NexusRaven: a Commercially-Permissive Language Model for Function Calling

Venkat Krishna Srinivasan Nexusflow
Zhen Dong Nexusflow
Banghua Zhu Nexusflow
Brian Yu Nexusflow
Damon Mosk-Aoyama Nexusflow
Kurt Keutzer Nexusflow
Jiantao Jiao Nexusflow
Jian Zhang Nexusflow

Abstract

The rise of open-source, commercially permissive large language models (LLMs) is revolutionizing generative AI, presenting organizations with enhanced control, minimized data risks, and cost benefits compared to proprietary models. However, in the field of tool use and function-calling LLMs, many open-source models, such as Gorilla and ToolLLAMA, are dependent on proprietary LLMs like GPT-4 for high-quality training data, which often faces legal restrictions for competitive commercial applications. In this paper, we introduce NexusRaven-13B, an open-source LLM designed for function calls. Originating from the CodeLLAMA-13B lineage, NexusRaven-13B employs a unique data curation via multi-step refinement, ensuring high-quality training data without relying on GPT-4 distillation. NexusRaven-13B matches GPT-3.5 in zero-shot function-calling accuracy. When combined with our second core technique, demonstration retrieval augmentation, its performance significantly surpasses GPT-4. The code, model, and demo will be available after the review process.

1 Introduction

Recent advances in large language models (LLMs) have enabled significant new capabilities, including natural dialogue, mathematical reasoning, program synthesis, and tool use (Brown et al., 2020; Bubeck et al., 2023; Chowdhery et al., 2022; OpenAI, 2023; Anthropic, 2023). The rapid evolution of generative AI has been marked by the development of open-source models (Touvron et al., 2023; Zheng et al., 2023; Patil et al., 2023; Qin et al., 2023b; Tang et al., 2023). These models offer a plethora of advantages, including enhanced control, reduced risks associated with sensitive data, and significant cost savings, especially when compared to proprietary models such as OpenAI's GPT-3.5/4. Such benefits have proven invaluable for the community looking to adopt in business applications.

To enable LLMs to accomplish complex tasks, there has been a growing interest in LLMs tailored for *function calling* and *tool use*, given their potential to play a pivotal role in real-world business applications, particularly in software operation tasks. In the function calling task, the LLM is given a list of candidate functions with definitions and docstrings, along with a human natural language instruction requesting to accomplish certain tasks. The LLM is expected to choose the right function from the list of candidate functions, along with generating the executable function call with the right arguments to accomplish the task.

Many existing open-source LLMs focusing on tool usage, such as Gorilla (Patil et al., 2023), ToolLLAMA (Qin et al., 2023b), and ToolAlpaca (Tang et al., 2023), rely heavily on proprietary LLMs like OpenAI's GPT-3.5/4 to generate large amounts of quality training data. However, legal con-

Workshop on Instruction Tuning and Instruction Following at NeurIPS 2023.

straints, like those in OpenAI's terms, prevent the use of such data for building models competitive with the proprietary LLMs in commercial use cases.

Furthermore, LLMs designed for function calling can be applied in real-world business scenarios to serve a crucial role in the common task of operating software. This demands a high degree of reliability and accuracy while keeping costs low. Unfortunately, flagship models with general code generation capabilities, such as CodeLLaMA-34B and GPT-4, all appear excessively large with regard to efficiency considerations. This motivates us to ask:

> *Can we build a commercially permissive open-source compact LLM designed for function calling, and use it to deliver solutions with quality competitive to or better than what proprietary LLMs offer, such as GPT-4 function calling API?*

Due to the reduced ability to reason and follow complex, multi-step instructions exhibited by the open LLMs as compared to closed LLMs, generating high-quality data from medium-sized open LLMs is a difficult task. Empirically, the key seems to be in reducing the reasoning burden of the task on these models; and we hypothesized that this can be accomplished by carefully engineering our pipelines to be a multi-stage refinement process, where each individual stage places as little reasoning demand on the LLM as possible. However, when combined end-to-end, the output should rival distillation data that closed models can provide via complex instructions.

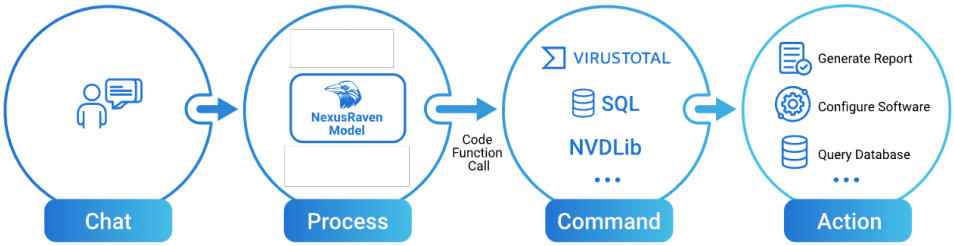

Figure 1: NexusRaven-13B translates user queries into executable function calling code, accomplishing tasks based on plain English input.

In response to this challenge, we present NexusRaven-13B, a new open-source Large Language Model (LLM) tailored for function calling in operating software tools. As depicted in Figure 1, NexusRaven-13B seamlessly processes human instructions in tandem with candidate API function documentation. In scenarios involving zero-shot or few-shot in-context learning, it is adept at generating executable function calling code, selecting the appropriate API, and determining the corresponding argument values.

Originating from the open-source lineage of the CodeLLAMA-13B model, NexusRaven-13B's training dataset was meticulously curated through a *multi-step refinement* process, leveraging CodeLLAMA-34B-instruct and LLaMA-70B-chat. This approach ensures its commercial permissiveness, setting a new benchmark in open-source function calling LLMs. We also propose a new *demonstration retrieval*-based system that further boosts the performance of the function calling accuracy by retrieving the most relevant demonstration examples from past seen query-response pairs.

With the two techniques above, NexusRaven-13B achieves the following:

- **Strong zero-shot performance:** NexusRaven-13B excels in zero-shot function calling for software not encountered during its training. Notably, NexusRaven-13B achieves 60% higher function call success rate compared to Gorilla (Patil et al., 2023), ToolLLAMA (Qin et al., 2023b), and ToolAlpaca (Tang et al., 2023) in the cybersecurity domain, and 4% higher than GPT-3.5 without training on any of the functions.
- **Enhancement with demonstration retrieval:** When integrated with our demonstration retrieval system, NexusRaven-13B shows a 30% higher function calling success rate than OpenAI GPT-4 in function calling for operating cybersecurity software, as seen in Figure 2. These software tools, such as CVE/CPE search and VirusTotal, were not included in the model training data and often feature sophisticated argument lists or extensive arrays of functions.

- **Cost effiective and low latency:** NexusRaven-13B achieves robust function calling capabilities without relying on high-latency search or iterative reasoning techniques, which typically require observing the outcomes of executing incorrect function calls. This guarantees truly interactive response times for applications built on NexusRaven-13B and eliminates the risks associated with incorrect function calls.

2 Related Work

LLMs with Tools The endeavor to enhance tool utilization in Large Language Models (LLMs) has burgeoned recently Mialon et al. (2023); Qin et al. (2023a). Pioneer works of LLMs with tools often augment models' tool-specific usage via fine-tuning Parisi et al. (2022); Schick et al. (2023), which poses challenges for practical plug-and-play usage. Recent advancements attach exemplary demos in the prompt, engaging LLMs with a spectrum of tools, from specialized ones such as code interpreters Gao et al. (2023); Chen et al. (2022), to retrieval augmented Li et al. (2022); Mueller et al. (2023) more versatile toolsets Patil et al. (2023); Xu et al. (2023); Zhuang et al. (2023); Qin et al. (2023a); Tang et al. (2023). Notably, ToolAlpaca Tang et al. (2023) covers various tool-use scenarios by building a multi-agent simulation environment to automatically generate tool-use corpus. ToolLLM Qin et al. (2023b) addresses the requisite of real-world API interactions by leveraging RapidAPI Hub, where ChatGPT is used to generate diverse human instructions involving these APIs, covering both single-tool and multi-tool scenarios. In addition to using LLMs with demonstrations, Hsieh et al. (2023) further simplifies prompt design by only leveraging documentation for tools without offering demos, which maintains competitive performance.

Furthermore, model reasoning also plays an important role in LLMs with tools, where existing works with simple prompting cannot fully elicit the capabilities in LLMs and therefore fail to handle complex instructions. CoT Wei et al. (2022), ART Paranjape et al. (2023), ReAct Yao et al. (2022), and Reflexion Shinn et al. (2023) leverage LLMs to automatically generate intermediate reasoning steps as well as actions, thereby improving interpretability and problem-solving abilities of LLMs in diverse decision-making tasks. In ToolLLM Qin et al. (2023b), the authors use ChatGPT to search for a valid solution path (chain of function calls) for each instruction. Although useful, a long solution path would significantly increase the number of function calls and therefore consume more time and resources.

Self-Refinement The idea of using language model to improve itself has appeared in the literature of self-refinement (Madaan et al., 2023) and reinforcement learning with AI feedback (RLAIF) (Lee et al., 2023; Bai et al., 2022). The work of self-refinement focuses on refining the responses during inference time to improve the quality of the responses, while RLAIF focuses on using refined responses to collect preference data. In contrast, our multi-round refinement relies on multiple models and multiple rounds to create different levels of refinement of the instruction-tuning dataset, including creating the prompt with given function calls, creating chain-of-thought data, and refining the responses.

Demonstration Retrieval Demonstration retrieval has been applied for semantic parsing (Pasupat et al., 2021), question answering (Mueller et al., 2023; Khattab et al., 2022), coding (Zhou et al., 2022) and identifying the right tool to use (Patil et al., 2023). In our approach, we show that such demonstration retrieval is helpful for improving the accuracy of function calling.

3 Methodology

3.1 Data Curation with Multi-step Refinement

In the context of creating powerful compact models, it is a standard approach to distill the generation of larger and more powerful models into smaller ones. The power of such distillation is especially well pronounced on models such as Gorilla, ToolLLAMA, and ToolAlpaca, which distill GPT-3.5/4 generations to refine base open-source models, resulting in models not commercially permissive due to OpenAI's terms of use.

While generation distillation in the literature may appear straightforward with GPT-4, we have found that these methods do not result in high-quality data when we attempt to distill generations for

function calling from commercially permissive models. Particularly, these commercially permissive models tend to not be able to follow instructions as well as GPT-4, so distilling diverse and high-quality examples that differ materially from the seed examples is difficult.

In addition to this, we observe that CodeLLaMA-34B-instruct only achieves a 48.3% success rate in generating API function calls for VirusTotal, while GPT-4 attains 80.8%. These observations are likely due to the gap in reasoning capability between CodeLLaMA and GPT-4.

To navigate through these challenges, we pivoted to a new data generation methodology. The principle of this new approach is to decompose the data generation into multiple steps, with each requiring simpler and more primitive reasoning, as shown in Figure 4. This reduces the reasoning and instruction-following burden of open-sourced models, allowing for viable data distillation pipelines to be built on them.

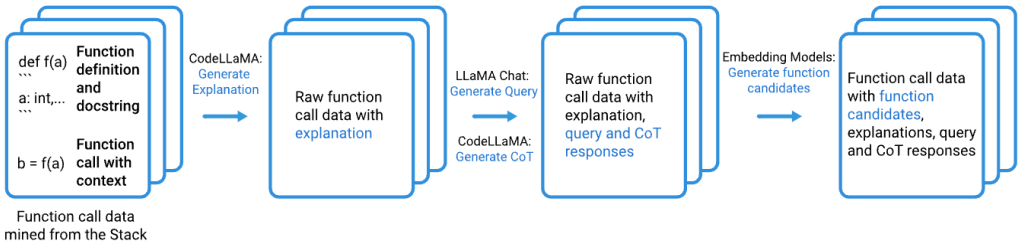

Figure 2: Illustration of our multi-step refinement pipeline.

One of the primary hurdles is relying on open-sourced models to generate *both the diverse ground truths and the queries*. We noticed that these models tended to generate new examples that are very similar to the seed examples. Rather, we pivoted, and instead of relying on the LLMs to infuse diversity, we relied on the examples themselves to be diverse.

We achieved this by starting our data curation by mining tuples of function definitions, docstrings, code context, and the actual function calls from open-sourced corpora, such as The Stack. We then prompt several LLMs to add information about these pairs, allowing them to be used as training examples. This information includes asking the model to generate natural language queries that plausibly might be addressed by executing this function, as well as the chain of thought that could plausibly lead to that function call. We expect this curation pipeline to deliver a large volume of diverse and meaningful generated pairs of plain English queries and function calling code, as well as the Chain-of-Thought (CoT) reasonings for mapping between the query/instruction and the code.

- **Mining**. We first mine raw function calls from The Stack. These calls include the function execution itself, but also a context window of about 100 lines around the call. It also includes the associated function prototype, the function body, and the accompanying docstrings. Functions that do not contain docstrings or are not statically resolvable are discarded.
- **Function Explanation**. We then feed the raw mined tuples into CodeLLaMA-34B-instruct and generate the capability description of functions. These function explanations are intended to help LLMs better understand the function of data generation.
- **Query Generation**. With the mined function calling code and the function explanation, we elicit LLaMA-70B-chat to generate a natural language query description for the code, which we find results in a higher quality natural language query than using CodeLLaMA-34B-instruct.
- **Chain-of-Thought Enhancement**. To explicitly improve the reasoning capability for function calling, we further leverage CodeLLaMA-34B-instruct to generate CoT traces elaborating on how the values for arguments are derived. We additionally use these CoT traces and the query to regenerate the function call code to further improve the compatibility between queries and the code.
- **Hard-Negative Candidate Function List Generation**. Given that selecting the right function from a list of candidates is also required for function calling capability, we use embedding models to augment each curated training data sample with a list of functions similar to

the intended one, increasing the difficulty of the task during training for better generalizable models.

This four-step pipeline empirically yields high-quality data samples consisting of queries/instructions, candidate function documentation, CoT reasoning, and the executable code for function calling. We further instruction-tune the CodeLLaMA-13B model with this generated data to supercharge the function calling capabilities.

3.2 Demonstration Retrieval Augmentation

The data curation procedure above, combined with instruction tuning, bumps the CodeLLaMA-13B function call accuracy on the VirusTotal dataset from 38% to 72%. Although a commendable increase, this accuracy is not sufficient for the robustness required in real-world scenarios. Our approach to achieving the highest level of accuracy is demonstration retrieval augmentation.

When integrated with LLMs, conventional retrieval systems primarily function as a component of caching systems. They assist in answering questions from extensive knowledge repositories and facilitate the extraction of the most relevant functions from vast API document collections. Our approach, which is different from these conventional approaches, is to use retrievers to directly source demonstration examples from a corpus of existing query-response pairs. The corpus has 16 examples on average per API function, and we use four-shot prompting to generate function calls. We find that this significantly boosts the function calling success rate from 72% to 94%.

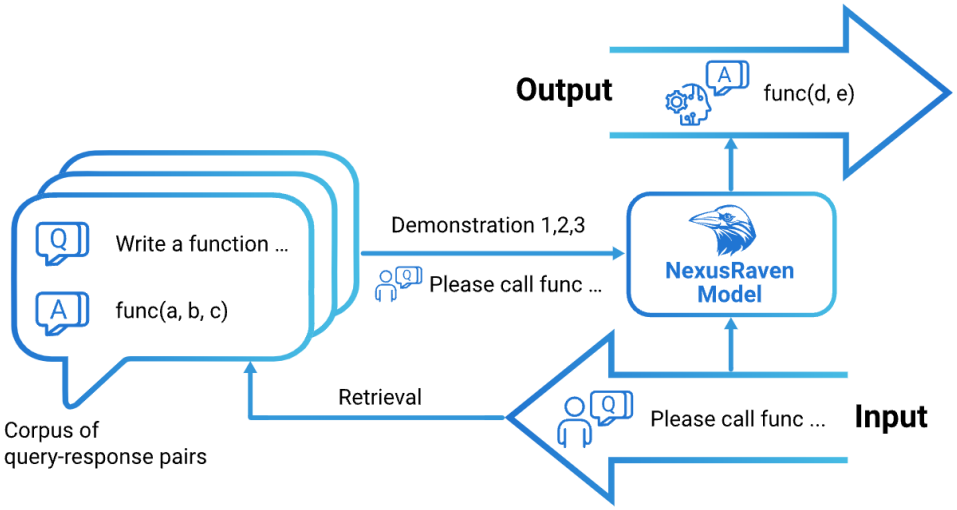

Figure 3: Illustration of our demonstration retrieval augmentation system. When presented with a new query, our system scans an existing corpus to identify demonstration examples that can enhance the quality of responses to that query.

We acknowledge that fine-tuning models on the corpus could boost model performance. Yet, our retrieval system offers two distinct advantages:

Generalization. While fine-tuning models directly can be expensive and risk weakening some capabilities, updating the retrieval corpus offers the best of both worlds: it keeps the model generalization capability on diverse and unseen tools, while still being able to significantly boost the accuracy for specific software.

Personalization. As we add more prompt-response pairs to the retrieval corpus through live interactions and feedback with the users, the system becomes better equipped to find closely matched responses to incoming queries, thereby significantly enhancing performance. It also adapts to a personalized use style when maintaining a corpus for past use cases.

4 Evaluating NexusRaven-13B for Function Calling Capability

We provide a comprehensive benchmark comparing NexusRaven-13B and existing function-calling models. We show that NexusRaven-13B is comparable to GPT-3.5 in the zero-shot setting, and surpasses the accuracy of GPT-4 when equipped with demonstration retrieval, in both cybersecurity and generic domains.

4.1 Evaluation Dataset and Pipeline

We evaluate the function calling capability of different models by sending natural language instructions to all models, along with the function definition and docstring for all available function calls. In each test sample, we provide multiple candidate functions. The candidate functions contain the ground truth function that shall be used for the instructions, along with noisy functions that are similar but not for the instructions. We evaluate the output of the model by comparing it with the ground truth by executing the function call and ensuring the arguments match the ground truth exactly.

We compare NexusRaven-13B with GPT-4, GPT-3.5-turbo-instruct, GPT-3.5-turbo, CodeLLaMA-13B-instruct, Gorilla, ToolLLaMA and ToolAlpaca. We do not directly compare with GPT-4 function calling API since our function description is longer than the maximum length allowed for function definitions. For the cybersecurity domain, our full benchmark consists of human queries to operate CVE and CPE Search, VirusTotal V3, and EmailRep. We collect functions from their API documentation and curate the ground truth answers by working together with cybersecurity domain experts. We make sure the ground truth answer is consistent with expert evaluation in real-world use cases.

For the generic domain, we consider two popular benchmarks in the literature, the ToolAlpaca-Simulated dataset, and the ToolLLM dataset, for our evaluation. The ToolLLM and ToolAlpaca datasets were generated using GPT-3.5. This inherently resulted in data samples with noisy ground truth annotations. These noisy ground truth annotations are challenging to be fully captured for perfectly reliable evaluation. Nonetheless, we conducted extensive filtering to remove data samples that certainly contained incorrect function calling codes. We will open-source the evaluation pipeline after the review process.

4.2 Performance of the Zero-shot Model

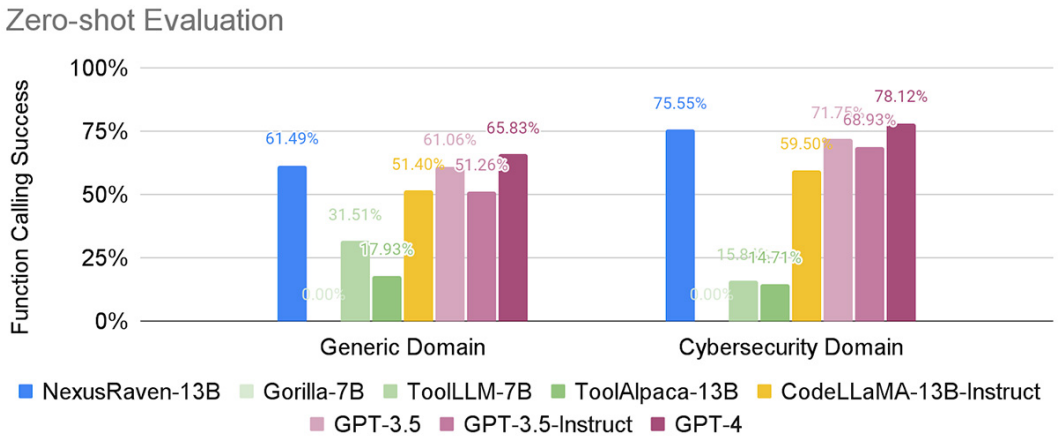

Figure 4: Zero-shot comparison between NexusRaven-13B and representative function calling models on both generic domain and cybersecurity domain.

As is shown in Figure 4, when prompting the models in a zero-shot setting, NexusRaven-13B demonstrates competitive performance in the cybersecurity domain and achieves a 4% higher success rate in the generic domain compared to GPT-3.5. It also beats all representative open source function-calling models in both the generic domain and the cybersecurity domain, showing 16% improvement over CodeLLaMA-13B-instruct, and 60% over other open source models in the cybersecurity domain.

In our zero-shot evaluation, we assumed the LLMs cannot search by observing the outcome of potentially incorrect function calls. This is because incorrect or unintentional function calls could produce unexpected detrimental effects on software systems. Nonetheless, we turned on the search feature of ToolLLM and ToolAlpaca without executing the potentially incorrect function calls. In this setting, we consider function calling to be successful if any trials in the search are correct. We enable this design to be extra fair to ToolLLM and ToolAlpaca. We also note here that Gorilla attains a low success rate because of the lack of generalization on software unseen during training. In contrast, NexusRaven-13B excels in the zero-shot function calling for generic domain software APIs not encountered during its training.

4.3 Performance of the Retrieval-augmented Model

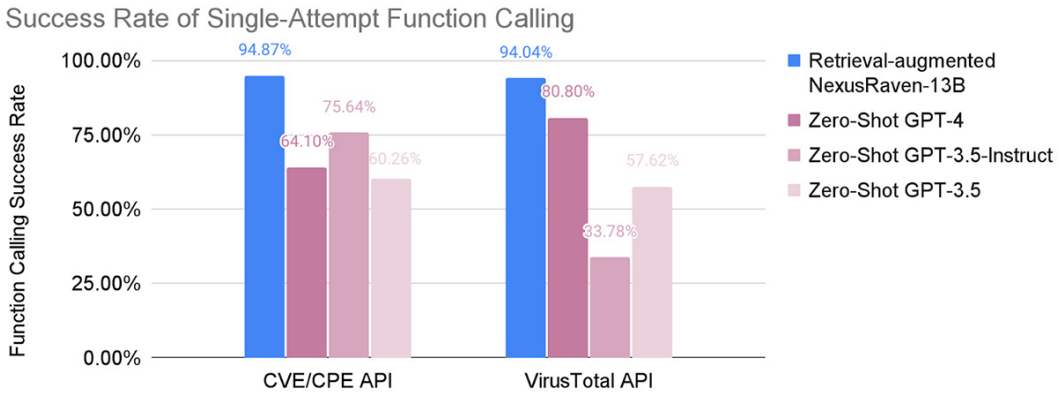

Figure 5: Comparison between the prototype system powered by NexusRaven-13B and the GPT-4 function calling API for CVE and CPE Search, VirusTotal V3, two challenging cybersecurity software in our benchmark.

We benchmarked the performance of the retrieval-augmented model. As is shown in Figure 2, our prototype system demonstrates a 30% higher function calling success rate on average than the GPT-4 on CVE/CPE, providing enterprise-grade function calling capabilities at the last mile of quality.

To ensure a fair comparison, we have done explicit decontamination to make sure there is no overlap between the retrieval corpus and the evaluation data.

4.4 Open Sourcing the Evaluation Framework

As LLMs for tools is a new area, we find that the evaluation datasets, methods, and code bases are fragmented and not necessarily compatible with each other. In addition, formatting for the descriptions of the tools is also massively varied, from OpenAPI descriptions to simple JSON descriptions. To facilitate a unified evaluation pipeline, we first standardize the description of tools as Python functions, despite them either being APIs or simple functions, as we found this description to be the most intuitive, most compatible with existing code and models, and easy to standardize into. In addition, we will open-source the evaluation framework integrating the current evaluation benchmarks. Users can directly plug in their function definitions, doc-strings, instruction lists, and ground truth data for one-click evaluation without any extra effort.

5 Conclusion and Future Steps

In this paper, we introduce two core techniques, multi-step refinement, and demonstration retrieval augmentation, enabling high-quality function-calling models, This version of NexusRaven-13B primarily emphasizes single-round interactions with humans through natural language instructions. We are eager to collaborate with the community to enhance NexusRaven's capability for multi-round interactions. We plan to further refine our evaluation benchmark as a comprehensive function calling benchmark. We are excited to standardize the evaluation of tool-using LLMs.

Acknowledgements

We extend our gratitude to the LLAMA 2 and CodeLLAMA team for empowering the open model community with their powerful pretrained models. The creation of NexusRaven-13B would not have been possible without the foundation laid by these open models. We also express our appreciation to the open scientific community for their pioneering efforts in code generation, exemplified by projects like The Stack, Starcoder, and CodeGen. These open-source models and data have significantly accelerated our progress in the field of code generation. Over the past year, the research community has achieved remarkable strides in tool-augmented LLMs. Works such as Gorilla, ToolLLM, ToolLlama, and Toolbench have provided the bedrock upon which we built NexusRaven.

References

Anthropic. Model card and evaluations for claude models, 2023. URL `https://www-files.anthropic.com/production/images/Model-Card-Claude-2.pdf`. Accessed: Sep. 27,2023.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

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

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International Conference on Machine Learning*, pp. 10764–10799. PMLR, 2023.

Cheng-Yu Hsieh, Si-An Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. Tool documentation enables zero-shot tool-usage with large language models. *arXiv preprint arXiv:2308.00675*, 2023.

Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive nlp. *arXiv preprint arXiv:2212.14024*, 2022.

Harrison Lee, Samrat Phatale, Hassan Mansoor, Kellie Lu, Thomas Mesnard, Colton Bishop, Victor Carbune, and Abhinav Rastogi. Rlaif: Scaling reinforcement learning from human feedback with ai feedback. *arXiv preprint arXiv:2309.00267*, 2023.

Huayang Li, Yixuan Su, Deng Cai, Yan Wang, and Lemao Liu. A survey on retrieval-augmented text generation. *arXiv preprint arXiv:2202.01110*, 2022.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023.

Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842*, 2023.

Aaron Mueller, Kanika Narang, Lambert Mathias, Qifan Wang, and Hamed Firooz. Meta-training with demonstration retrieval for efficient few-shot learning. *arXiv preprint arXiv:2307.00119*, 2023.

OpenAI. Gpt-4 technical report, 2023.

Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. Art: Automatic multi-step reasoning and tool-use for large language models. *arXiv preprint arXiv:2303.09014*, 2023.

Aaron Parisi, Yao Zhao, and Noah Fiedel. Talm: Tool augmented language models. *arXiv preprint arXiv:2205.12255*, 2022.

Panupong Pasupat, Yuan Zhang, and Kelvin Guu. Controllable semantic parsing via retrieval augmentation. *arXiv preprint arXiv:2110.08458*, 2021.

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.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, et al. Tool learning with foundation models. *arXiv preprint arXiv:2304.08354*, 2023a.

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*, 2023b.

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

Noah Shinn, Beck Labash, and Ashwin Gopinath. Reflexion: an autonomous agent with dynamic memory and self-reflection. *arXiv preprint arXiv:2303.11366*, 2023.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, and Le Sun. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*, 2023.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. On the tool manipulation capability of open-source large language models. *arXiv preprint arXiv:2305.16504*, 2023.

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.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*, 2023.

Shuyan Zhou, Uri Alon, Frank F Xu, Zhengbao JIang, and Graham Neubig. Doccoder: Generating code by retrieving and reading docs. *arXiv preprint arXiv:2207.05987*, 2022.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. Toolqa: A dataset for llm question answering with external tools. *arXiv preprint arXiv:2306.13304*, 2023.