JORA: JAX Tensor-Parallel LoRA Library for Retrieval Augmented Fine-Tuning

Anique Tahir
Arizona State University
699 S. Mill Avenue
Tempe, AZ
research@anique.org

Lu Cheng
University of Illinois Chicago
851 S. Morgan St.
Chicago, IL
lucheng@uic.edu

Huan Liu
Arizona State University
699 S. Mill Avenue
Tempe, AZ
huanliu@asu.edu

Open-source repository:
https://github.com/aniquetahir/JORA
Supplementary video:
https://youtu.be/-auF_9wF2S0?si=o50HBySZjTpjtWR_

Abstract

The scaling of Large Language Models (LLMs) for retrieval-based tasks, particularly in Retrieval Augmented Generation (RAG), faces significant memory constraints, especially when fine-tuning extensive prompt sequences. Current open-source libraries support full-model inference and fine-tuning across multiple GPUs but fall short of accommodating the efficient parameter distribution required for retrieved context. Addressing this gap, we introduce a novel framework for PEFT-compatible fine-tuning of GPT models, leveraging distributed training. Our framework uniquely utilizes JAX's just-in-time (JIT) compilation and tensor-sharding for efficient resource management, thereby enabling accelerated fine-tuning with reduced memory requirements. This advancement significantly improves the scalability and feasibility of fine-tuning LLMs for complex RAG applications, even on systems with limited GPU resources. Our experiments show more than 12x improvement in runtime compared to Hugging Face/DeepSpeed implementation with four GPUs while consuming less than half the VRAM per GPU.

1 Introduction

Large Language Models (LLMs) like ChatGPT (Achiam et al., 2023) have revolutionized the field of natural language processing, paving the way for open-source alternatives that offer more flexibility in fine-tuning. Llama-2 (Touvron et al., 2023), a prominent LLM, exemplifies this trend, offering extensive customization at the architecture level. Alongside, Parameter Efficient Fine-Tuning (PEFT) (Fu et al., 2023) techniques like Low-Rank Adaptation have emerged, optimizing resource utilization in training these models. Retrieval Augmented Generation (RAG) (Lewis et al., 2020a) is a paradigm that leverages a corpus to enrich LLM prompts with relevant context. However, when fine-tuning on retrieval-based context, the quadratic memory scaling of transformer models with prompt length poses significant challenges, especially when integrating large context sizes. The training process, which employs teacher-forcing at each step of the sequence, exacerbates memory demands, creating a bottleneck for effective LLM utilization in RAG.

Current machine learning frameworks facilitate LLM fine-tuning on distributed systems, employing model and pipeline parallelism strategies. However, these frameworks lack support for PEFT, specifically in the context of parallel training. While libraries such as DeepSpeed (Rasley et al., 2020) and Accelerate (Gugger et al., 2022) offer data parallelism for fine-tuning the entire model, these libraries lack support for tensor-parallel training in the PEFT setting. In addition, combining multiple libraries adds unnecessary boilerplate code to glue together dependencies required for parameter-efficient and distributed training. These libraries also require boilerplate code for configuration since they target multiple models.

To bridge this gap, we introduce JORA (JAX-based LORA), a library tailored for Llama-2 models, designed to enhance the fine-tuning process for RAG applications. Utilizing JAX's just-in-time (JIT) compilation and innovative tensor-sharding techniques, JORA not only accelerates the fine-tuning process but also significantly optimizes memory usage (Bradbury et al., 2018). Our evaluations across standard training GPUs demonstrate substantial improvements in training time and memory efficiency, addressing the critical challenges of PEFT in retrieval-based training. Our library also provides valuable helpers for using instruct format datasets, merging LORA parameters, and convert-

Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations), pages 152–159
August 11-16, 2024 ©2024 Association for Computational Linguistics

ing fine-tuned models to Hugging Face compatible formats. Our work makes PEFT more accessible and efficient for LLMs, particularly in resource-constrained environments. By enhancing the scalability and efficiency of LLMs in retrieval augmented fine-tuning (RAFT), JORA opens new avenues for advanced natural language processing applications.

2 Background

JORA introduces the concept of RAFT. This workflow employs retrieved knowledge and outcomes to create context and expected outputs. The fine-tuning process encourages the model to learn a rationale to derive the output from the knowledge. Prior related work focuses on RAG, the inference counterpart of RAFT, whose bottleneck is the sequence length used for context in the prompt. Since RAFT shares the same bottleneck, our framework focuses on adding efficiency by providing a memory-efficient and distributed backend while exposing an intuitive API. We highlight the importance of RAG and the capabilities of other libraries which aim to solve related problems. We highlight how our library fills the gap.

2.1 Retrieval Augmented Generation

RAG has gained significant attention in recent years, with various approaches exploring it to enhance LLM generation. The integration of dense and sparse retrievers with LLMs, as discussed in (Robertson et al., 2009; Seo et al., 2019), highlights the diversity in retrieval techniques used for augmenting LMs. Chen et al. (2017), Clark and Gardner (2017), and others have contributed to conditioning LMs on retrieved documents, demonstrating significant improvements in knowledge-intensive tasks (Lee et al., 2019; Guu et al., 2020; Khandelwal et al., 2019; Lewis et al., 2020b; Izacard and Grave, 2020; Borgeaud et al., 2022; Murez et al., 2020). The concept of chain-of-thought prompting in combination with retrieval mechanisms, as proposed by Wei et al. (2022), marks a novel approach in this domain. The evolution of LMs into agent-like models, capable of generating queries and performing actions based on prompts, is evident in the works of Thoppilan et al. (2022), who introduced models like LaMDA. Menick et al. (2022), Komeili et al. (2021), and Nakano et al. (2021) further explored the generation of internet search queries by LMs.

2.2 Parallel Training Libraries

Several open-source libraries expose an interface for multi-GPU training for LLMs. Hugging Face implementation of Transformer models allows multi-GPU inference. The Transformers library also includes a trainer. Hugging Face's Accelerate (Gugger et al., 2022) library is a tool designed to simplify the process of running PyTorch training scripts on different devices, including CPU, single GPU, multiple GPUs, and TPUs while supporting mixed precision and distributed settings. It offers an easy-to-use API that allows users to run their PyTorch code across any distributed configuration with minimal changes, making training and inference at scale more straightforward. DeepSpeed (Rasley et al., 2020) is an open-source optimization library for PyTorch developed by Microsoft. It is designed to accelerate the training and inference of deep learning models, mainly focusing on large-scale models. The library addresses challenges such as memory constraints and slow training times, aiming to enhance deep learning workflows' performance and efficiency. Accelerate utilizes DeepSpeed or FSDP for distributed training.

JORA solves several issues with prior libraries: i) we target specific models to reduce the boilerplate required for the training process, ii) we utilize JAX's jit optimizations for training to improve training performance compared to PyTorch. iii) we provide a tensor-parallel, multi-GPU implementation of training, and iv) we provide utility functions to simplify the data loading experience, fine-tuning the model, and compatibility with the Hugging Face ecosystem.

3 JORA Framework

JORA is a library for RAFT. Its purpose is to make fine-tuning based on retrieved context more user-friendly. In addition, it is designed to make RAFT faster and more resource-efficient. Figure 1 gives a high-level overview of JORA.

3.0.1 JAX

One of the highlights of our library is that it allows LoRA training of LLMs using the JAX framework. JAX provides composable transformations of numerical functions e.g. automatic differentiation (grad), vectorization (vmap), parallelization (pmap), and just-in-time compilation (jit) (Bradbury et al., 2018). A function must

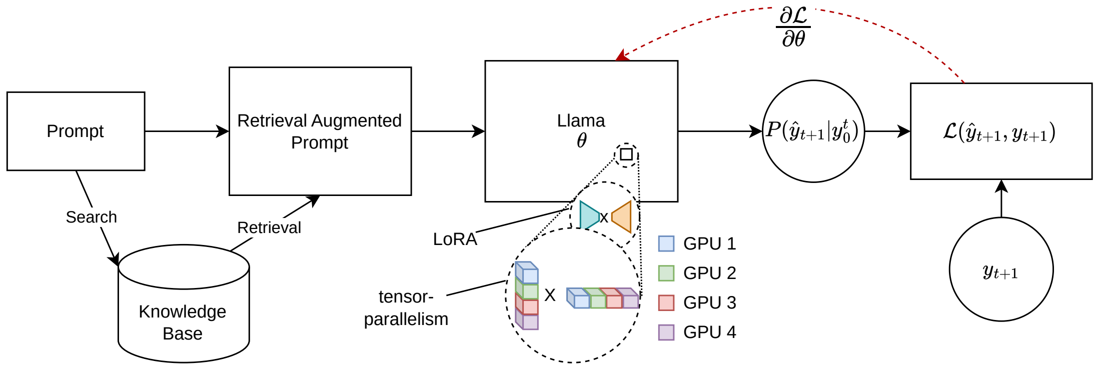


Figure 1: JORA is a library that aids in Retrieval Augmented Fine-Tuning by eliminating unnecessary boilerplate and introducing memory efficient training through tensor-parallelism and LoRA.

be pure and statically composed to benefit from these transformations. Functions compiled by JAX use the Accelerated Linear Algebra (XLA) library. Jit compilation allows program optimizations to the XLA to improve execution speed which is ideal for compute-heavy architectures such as transformers.

3.0.2 Dataset Loading and Training

```
[
  {
    "instruction": "Calculate the area
    of the following shape in square
    centimeters.",
    "input": "rectangle of size 4 cm x 5
     cm",
    "output": "20cm^2"
  },
  ...
]
```

Listing 1: An example of Alpaca format data.

Even though JORA is compatible with general-purpose fine-tuning pipelines, we provide helper functions for loading training data in alpaca format (Taori et al., 2023). The Alpaca dataset format is ideal for RAFT since it follows the instruction-tuning format. Each sample in this format may contain an instruction, input (optional), and output. Listing 1 shows an example of this data format. Retrieved knowledge can be used as the input and separated from the instruction and output. The output represents the sequence that the model generates.

```
class AlpacaDataset(Dataset):
    def __init__(self, *, path: str,
    split=Union[Literal['train'],
    Literal['test']],
                 split_percentage=0.8,
    tokenizer=None, max_len=512,
    alpaca_mix=0.3) -> None:
```

Listing 2: Function signature for the constructor for AlpacaDataset.

We provide the class 'AlpacaDataset' for user-friendly data loading, which inherits from PyTorch's 'Dataset' class. Listing 2 shows the signature for the constructor for this class. In addition to loading the dataset, the $alpaca\_mix$ parameter allows merging a percentage of the original alpaca dataset to prevent overfitting on the fine-tuned data. The class also provides the ability to create training and testing splits based on the provided split percentage. The AlpacaDataset collators apply instruction-masking by default.

3.0.3 Training API

How fine-tuning proceeds depends on a variety of parameters. Since this library aims to simplify the training process, JORA provides common defaults for starters. In addition, it allows customization of the training process for more advanced usage. Listing 3 shows the configuration class. JAX_PARAMS_PATH specifies the location of the model parameters. $LLAMA2\_HF\_PATH$ specifies the location of Meta's model in Hugging Face format. Our library uses the Hugging Face model path to access it's tokenizer. Since the release of JAX native LLMs, such as Gemma (Team et al., 2024), our library supports loading models without a Hugging Face format. For the sake of brevity, our examples follow the datastructures for Llama-2. Other model configurations follow suit with model specific naming schemes.

```
class ParallamaConfig(NamedTuple):
    JAX_PARAMS_PATH: str
    LLAMA2_HF_PATH: str
    LORA_R: int = 16
    LORA_ALPHA: int = 16
    LORA_DROPOUT: float = 0.05
```

Listing 3: JORA allows the common defaults for the configuration with room for specificity.

3.0.4 Model Transfer API

Most open-source libraries that utilize LLM's are compatible with Hugging Face's model format. Since JORA uses JAX for its training procedure, the caveat is incompatibility with the popular libraries. To overcome this limitation, we provide a simple script to convert models trained using our library to the Hugging Face format. Listing 4 provides a description of the conversion script usage.

JORA builds on LLM implementations in JAX which uses jit and vmap. GPT-based models use the decoder component of the transformer architecture to produce text autoregressively. Since transformer models consist of multi-headed self-attention, the memory used at the inference stage scales quadratically with the input sequence length. This is a significant drawback for RAFT since augmenting a prompt with retrieved-context adds to the sequence length. As such, one of the aims of our library is to assuage the memory utilization requirements by efficiently distributing memory usage across GPU resources.

```
SYNOPSIS
    huggingface_merger.py
    HUGGINGFACE_PATH JAX_PATH SAVE_PATH

POSITIONAL ARGUMENTS
    HUGGINGFACE_PATH
        Type: str
        path to the HuggingFace llama
    model
    JAX_PATH
        Type: str
        path to LoRA parameters fine-
    tuned by JORA
    SAVE_PATH
        Type: str
        path to save the updated
    HuggingFace llama model
```

Listing 4: Hugging Face conversion script can be invoked from the command-line. The converted model can be used with other Hugging Face compatible libraries such as LangChain.

For our implementation of LoRA, we follow the suggestions presented by Hu et al. (2021), i.e., the query and value attention weights are enhanced. Specifically, the approach suggests that the computation, $W_0x + b_0$, can be tuned through $W_0x + b_0 + BAx$ where W_0 are subset of the models weights, B, A are the trainable countports of W_0 added by LoRA, $W_0, BA \in \mathbb{R}^{m \times n}$, $A \in \mathbb{R}^{r \times n}$, $B \in \mathbb{R}^{m \times r}$, and $r << m, n$.

Here, B and A are the trainable weights. W_0 and b_0 represent the weights and biases of a specific neural network component. Composing the trainable parameters to lower rank values significantly reduces the total parameters involved in backpropagation. Generative models are trained to predict the next token, given past tokens auto-regressively. Thus, the objective, $\mathcal{L}$, of the LLM is to reduce the discrepancy between the next predicted token $\hat{y}_{t+1}$ and the next ground truth token y_{t+1}, given the past tokens in the ground truth sequence, y_0^t. Consequently, the trained language model predicts the next token, given the past predicted tokens, $\hat{y}_0^t$.

For our implementation of LoRA, we add the LoRA parameters to the original weights as highlighted in Equation 1. The values of B and A are initialized from zeros and normal sampling, respectively.

$$\begin{aligned} Output &= W_0x + b_0 + BAx \\ &= (W_0 + BA)x + b_0 \end{aligned} \tag{1}$$

JORA parallelizes all parameters of the Llama model using JAX's positional sharding module. Transformers inherently support distributed computations through the use of parallel decoder blocks. GPT's consists of several layers of parallel decoder blocks. We utilize the inherent design and shard on the decoder axis. Projection and Embedding layers are sharded on the non-sequential dimension to avoid variation due to the input.

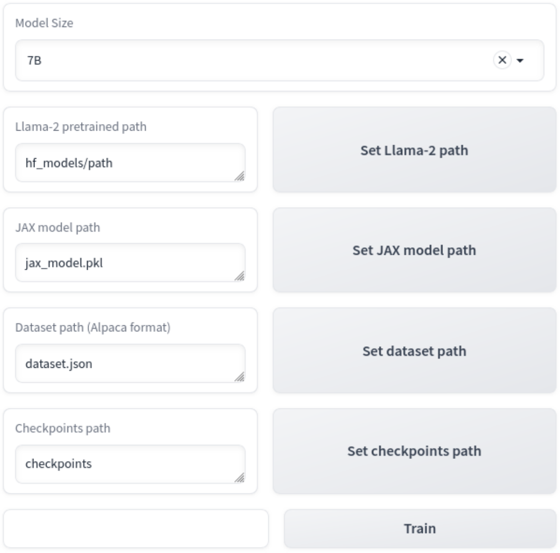
3.0.5 Library Usage

One of the core aims of JORA is to make fine-tuning easily accessible to the end-user. Compared to Hugging Face, JORA significantly reduces the lines of code to get started. In addition, JORA provides a GUI for fine-tuning LLMs. The following code can be used to fine-tune a model with minimal changes to default training parameters:

```
from jora import train_lora,
    ParallamaConfig,
    generate_alpaca_dataset

config = ParallamaConfig(
        MODEL_SIZE=model_size,
        JAX_PARAMS_PATH=jax_path,
        LLAMA2_META_PATH=hf_path)
dataset = generate_alpaca_dataset(
    dataset_path, 'train', config)
train_lora(config, dataset,
    checkpoint_path)
```

Alternatively, the GUI can set the fine-tuning parameters and training. Fig. 2 shows the interface for the GUI. It can be invoked with the following command:



Model Size: 7B

Llama-2 pretrained path: hf_models/path

JAX model path: jax_model.pkl

Dataset path (Alpaca format): dataset.json

Checkpoints path: checkpoints

Figure 2: JORA provides a simple GUI for fine-tuning.

```
python -m jora.gui
```

4 Experiments

We measure the improvement introduced by JORA in terms of memory utilization and computation speed, conducting experiments using Hugging Face/DeepSpeed for comparison. Our setup consists of a node of the SOL supercomputer (Jennewein et al., 2023) with 4 x A100 with 40GB of VRAM each, an AMD EPYC 75F3 32-core Processor, and 512GB of RAM. The GPUs are cross-connected using NVLink. All experiments use 16-bit brain floating point for parameter precision for a fair comparison.

4.1 Memory Utilization Analysis

We compare the memory utilization of our implementation with that of the Hugging Face trainer using Accelerate and PEFT. Our implementation is adapted from the examples in the official Hugging Face PEFT library, which uses Accelerate and DeepSpeed for parallel computation. Through parallelization, several parameters are replicated across multiple GPUs. As such, the total memory utilized by parallel training is greater than that used in a single GPU setting. However, the advantage of multi-GPU training is that the memory used by each GPU individually is less than that used in single-GPU training. JAX pre-allocates memory to avoid fragmentation, which makes measuring active allocation a challenge. For memory utilization analysis, we override this behavior by setting the XLA_PYTHON_CLIENT_ALLOCATOR environment variable to ‘platform.’ This environment variable informs JAX to allocate and deallocate memory as needed but impacts performance. Thus, for the performance evaluation, we use the default configuration.

For parallel training, DeepSpeed distributes parameters using data parallelism. Thus, though a single sample cannot be distributed, multiple samples can be aggregated, improving performance. Thus, JORA is beneficial since it allows a single lengthy sequence to backpropagate across multiple GPUs. Table 1 shows that JORA uses less memory per resource as the number of resources increases. The only case where Hugging Face/DeepSpeed consumes lower memory is where only one GPU is available.

4.2 Computation Time Comparison

We also measure computation time using the same RAFT dataset for the Hugging Face and JORA implementations over iterations of 1, 2, and 4 GPUs. Table 1 presents these results. JORA shows consistently better performance than Hugging Face implementation, with JORA implementation being over 12 times faster than the baseline with 4 GPUs. Since DeepSpeed used data parallelism, we observe a performance impact in multi-GPU settings, with the bottleneck being the slowest GPU/sample for backpropagation. In addition to improved performance, since JORA uses JAX’s jit functionality to run compiled computations, the performance of the implementation shows more consistency. We observe a computation performance drop between single and multiple GPUs. This drop could be attributed to cross-GPU communication overhead.

5 An Example Usage Scenario

JORA is designed to aid in RAFT. In this section, we demonstrate a RAFT use case by fine-tuning it on a social media dataset (Papasavva et al., 2020) to help LLMs enable social-context understanding. The purpose of RAG is to add additional context to a prompt by searching for knowledge and adding additional information. For RAFT, data can be created based on retrieved knowledge. The LLM learns to generate the retrieved answer based on the context since the key rationale is held back. A simple example is a database query, which corresponds to a process that may be taken to produce

	GPUs	1	2	4
Hugging Face PEFT w/ Microsoft DeepSpeed ZeRO-3	**Mem (MB)**	**20645.2** **(39.81)**	23056 / 23024 (14.63 / 29.29)	23978 / 23921 / 23463 / 23397 (47.87 / 50.39 / 31.96 / 17.46)
	Performance (secs)	4.56 (0.04)	2.81 (0.02)	5.45 (0.09)
JORA (Ours)	**Mem (MB)**	23102 (0.00)	**16068 / 16008** **(0.00 / 0.00)**	**11460 / 11448 / 11448 / 11400** **(0.0 / 0.00 / 0.00 / 0.00)**
	Performance (secs)	**0.19** **(0.00)**	**0.79** **(0.00)**	**0.44** **(0.00)**

Table 1: JORA shows significant improvement w.r.t. Hugging Face implementation of PEFT paired with DeepSpeed for parallelization. JORA uses tensor-parallelism to distribute memory allocation for parameters across GPU resources. The number in the brackets denotes the standard deviation across five runs.

an output by evaluating the database. If the query is not provided but rather a natural language equivalent is provided, the LLM must learn the heuristics represented by the hidden query.

Since prompt tuning is insufficient for models to develop social-context understanding (Gandhi et al., 2023), we use a fine-tuning process consisting of two phases to add knowledge to an LLM. Both phases of fine-tuning use PEFT. For our problem setting, rather than just predicting the following words, we aim to gain an understanding of the relation across different comments in a social media session. For instance, a comment in a social media session may target the previous comment, the original post that spawned the session, or some comment in the middle of the discourse. To glean insight into the target of the comment in terms of its context, reasoning between the structure of the conversation is critical. Unfortunately, the LLM pre-training does not consider these relationships specifically, and there is no public data related to reasoning at the comment level in social media discourse. Thus, we rely on other general-purpose structured data as a surrogate to learn structure and reasoning. We use the WikiTableQuestions (Pasupat and Liang, 2015) dataset to infuse structural intelligence into the model. This dataset consists of various independent tables, questions based on one of the tables, and a corresponding answer. To answer these questions, using the data in the input table is vital. Some answers require aggregate reasoning.

For the directionality analysis task (which post is targeted by another comment in the same session), we leveraged a corpus of 4chan threads (Papasavva et al., 2020). This dataset consists of ∼3 million threads and ∼100 million posts. Since 4chan allows its users to tag whom they reply to, we use this data as the ground truth for directionality information. We examine whether our RAFT phases improve (i) the model's ability to detect the post we are targeting for behavior comprehension and (ii) the model's ability to distinguish who is being targeted by the poster. 4chan allows posters to mention more than one comment as the target of the reply. Here, we consider the model successful if one of the multiple comments is identified. Table 2 shows the result of our experiment. The RAFT model significantly improves performance over the pre-trained counterparts. This illustrates the application of RAFT to improve LLM performance in social media analysis. *Social media conversation threads can provide important context but they can span large sequences. JORA helps in the training process here by splitting a discourse sequence's computation tensors across multiple GPUs. This is not possible using HuggingFace/Deepspeed since Data-Parallelism in these frameworks distributes the workload between different data instances rather than dividing the computation for a single data instance among multiple accelerators.*

	Target Post	Reply Post	p(Reply | Target)
7B	0.082	0.153	0.643
13B	0.159	0.200	0.815
7B-RAFT	0.865	0.541	0.558
13B-RAFT	**0.971**	**0.847**	**0.855**

Table 2: The veracity of the directionality identification improves with the RAFT fine-tuning phases w.r.t. the baselines. Given the conversation as context, the values represent the accuracy of detecting the respective posts. Llama-2 models are used.

6 Conclusion

This paper presents JORA, a JAX-based library for Retrieval Augment fine-tuning of Llama-2 models. JORA provides convenient functions for data manipulation and training. In addition, it implements best practices for memory efficient and performant training. By using a combination of LoRA, tensor-parallelism, and jit, JORA can significantly improve memory efficiency and computation time over a distributed environment compared to Hugging Face/DeepSpeed. Finally, JORA can export trained models to the popular Hugging Face model format for downstream usage with other Hugging Face-compatible libraries.

Acknowledgements

This work is supported by Office of Naval Research awards #N00014-21-1-400 and #N00014-22-1-2291. L.C. is supported by NSF award #2312862 and a Cisco gift grant.

References

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

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. 2022. Improving language models by retrieving from trillions of tokens. pages 2206–2240.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. JAX: composable transformations of Python+NumPy programs.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading wikipedia to answer open-domain questions.

Christopher Clark and Matt Gardner. 2017. Simple and effective multi-paragraph reading comprehension.

Zihao Fu, Haoran Yang, Anthony Man-Cho So, Wai Lam, Lidong Bing, and Nigel Collier. 2023. On the effectiveness of parameter-efficient fine-tuning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12799–12807.

Kanishk Gandhi, Jan-Philipp Fränken, Tobias Gerstenberg, and Noah D Goodman. 2023. Understanding social reasoning in language models with language models. *arXiv preprint arXiv:2306.15448*.

Sylvain Gugger, Lysandre Debut, Thomas Wolf, Philipp Schmid, Zachary Mueller, Sourab Mangrulkar, Marc Sun, and Benjamin Bossan. 2022. Accelerate: Training and inference at scale made simple, efficient and adaptable. `https://github.com/huggingface/accelerate`.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. pages 3929–3938.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Gautier Izacard and Edouard Grave. 2020. Leveraging passage retrieval with generative models for open domain question answering. *arXiv preprint arXiv:2007.01282*.

Douglas M. Jennewein, Johnathan Lee, Chris Kurtz, Will Dizon, Ian Shaeffer, Alan Chapman, Alejandro Chiquete, Josh Burks, Amber Carlson, Natalie Mason, Arhat Kobwala, Thirugnanam Jagadeesan, Praful Barghav, Torey Battelle, Rebecca Belshe, Debra McCaffrey, Marisa Brazil, Chaitanya Inumella, Kirby Kuznia, Jade Buzinski, Sean Dudley, Dhruvil Shah, Gil Speyer, and Jason Yalim. 2023. The Sol Supercomputer at Arizona State University. In *Practice and Experience in Advanced Research Computing*, PEARC '23, pages 296–301, New York, NY, USA. Association for Computing Machinery.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2019. Generalization through memorization: Nearest neighbor language models. *arXiv preprint arXiv:1911.00172*.

Mojtaba Komeili, Kurt Shuster, and Jason Weston. 2021. Internet-augmented dialogue generation.

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent retrieval for weakly supervised open domain question answering.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020a. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020b. Retrieval-augmented generation for knowledge-intensive nlp tasks. volume 33, pages 9459–9474.

Jacob Menick, Maja Trebacz, Vladimir Mikulik, John Aslanides, Francis Song, Martin Chadwick, Mia Glaese, Susannah Young, Lucy Campbell-Gillingham, Geoffrey Irving, et al. 2022. Teaching language models to support answers with verified quotes. *arXiv preprint arXiv:2203.11147*.

Zak Murez, Tarrence Van As, James Bartolozzi, Ayan Sinha, Vijay Badrinarayanan, and Andrew Rabinovich. 2020. Atlas: End-to-end 3d scene reconstruction from posed images. In *Computer Vision–ECCV*

2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VII 16, pages 414–431. Springer.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. 2021. Webgpt: Browser-assisted question-answering with human feedback.

Antonis Papasavva, Savvas Zannettou, Emiliano De Cristofaro, Gianluca Stringhini, and Jeremy Blackburn. 2020. Raiders of the lost kek: 3.5 years of augmented 4chan posts from the politically incorrect board. In *Proceedings of the international AAAI conference on web and social media*, volume 14, pages 885–894.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. *arXiv preprint arXiv:1508.00305*.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506.

Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

Minjoon Seo, Jinhyuk Lee, Tom Kwiatkowski, Ankur P Parikh, Ali Farhadi, and Hannaneh Hajishirzi. 2019. Real-time open-domain question answering with dense-sparse phrase index. *arXiv preprint arXiv:1906.05807*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. 2024. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. 2022. Lamda: Language models for dialog applications.

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*.

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