness through environmental feedback. Our method successfully learns human-understandable representations for classification tasks (e.g., instructions and documents) and API call tasks (e.g., APIbench and Virtual Home), significantly improving GPT-4's task performance.

Task description: Given a user instruction, the LLM calls API from the huggingface API set to finish the instruction.
Retriever: Parametric retriever and non-parametric retriever (e.g., BM25).

Entities (APIs) and their plain natural language representation: FacebookAI /roberta-large (Natural Language Processing Feature Extraction);
facebook/dino-vitb16 (Computer Vision Image Classification);
microsoft/codebert-base (Multimodal Feature Extraction);
...

User instruction: Write an API implementation that takes customer reviews as input and extracts features to analyze customer sentiment. **Label:** APIs of the natural language processing feature extraction domain (e.g., FacebookAI /roberta-large).
The natural language representation of APIs of the natural language processing feature extraction domain learned by our method: *Pattern 1:* Language-Specific Processing - Adapting feature extraction techniques to handle text in a specific language or multiple languages (e.g., 'Extract some features from a Korean text'). *Pattern 2:* Text Analysis Techniques This pattern includes the methods or techniques used for feature extraction and text analysis (e.g., 'create contextual embeddings for text'). *Pattern 3:* This pattern is about extracting features from text to determine sentiment or emotion, often used in analyzing customer reviews (e.g., 'provide the overall sentiment'). *Pattern 4:* Feature Extraction for Classification and Categorization This pattern involves the extraction of features from text to be used in classification models, such as support vector machines (SVMs) or categorization based on similarity (e.g., 'build a classifier to identify the most suitable applicants').

Table 1: We list typical elements of the APIBench task.

1 Introduction

Large Language Models (LLMs), such as GPT-4 (OpenAI, 2023), LLaMa-2 (Touvron et al., 2023), and Gemini (Team et al., 2023), successfully achieved strong zero-shot/few-shot performance on various natural language tasks (Qin et al., 2023a; Frieder et al., 2023; Kojima et al., 2022; Zhong et al., 2023; Hendrycks et al., 2020). To complete more challenging tasks that requires operating multiple tools (Guo et al., 2023c; Zhou et al., 2023; Liu et al., 2023; Mao et al., 2023), LLMs need to understand and use these new entities (e.g., classes and tools). To address these tasks, a straightforward way is to fine-tune LLMs on the training corpus (Patil et al., 2023; Qin et al., 2023b), which needs many GPUs. (Ovadia et al., 2023; Zhang et al., 2023) retrieves relevant entities from all entities and then uses the retrieved entities to augment the LLM for task completion (Ovadia et al., 2023; Zhang et al., 2023). However, plain natural language representations of entities do not always make the retriever capture the complex relation between entities and the task instruction, causing inaccurate retrieved results.

To address the poor entity representation, we propose the Natural Language Representation Optimization Problem, which seeks the best representations of entities for complex task completion. Cur-

Findings of the Association for Computational Linguistics: ACL 2024, pages 9145–9154
August 11-16, 2024 ©2024 Association for Computational Linguistics

rently, users must experiment with various representations to find the most suitable, lacking knowledge of compatibility with specific retrievers and LLMs. To streamline this process, we introduce the Learning to Represent with Natural Language method, leveraging LLMs to automatically optimize representations. Our method aims to learn entity representations comprising multiple patterns, each describing a unique property. It operates in two main stages: (1) Multi-step iterations to refine entity representations. (2) Within each iteration, LLMs are prompted to add new patterns and edit existing ones based on entity samples, evaluating effectiveness through feedback.

We demonstrate our method's learned representation alongside typical elements of the API call task in Table 1. Compared to plain natural language representation, our learned representation offers 4 distinct patterns for describing API usage, each encapsulating general API usage scenarios with typical examples. These patterns, derived from user instructions, aid in improved retrieval and usage of the Language Model (LLM). Additionally, the learned representation is human-readable and amenable to expert editing. Experimental results on classification and API call tasks show that our method significantly enhances the retrieval performance and the GPT-4's task performance.

2 Related Work

Large Language Models and Prompt Optimization (e.g., GPT3 (Brown et al., 2020), GPT-4 (OpenAI, 2023), Gemini (Team et al., 2023), and LLaMA (Touvron et al., 2023)) can solve various basic natural language tasks (Qin et al., 2023a), write mathematical proof draft (Jiang et al., 2022), and pass human standard examinations (Zhong et al., 2023). Furthermore, with the help of tools (e.g., APIs (Liang et al., 2023) and search engine (Schick et al., 2023)), LLMs can complete PPT tasks (Guo et al., 2023c) and control operating system (Liu et al., 2023). Prompting optimization methods further improve the LLM's performance by optimizing the input prompt based on feedback. APE (Zhou et al., 2022) and Instruction Induction (Honovich et al., 2022) update task instruction by selecting the best instruction candidate based on the validation performance. Learning to plan (Guo et al., 2023b) optimizes the task plan consisting of step-by-step solutions based on the error made by LLMs. EvoPrompt (Guo et al., 2023a) iteratively optimizes the prompts based on the evolutionary algorithms.

Representation Learning in Pretrained Language Models The recent mainstream paradigm for natural language representation learning is to first pre-train the language model on a large-scale corpus with self-supervised losses like auto-regressive loss and masked language modeling loss (Devlin et al., 2019). Analysis experiments show that the pretrained models (e.g., GPT (Radford and Narasimhan, 2018), RoBERTa (Liu et al., 2019b), and T5 (Raffel et al., 2019)) can learn rich linguistic knowledge (Tenney et al., 2019b; Liu et al., 2019a; Tenney et al., 2019a) and world knowledge (Petroni et al., 2019; Bouraoui et al., 2019; Feldman et al., 2019) during the pre-training process. However, learning natural language representations of entities of the task has not been investigated.

3 Natural Language Representation Optimization Using LLMs

We start with a task defined by a dataset $D_{train} = (Q, A)$, where Q represents the input query and A the corresponding answer containing entities e (e.g., a class or API sequence). Typically, to tackle this task, we employ a retriever R to select an entity e from a set S that relates to the query Q. This selection is based on ranking the similarity between the query and the natural language descriptions p_e of entities in S. This process can be formalized as $\hat{e} = R(Q, P)$, where P comprises the natural language representations of all entities, and $\hat{e}$ is the retrieved entity. Subsequently, this approach enriches the Language Model (LM) M with the retrieved $\hat{e}$ to generate an answer $M(\hat{e})$ for completing Q. The objective of optimizing natural language representations is to find a set P such that R can accurately retrieve relevant entities based on P, and $M(\hat{e}) = A$. Formally, we formulate this task as an optimization problem aiming to maximize the expectation of a per-sample score $f(M(\hat{e}), Q, A) = f(M(R(Q, P)), Q, A)$ over possible (Q, A).

$$P^* = \arg\max_P \mathbb{E}(Q, A)[f(M(R(Q, P)), Q, A)] \tag{1}$$

4 Method

In this section, we present our method, "Learning to Represent with Natural Language," for automatically optimizing the natural language representation P. Our method features two key aspects: (1) Each entity's representation is managed as a list of patterns describing its properties, which are updated via add and edit operations. (2) Inspired by APE (Zhou et al., 2022), our approach employs an iteration-based optimization strategy to discover the optimal representation. Each iteration involves the following three steps.

4.1 Collecting Incorrect Samples Based on Current Representation

To iteratively optimize the representation, our method first identifies incorrect samples unresolved by the current iteration's representation. Specifically, at the t-th training iteration, for each entity e, we collect its false negative samples $S^e_{FN} = \{(Q, A)|e \notin M(\hat{e}) \cap e \in A\}$ from the training dataset $D = (Q, A)$, where the retriever and LLM generate answers lacking necessary entity e based on the current representation P_t. Similarly, we gather false positive samples $S^e_{FP} = \{(Q, A)|e \in M(\hat{e}) \cap e \notin A\}$ from the training dataset, where the retriever R and LLM M mistakenly include e in the generated answer $M(\hat{e})$ for query Q based on representation P_t.

4.2 Adding New Patterns and Editing Existing Pattern

For a given entity e and its current representation P^e_t, the retriever R might not capture the relevance between P^e_t and some samples of e, resulting in false positives, S^e_{FN}. To address this, we prompt the LLM to derive new patterns, p^e_t, from S^e_{FN}, aiming to enhance the relevance between S^e_{FN} and P^e_t by adding p^e_t to P^e_t. We illustrate this addition prompt in Figure 1. Additionally, the retriever R may incorrectly assign high relevance between samples of other entities and P^e_t, leading to false positive samples, S^e_{FP}. To mitigate this, we prompt the LLM to edit the existing patterns of P^e_t, resulting in $\overline{P}^e_t$, to decrease the relevance between P^e_t and SFP^e. We demonstrate this editing prompt in Figure 2, where we request the LLM to modify words and phrases in the patterns to reduce erroneous high relevance while preserving the representativeness of e's properties.

The addition prompt

Here is a retrieval task. The following samples are about the entity: <entity name>. To help the user distinguish the sample of this entity from samples from other entities, you can extract essential and core patterns from samples of this entity. Task description
The essential and core patterns always appear in the following samples and can represent the key. (Representative)
Each pattern should have a short and general description, together with 2$\tilde{3}$ representative, short, and key cases. For the cases, you must only copy the original words or phrases (nouns and verbs) from the samples as the cases. You must not directly copy the whole sample as your cases. (pattern form)
Similar cases should be put together with a general form. Any two patterns should be exclusive. (Unique)
You need to find essential and core patterns as much as possible. (Holistic)
You can list the patterns in a list. Samples: <False Negative Samples>

Figure 1: Prompt for adding new pattern.

4.3 Verifying the Effectiveness of Patterns Based on Feedback

To assess the effectiveness of newly generated patterns p^e_t and edited patterns $\overline{P}^e_t$, we evaluate whether updating P_t with p^e_t and SFP^e improves the average validation score $f(M(R(Q, P)), Q, A)$. Specifically, We first insert p^e_t into the original representation P^e_t of entity e and compare the average validation scores before and after the update. If the updated score is higher than a threshold, we consider p^e_t an effective update to P_t; otherwise, we maintain P^e_t as the original representation. Similarly, we evaluate the effectiveness of $\overline{P}^e_t$ by replacing P^e_t with $\overline{P}^e_t$ and comparing validation performance. After verification, we obtain the new representation P^e_{t+1} updated with the new patterns p^e_t and edited patterns $\overline{P}^e_t$ if they demonstrate an improvement in validation performance.

The edition prompt

Here's an edit task. The following patterns pertain to the entity: <entity name>. However, some patterns mistakenly classify samples of other entities as belonging to <entity name>, which we refer to as 'wrong samples.'
Your task is to modify the current patterns in order to reduce their similarity to the wrong samples, but these modified patterns should still accurately represent the key pattern for 0.
To edit the pattern, you must re-write important words or phrases that are similar to the wrong samples, and delete irrelevant words in the pattern for the key.
You should try your best to make the pattern different from the wrong samples. Patterns without wrong samples should not be edited.
Once you have made the necessary edits, please compile a list containing all the patterns, ensuring that the number of patterns remains the same as before.
<Previous patterns>
<Wrong samples>

Figure 2: Prompt for editing existing patterns.

4.4 Put it together

At iteration t, our method applies the three steps to each entity in set S one by one to obtain new entities' representation P_{t+1}. Optimization continues until reaching the maximum iteration or no improvement in validation performance over the last three iterations. We illustrate our method's algorithm procedure in Appendix B.

5 Experiments

Datasets For **Classification Tasks**, their entities are classes and we consider (1) the banking intent classification task Casanueva et al. (2020) (2) the web of science document classification task Kowsari et al. (2017) (WOS5736). For **API call Tasks**, their entities are APIs and we consider (1) the APIbench benchmark (Patil et al., 2023). It has three sub-tasks consisting of APIs extracted from huggingface, torchhub, and tensorflow respectively. (2) the virtual-home task (Puig et al., 2018) where the LLM needs to call and generate the API sequence to finish the instruction. In the test process, we follow Patil et al. (2023); Xu et al. (2023) and report the Longest Common Sub-sequence (LCS) score for VirtualHome and accuracy performance for other tasks. More details are in Appendix C

Model and retrievers: We utilize the GPT-4 model (GPT-4-turbo) as the LLM to balance performance and the inference cost. For the virtual home task, following Xu et al. (2023), we employ the BM25 retriever, treating each API as a separate document. Retrieval involves searching the index and fetching the most relevant APIs based on instructions. For other tasks, each API/class is indexed as an individual document and embedded using the openai text-embedding-ada-002 API. Relevant APIs/classes are retrieved based on cosine similarity between their embeddings and the user instruction or passage.

Baselines and Hyper-parameters (1) 'Retriever-only', which only uses the retriever to retrieve the most relevant entities for the instruction. Each entity uses its plain natural language representation as its document. (2) 'Retriever top k + LLM selection', which uses the retriever to retrieve the most relevant top k entities for the instruction and then prompts the LLM to select the necessary entities from the k entities (See the prompt in Figure 3). (3) Automatic Prompt Engineer Liu et al. (2021b), which iteratively updates the representations by generating representation candidates of each entity and then selecting the candidate with the best validation performance. (4) 'Retriever+learning to represent', which follows (1) but replaces plain representations with learned representations by our method. (5) 'Retriever top k + learning to represent+ LLM selection', which follows (2) but using the learned representations by our method. Details and hyperparameters are in Appendix D.

5.1 Main results

The results in Table 2 show: (1) Our method significantly enhances the retriever's performance, e.g., from 35 (plain representation in 'Retriever only') to 68.4 (learned representation in 'Retriever+learning to present') on the Hugging Face API task. (2) The LLM can utilize the learned representation to make decisions, achieving the best performance

Task	Banking77	WebOfScience	HuggingFace	TorchHub	Tensorflow	VirtualHome
Retriever-only	56.3	0.2	35.0	29.7	41.5	21.0
Retriever top k + LLM selection	69.7	0.2	56.7	62.4	67.4	23.1
APE (Zhou et al., 2022)	83.9	66.2	64.8	71.0	76.4	22.0
Retriever+learning to represent	85.8	75.2	68.4	61.9	68.0	23.6
Retriever top k + learning to represent+ LLM selection	**91.2**	**75.8**	**80.5**	**74.7**	**89.9**	**26.8**

Table 2: Performance of the 6 tasks. 'Huggingface', 'TorchHub', and 'Tensorflow' are 3 sub-tasks of the APIBench.

Task	Banking77	HuggingFace	VirtualHome
w/o addition	61.3	42.2	22.0
w/o edition	87.7	75.2	24.7
iteration number 5	63.9	73.2	23.2
iteration number 10	91.2	80.5	26.8
iteration number 15	92.2	81.8	26.5
threshold 0	89.2	77.5	25.8
threshold 0.01	91.2	80.5	26.8
threshold 0.05	90.2	81.2	26.2

Table 3: Ablation study of our method and report the performance based on 'Retriever top k + learning to represent+ LLM selection'.

across all baselines on the 6 tasks ('Retriever top k + learning to represent+ LLM selection'). Interestingly, despite the web-of-science task providing only class indices (e.g., 1, 2, 3, ...), our method still can learn general representations from samples to improve task performance.

The LLM selection prompt

Which one is the most suitable API/Class to complete the user's instruction? <Instruction>
for the vituralhome task, 'which APIs are necessary to complete the user's instruction? <Instruction>'.
You can refer to the patterns of the option. Then you must follow this pattern to output only the letter: " I choose the option [Letter].
<Option A> Natural language Representation of option A … <Option E> Natural language Representation of option E

Figure 3: Prompt for choosing API.

Finetuning VS Learning to represent Patil et al. (2023) fine-tunes the LLaMa-7b model on the APIbench training set, achieving accuracy performances of 55.6, 67.3, and 61.8 on the Hugging Face, TensorFlow, and TorchHub APIbench tasks, respectively. Additionally, the fine-tuned RoBERT model in Shao et al. (2023) attains 88.6 accuracy in the banking77 task. In comparison, our method achieves superior performance with human-understandable natural language representation.

5.2 Analysis

Case study of the learned represetations We compare the representations directly generated from GPT-4 and the learned representation in Table 4. Our learned representation uses more detailed words and aligns closely with the task samples. More examples are in Table 5.

Ablation study We conduct the ablation study on each component of our method as shown in Table 3. Our findings are: (1) Addition and edition operations are crucial. (2) Increased iterations improve performance but escalate inference cost; thus, we set the iteration number as 10. (3) A higher threshold filters out noisy updates but risks losing useful ones; hence, we set the threshold at 0.01.

6 Conclusion

This paper focuses on challenging tasks requiring processing multiple entities for completion. We propose the natural language representation optimization problem to find the optimal entity representation for improved retrieval and LLM utilization. We introduce the learning to represent with natural language method, which automatically optimizes representation based on feedback. Experiments demonstrate our method boosts the LLM's task performance significantly, along with high-quality entity natural language representation.

7 Limitations and potential risks

While our method shows notable enhancements in tasks like classification and API calls, it does have limitations:

It assumes access to a pre-existing training set, making deployment in open-domain settings challenging. We plan to address this in future research.

Theoretical guarantees and convergence analysis for our method are lacking. We aim to investigate this in future studies.

We do not foresee significant risks, as our experiments utilize public datasets and our method is tailored to classification and API call tasks.

References

Zied Bouraoui, José Camacho-Collados, and Steven Schockaert. 2019. Inducing relational knowledge from bert. In *AAAI Conference on Artificial Intelligence*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Iñigo Casanueva, Tadas Temcinas, Daniela Gerz, Matthew Henderson, and Ivan Vulic. 2020. Efficient intent detection with dual sentence encoders. In *Proceedings of the 2nd Workshop on NLP for ConvAI - ACL 2020*. Data available at https://github.com/PolyAI-LDN/task-specific-datasets.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics*.

Joshua Feldman, Joe Davison, and Alexander M. Rush. 2019. Commonsense knowledge mining from pretrained models. *ArXiv*, abs/1909.00505.

Simon Frieder, Luca Pinchetti, Ryan-Rhys Griffiths, Tommaso Salvatori, Thomas Lukasiewicz, Philipp Christian Petersen, Alexis Chevalier, and Julius Berner. 2023. Mathematical capabilities of chatgpt. *arXiv preprint arXiv:2301.13867*.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2023a. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532*.

Yiduo Guo, Yaobo Liang, Chenfei Wu, Wenshan Wu, Dongyan Zhao, and Nan Duan. 2023b. Learning to program with natural language. *arXiv preprint arXiv:2304.10464*.

Yiduo Guo, Zekai Zhang, Yaobo Liang, Dongyan Zhao, and Duan Nan. 2023c. Pptc benchmark: Evaluating large language models for powerpoint task completion. *arXiv preprint arXiv:2311.01767*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Or Honovich, Uri Shaham, Samuel R Bowman, and Omer Levy. 2022. Instruction induction: From few examples to natural language task descriptions. *arXiv preprint arXiv:2205.10782*.

Albert Q Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. 2022. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. *arXiv preprint arXiv:2210.12283*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Kamran Kowsari, Donald E Brown, Mojtaba Heidarysafa, Kiana Jafari Meimandi, , Matthew S Gerber, and Laura E Barnes. 2017. Hdltex: Hierarchical deep learning for text classification. In *Machine Learning and Applications (ICMLA), 2017 16th IEEE International Conference on*. IEEE.

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

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2021a. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*.

Nelson F. Liu, Matt Gardner, Yonatan Belinkov, Matthew E. Peters, and Noah A. Smith. 2019a. Linguistic knowledge and transferability of contextual representations. *ArXiv*, abs/1903.08855.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2023. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv: 2308.03688*.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021b. Gpt understands, too. *arXiv preprint arXiv:2103.10385*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019b. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692.

Jiageng Mao, Junjie Ye, Yuxi Qian, Marco Pavone, and Yue Wang. 2023. A language agent for autonomous driving. *ArXiv*, abs/2311.10813.

OpenAI. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Oded Ovadia, Menachem Brief, Moshik Mishaeli, and Oren Elisha. 2023. Fine-tuning or retrieval? comparing knowledge injection in llms. *arXiv preprint arXiv:2312.05934*.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *ArXiv*, abs/2305.15334.

Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H. Miller, and Sebastian Riedel. 2019. Language models as knowledge bases? *ArXiv*, abs/1909.01066.

Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. 2018. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8494–8502.

Chengwei Qin, Aston Zhang, Zhuosheng Zhang, Jiaao Chen, Michihiro Yasunaga, and Diyi Yang. 2023a. Is chatgpt a general-purpose natural language processing task solver? *arXiv preprint arXiv:2302.06476*.

Yujia Qin, Shi Liang, Yining Ye, Kunlun Zhu, Lan Yan, Ya-Ting Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Marc H. Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023b. Toolllm: Facilitating large language models to master 16000+ real-world apis. *ArXiv*, abs/2307.16789.

Alec Radford and Karthik Narasimhan. 2018. Improving language understanding by generative pretraining.

Colin Raffel, Noam M. Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.

Yijia Shao, Yiduo Guo, Dongyan Zhao, and Bing Liu. 2023. Class-incremental learning based on label generation. *arXiv preprint arXiv:2306.12619*.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019a. Bert rediscovers the classical nlp pipeline. In *Annual Meeting of the Association for Computational Linguistics*.

Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R. Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel R. Bowman, Dipanjan Das, and Ellie Pavlick. 2019b. What do you learn from context? probing for sentence structure in contextualized word representations. *ArXiv*, abs/1905.06316.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. 2023. On the tool manipulation capability of open-source large language models.

Peitian Zhang, Shitao Xiao, Zheng Liu, Zhicheng Dou, and Jian-Yun Nie. 2023. Retrieve anything to augment large language models. *arXiv preprint arXiv:2310.07554*.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. 2023. Agieval: A human-centric benchmark for evaluating foundation models. *arXiv preprint arXiv:2304.06364*.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*.

A Prompts of our method

We illustrate the addition prompt and the edition prompt of our method in Figure 1 and Figure 1 respectively.

B Algorithm procedure

We illustrate our method in Algorithm.1.

C Task details

For **Classification Tasks**, their entities are classes and we consider (1) the banking intent classification task Casanueva et al. (2020) where the LLM needs to classify instructions (sentences) belonging to 77 different intents. (2) the web of science task Kowsari et al. (2017) (WOS5736) where the LLM needs to classify passages belonging to 11 different science categories. For **API call Tasks**, their entities are APIs and we consider (1) the APIbench benchmark Patil et al. (2023) where the LLM needs to call the correct API to finish the instruction. It has three sub-tasks consisting of APIs extracted

Algorithm 1 Our Method

```
procedure OPTIMIZE($P$: Entities' representations)
    $P_0 \leftarrow$ plain natural language representations
    $f_{val} \leftarrow$ validation performance with $P_0$
    iteration $t \leftarrow 0$
    while iteration $t <$ maximum iteration do
        for each entity $e \in S$ do
            false negative samples $S^e_{FN}$, false positive samples $S^e_{FP} \leftarrow$ collecting incorrect samples from dataset $D$ with $P_t$
            new pattern $p^e_t \leftarrow$ LLM$M, S^e_{FN}$
            Edited patterns $\overline{P}^e_t \leftarrow$ LLM$M, P^e_t, S^e_{FP}$
            $f_{add} \leftarrow$ validation performance with $P_t + p^e_t$ (add $p^e_t$ to $P^e_t$ )
            if $f_{add} > f_{val}$+threshold then
                $P_t \leftarrow P_t + p^e_t$
                $f_{val} \leftarrow f_{add}$
            end if
            $f_{edit} \leftarrow$ validation performance with $P_t$ edited by $\overline{P}^e_t$ (replace $P^e_t$ with $\overline{P}^e_t$ )
            if $f_{edit} > f_{val}$+threshold then
                $P_t \leftarrow$P$_t$ edited by $\overline{P}^e_t$
                $f_{val} \leftarrow f_{edit}$
            end if
        end for
        iteration $t \leftarrow$ iteration $t + 1$
    end while
    return $P_t$
end procedure
```

from huggingface, torchhub, and tensorflow respectively. (2) the virtual-home task Puig et al. (2018) where the LLM needs to call and generate the API sequence to finish the instruction.

D Details of baselines and hyper-parameters

In (1) 'retriever-only', for the virtual home task, we follow Xu et al. (2023) and retrieve the top 10 relevant APIs to augment the LLM. For other tasks, we choose the most relevant API as the final answer. In (2) 'retriever top k + LLM selection', the selected entities by the LLM can be used to augment the LLM (virtual home) or as the final answer (classification task). We set k as 5 here. In (3) kNN selection method Liu et al. (2021a), for the virtual home task, APIs from the labels of these samples augment LLM; for other tasks, the majority class of labels from k samples is chosen as the final answer. This baseline utilizes the whole training set. We set k as 5 here. (4) Automatic Prompt Engineer (APE) Liu et al. (2021b) that iteratively updates the representations by generating representation candidates of each entity and then selecting the candidate with the best validation performance as the entity's representation. In the test process, we follow (2) except for replacing the plain natural language representation of each entity with the learned representation by the APE method for retrieval. This baseline utilizes the training set to iteratively optimize the representations.

Hyperparamters: In the learning process, we randomly split the original training set into 4:1 for constructing the training set and the validation set respectively. We use the training dataset D to find the optimal representations and test methods' performance on the original test dataset. We use the average F1 score over all entities as the score f in the verification process and set the threshold as 0.01 (Sec. 4.4). We set the maximum iteration as 10. We use the 1201 version of GPT-4-turbo and set its temperature as 0. We follow the official code of APE Liu et al. (2021b) to conduct our experiment. It also has 10 iterations. At each iteration, we select the best candidate from 5 candidates for each entity.

E Case studies

The prompt we used for prompting GPT4 to generate the natural language representation To help the user distinguish the sample of this API: API for natural language processing feature extraction domain from samples from other APIs, you can extract essential and core patterns from samples of this entity.
The essential and core patterns always appear in the following samples and can represent the key.
Each pattern should have a short and general description, together with 2 3 representative, short, and key cases.
Similar cases should be put together with a general form.
Any two patterns should be exclusive.
You need to find essential and core patterns as much as possible.
You can list the patterns in a list **The natural language representation of APIs of the natural language processing feature extraction domain generated by the LLM:** *Pattern 1:* Part-of-Speech (POS) Tagging:
Description: Assigning grammatical tags to each word in a sentence indicating its syntactic role. Cases: Input: "The cat sat on the mat." Output: [("The", "DT"), ("cat", "NN"), ("sat", "VBD"), ("on", "IN"), ("the", "DT"), ("mat", "NN"), (".", ".")] Input: "She sells seashells by the seashore." Output: [("She", "PRP"), ("sells", "VBZ"), ("seashells", "NNS"), ("by", "IN"), ("the", "DT"), ("seashore", "NN"), (".", ".")]
Pattern 2: Named Entity Recognition (NER):
Description: Identifying and classifying named entities such as persons, organizations, or locations. Cases: Input: "Apple is headquartered in Cupertino, California." Output: [("Apple", "ORG"), ("Cupertino", "LOC"), (",", "O"), ("California", "LOC"), (".", "O")] Input: "Harry Potter is a series of fantasy novels written by J.K. Rowling." Output: [("Harry Potter", "WORK$_{O}F_{A}RT$"), (" $J.K.Rowling$", " $PERSON$")]
Pattern 3: Sentiment Analysis:
Description: Determining the sentiment or emotional tone expressed in a piece of text. Cases: Input: "I love this product! It works great." Output: Positive sentiment Input: "The service was terrible. I wouldn't recommend it." Output: Negative sentiment

The natural language representation of APIs of the natural language processing feature extraction domain learned by our method: *Pattern 1:* Language-Specific Processing - Adapting feature extraction techniques to handle text in a specific language or multiple languages (e.g., 'Extract some features from a Korean text').
Pattern 2: Text Analysis Techniques This pattern includes the methods or techniques used for feature extraction and text analysis (e.g., 'create contextual embeddings for text').
Pattern 3: This pattern is about extracting features from text to determine sentiment or emotion, often used in analyzing customer reviews (e.g., 'provide the overall sentiment').
Pattern 4: Feature Extraction for Classification and Categorization This pattern involves the extraction of features from text to be used in classification models, such as support vector machines (SVMs) or categorization based on similarity (e.g., 'build a classifier to identify the most suitable applicants').

Table 4: Case study.

The natural language representation of the 'atm support' API of the banking77 classification task learned by our method: *Pattern 1:* "ATM Card Acceptance Inquiry - Users seek information on whether ATMs will accept their card for withdrawals. 'Can I withdraw money using this card at any ATM?',

Pattern 2: "ATM Service for Card Types - Questions about ATM services catering to particular card brands for cash withdrawals. 'Where can I find ATMs that service Mastercard for cash withdrawals?',

Pattern 3: "Questions regarding whether an ATM will accept or decline a specific card. - What ATMs accept my card?"]

The natural language representation of the 'Grab' API of the Virtualhome task: *Pattern 1:* 'Reading and Information Acquisition - Tasks that involve the agent needing to physically handle reading materials. (Clarified to specify the action of grabbing reading materials) - Example Obtain book for reading, Acquire magazine to read',

Pattern 2: 'Preparatory Actions - Tasks that involve the agent preparing for an activity or setting up an environment, potentially requiring the agent to take hold of essential objects. (Clarified to emphasize the action of grabbing objects for preparation) - Example Gather ingredients for cooking, Collect mail for sorting',

Pattern 3: 'Electronic Device Operation Tasks involving the operation of electronic devices. - Turn on light - Change TV channel - Turn on TV', 'Entertainment and Leisure Tasks related to leisure activities or entertainment. - Play on laptop - Playing video game',

Pattern 4:'Personal care or readiness - The agent is given tasks related to personal care or preparing to leave, which may involve grabbing personal items. - Get ready to leave - Go to toilet - Get ready for school',

Pattern 5: 'Coffee Preparation Tasks that involve making coffee. Make coffee']

Table 5: More examples.