Natural Language Processing
Large Language Models Usage and Evaluation Patterns

Felipe Bravo-Marquez

December 8, 2023

Introduction

- Since the inception of Large Language Models, various patterns of use and evaluation of this technology have emerged.
- In this talk, we will try to organize these patterns and give a general overview of them.

Source:

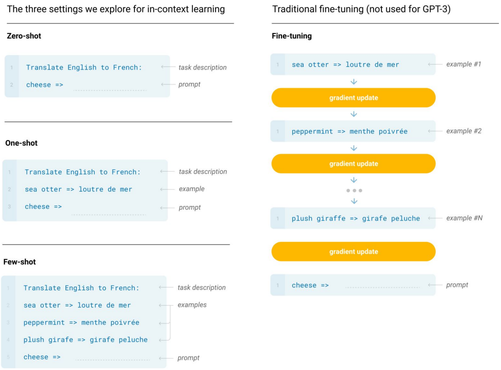
`https://www.masayume.it/img/masayume/Large-Language-Models.jpg`

Recap: What is an LLM

- Large Language Model: An autoregressive language model trained with a Transformer neural network on a large corpus (hundreds of bullions of tokens) and a large parameter space (billions) to predict the next word.
- It is usually later aligned to work as a user assistant using techniques such as Reinforcement Learning From Human Feedback [Ouyang et al., 2022] or supervised fine-tuning.
- Some are private (access via API or web browser): Google Bard, ChatGPT, etc.
- Others are open (model's weights can be downloaded): Llama, LLama2, Falcon, etc.

Zero-shot, One-shot, and Few-shot Learning

The most remarkable feature of these models is their few-shot, one-shot, zero-shot learning capabilities (also known as “in-context-learning”).



This means that they can learn new tasks without large amounts of human-annotated data.

Talk Overview

- Despite the recency of this technology, its adoption has been tremendous in many areas.
- Below, we propose a simple categorization of the ways in which LLMs are used and evaluated.
- These patterns will serve as the narrative backbone of this presentation.

Usage Patterns

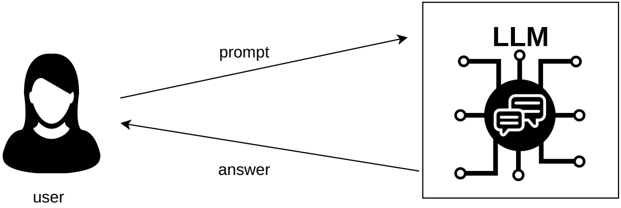
1. General-domain Assistant
2. Domain-specific Assistant
   2.1 Retrieval-augmented generation
   2.2 Fine-Tuning
3. LLM-based Applications
   3.1 API calls
   3.2 Autonomous Agents

Evaluation Patterns

- MTBench
- LLM Arena

Usage Pattern 1: General-domain Assistant

- In this pattern a user interacts with the LLM proving prompts as input and receiving a text as answer.
- The knowledge the LLM has access is limited to the corpus on which it was trained and the context given in the prompt.



Tasks

LLMs can solve many tasks with this pattern:

- Textual: Language understanding and common sense (e.g., rewriting, summarizing, translating, answering questions)
- Arithmetic: Mathematical reasoning (it can fail in many cases though)
- Visual: Multimodal reasoning involving pictures (GPT-4, Llava)
- Symbolic: Structured input such as programming languages

Source:

`https://twitter.com/IntuitMachine/status/1727079666001870877`.

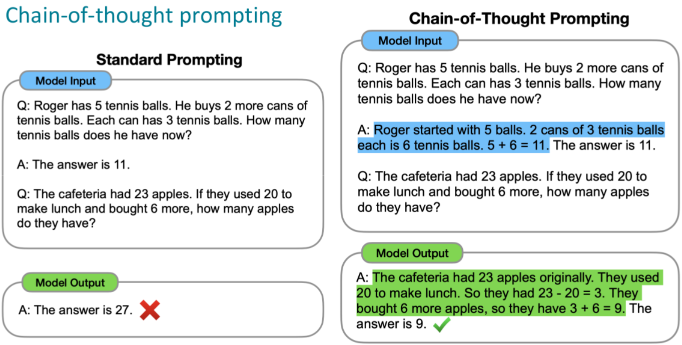
Prompt Engineering: Guiding the Language Model

Prompt engineering, often referred to as “Prompting,” is the discipline or “art” of crafting effective prompts to guide the Language Model (LM) towards generating accurate responses. Some common prompting guidelines:

- **Clarity and Conciseness**: Clearly articulate the prompt to minimize ambiguity and ensure the LM understands the task at hand.
- **Use of Specific Examples**: Provide concrete examples within the prompt to offer the LM context.
- **Role-based Prompts**: incorporating roles into the prompts (e.g., a tour guide, a teacher, a doctor, a sales person).
- **Desired Output Specification**: Clearly define the desired format of the output (e.g, JSON, HTML, csv, markdown, latex).

Chain-of-thought Prompting

- Chain-of-thought prompting is a mechanism for eliciting multi-step reasoning behavior in large language models.
- This method involves augmenting each exemplar in a few-shot prompt with a connected sequence of thoughts, creating a structured chain of logical steps. [Wei et al., 2022]

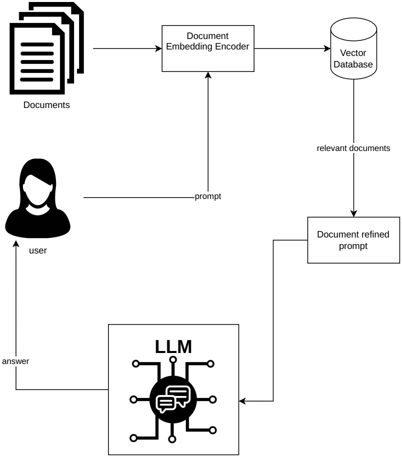


Usage Pattern 2: Domain-specific Assistant

- Idea: Incorporate domain-specific knowledge not covered in training (e.g., recent news, private documents).
- This is very common for companies developing chatbots with private documents or for creating more domain-specific chatbots.
- There are two main patterns to achieve this:
  1. Retrieval-Augmented Generation (Vector Databases)
  2. Fine-Tuning

Usage Pattern 2.1: Retrieval-Augmented Generation

Idea: Incorporate domain-specific knowledge into the query using information retrieval and document embeddings (i.e., densely encoded vectors that capture the semantic information of the document). [Lewis et al., 2021].

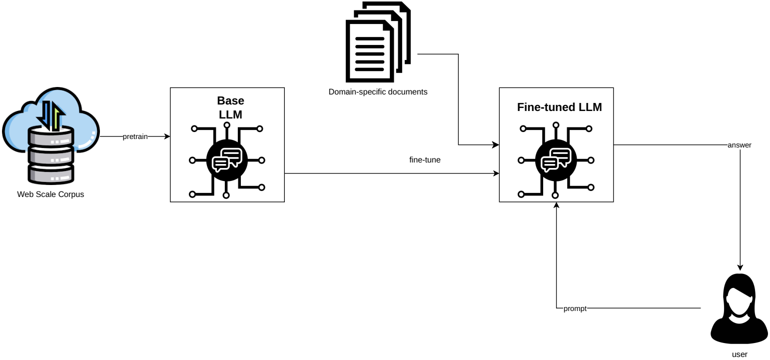


Retrieval-Augmented Generation Process

1. Encode all domain-specific documents using document embeddings and store them in a vector database.
    - OpenAI provides a vectorizer called (text2vec-openai), but there are many open source alternatives.
    - There are also many vector databases available, a popular one is **weaviate**.
2. Encode the prompt with the same vectorizer used to encode documents.
3. Use prompt embedding and the vector database to retrieve relevant documents based on similarity.
4. Create a refined prompt that includes a domain-specific role, the user prompt, and the retrieved documents as contexts.
5. Send the refined prompt to the LLM and return the response to the user.

Usage Pattern 2.2: Fine-Tuning

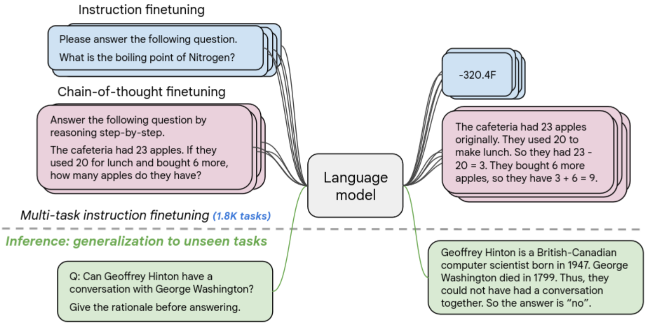
Idea: Incorporate domain-specific knowledge by fine-tuning a pre-trained LLM with the next-token prediction task over a domain-specific corpus and interact with the resulting LLM.



This can be computationally expensive unless some tricks are used.

Instruction Fine-tuning

- A more efficient way to fine-tune Large Language Models is Instruction Fine-Tuning [Chung et al., 2022].
- Idea: instead of fine-tuning the LLM with raw text with next token prediction, train it with pairs of prompts and user-aligned answers.
- Collect examples of (instruction, output) pairs across many tasks and finetune an LM.
- Evaluate on unseen tasks.



Datasets for Instruction Fine-Tuning

- Alpaca Data: 52k English instruction examples generated using OpenAI's text-davinci-003 with self-instruct.
- Evol-Instruct (Xu et al., April 2023): A rewritten set of 250k English instruction-response pairs based on the Alpaca data
- Vicuna ShareGPT: 70k English conversations shared by users and scraped from
- Baize data (Xu et al., April 2023)
- databricks-dolly-15k (Conover et al., April 2023)
- OpenAssistant Conversations (Köpf et al., April 2023)
- LIMA data (Zhou et al., May 2023).

source:

https://nlpnewsletter.substack.com/p/instruction-tuning-vol-2

Considerations for Instruction Tuning

- **Data Source:** How was the data obtained? Most datasets have been generated using ChatGPT. They may thus inherit biases of the source model or may be noisy.
- **Data Quality:** Was any filtering done to improve the quality of the generated data? In most cases, filtering is based on simple heuristics or a pre-trained model, which can result in noisy data.
- **Domain and Language Coverage:** Most datasets focus on general QA-style in English, but methods can be adapted for other domains or languages.
- **Number of Dialog Turns:** Single-turn datasets include a prompt and response. Consider multi-turn data for training a more conversational model.
- **License Terms:** Data from OpenAI models follows OpenAI terms, restricting use for competing models. Seek datasets with more permissive licenses to avoid legal complications.

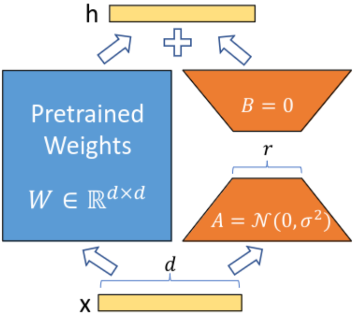
source:
`https://nlpnewsletter.substack.com/p/instruction-tuning-vol-2`

Parameter Efficient Fine Tuning (PEFT)

- Traditional model fine-tuning (i.e., adjusting all Transformer weights via backpropagation) is time and resource-intensive.
- Parameter Efficient Fine Tuning (PEFT) are a series of techniques to mitigate this problem.
- The most popular ones are LoRA (Low Rank Adaptation) [Hu et al., 2021] and QLoRA [Dettmers et al., 2023].
- These approaches achieve siginficant reduction in the number of trainable parameters allowing to perform fine-tuning with affordable hardware.

LoRA

- LoRa is based on the idea of adapters.
- Instead of fine-tuning the whole network, we freeze it during the fine-tuning process and add a few parameters that are trained to adapt the original model to the new data.



LoRA

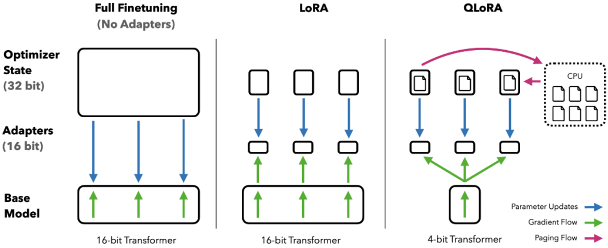
- Let $W_0 \in \mathbb{R}^{d \times k}$ be the weight matrix of the pre-trained network.
- LoRA uses two adjustable low-rank decomposition matrices of predefined rank r, $B \in \mathbb{R}^{d \times r}$ and $A \in \mathbb{R}^{r \times k}$.
- The resulting fine-tuned network W_L is obtained as follows:

$$W_L = W_0 + \Delta W = W_0 + BA, \quad B \in \mathbb{R}^{d \times r}, \quad A \in \mathbb{R}^{r \times d} \tag{1}$$

- During training, W_0 remains frozen, and only B and A receive gradient updates.
- The lower the value of r, the fewer parameters will be trained during fine-tuning (there is a trade-off between performance and computation costs).

QLoRA

- QLoRA is an even more memory efficient version of LoRA where the pretrained model is loaded to GPU memory as **quantized** 4-bit weights.
- QLoRA dequantizes weights from the storage data type to the computation data type to perform the forward and backward passes.
- It only computes weight gradients for the LoRA parameters which use 16-bit bfloat.
- QLORA reduces the average memory requirements of finetuning a 65B parameter model from $> 780GB$ of GPU memory to $< 48GB$.



Quantization

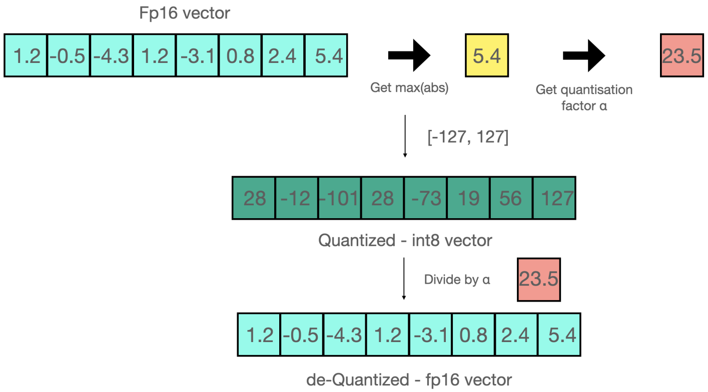
- Quantization: Process of discretizing input from high-bit representation to low-bit representation.
- To ensure that the entire range of the low-bit data type is used, the input data type is commonly rescaled into the target data type range through normalization by the absolute maximum of the input elements.
- Example: quantizing a 32-bit Floating Point (FP32) tensor into a Int8 tensor with range $[-127, 127]$:

$$X_{\text{Int8}} = \text{round}_{\text{digits}(0)}\left(\frac{127}{\text{absmax}(X_{\text{FP32}})} \cdot X_{\text{FP32}}\right) = \text{round}_{\text{digits}(0)}\left(c_{\text{FP32}} \cdot X_{\text{FP32}}\right)$$

  where c is the quantization constant or quantization scale.
- Dequantization:

$$\text{dequant}(c_{\text{FP32}}, X_{\text{Int8}}) = \frac{X_{\text{Int8}}}{c_{\text{FP32}}} = X_{\text{FP32}}$$

Quantization



source: `https://huggingface.co/blog/hf-bitsandbytes-integration`

QLoRA main Ideas

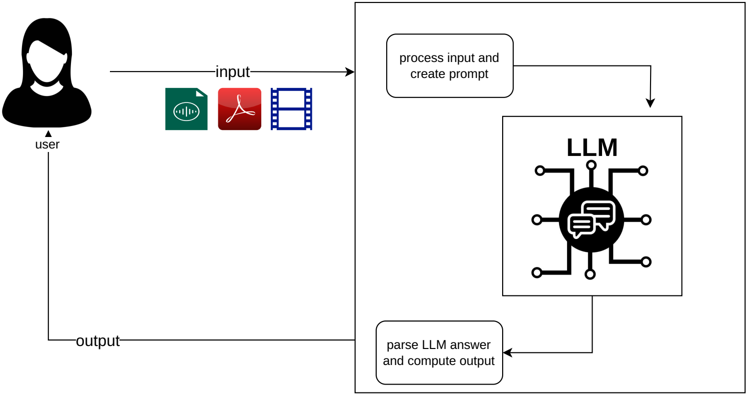
- 4-bit NormalFloat, an information theoretically optimal quantization data type for normally distributed data[1] that yields better empirical results than 4-bit Integers and 4-bit Floats.
- Double Quantization, a method that quantizes the quantization constants, saving an average of about 0.37 bits per parameter (approximately 3 GB for a 65B model).
- Paged Optimizers, using NVIDIA unified memory to avoid the gradient checkpointing memory spikes that occur when processing a mini-batch with a long sequence length.

[1] pretrained neural network weights usually have a zero-centered normal distribution with standard deviation σ

Usage Pattern 3: Applications

- LLMs are widely embedded in software through API calls.
- Why? Because LLM's apart from generating text cannot perform other type of actions.
- Example: search engines (`you.com`), Bing chat.
- For example, PDF summarization software. You write software that first converts the PDF to raw text and then sends it to an LLM for summarization.
- Or software for summarizing video conferences. First the audio is transcribed and then summarized with an LLM.
- From a software point of view, this isn't much different from any software that interacts with an external API.
- ChatGPT Plugins: `https://gptstore.ai/`

Usage Pattern 3: Applications



Usage Pattern 3.1: Autonomous Agents

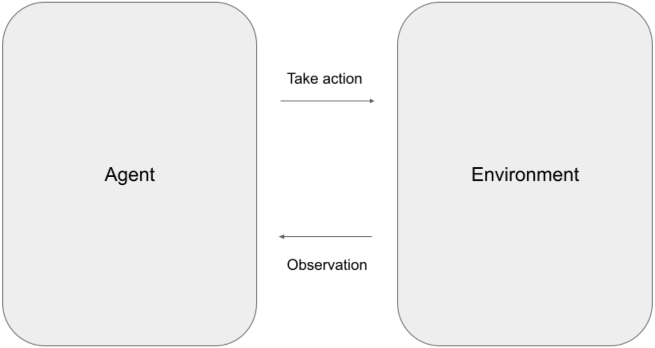
- Autonomous agents lie at the heart of classical AI.
- Agents are programs that interact autonomously with an environment and perform actions to achieve specific goals.
- The ReaAct (Reasoning and Acting) pattern [**?**] is a framework is which LLMs are used to generate both reasoning traces and task-specific actions in an interleaved manner for agents.
- Agents are a special kind of LLMs application in which the LLM serves as the reasoning and planning component of the software.
- LangChain is a very popular python Library for implementing LLM-based agents.

LangChain

General setup of an agent

- Consider a general setup of an agent interacting with an environment to solve a task.
- There are several discrete time steps $t_0, t_1, \ldots, t_n$.
- There is a set of all possible states or observations O received from the environment.
- There is a set of actions A that the agent can take.
- At each time step t, the agent receives an observation $o_t \in O$ and takes the action $a_t \in A$ following a policy $\pi(a_t|c_t)$ where context $c_t = (o_1, a_1, \ldots, o_{t-1}, a_{t-1}, o_t)$.
- Learning a policy is challenging when the mapping $c_t \mapsto a_t$ is highly implicit and requires extensive computation.

General setup of an agent



Source: `https://leftasexercise.com/2023/06/17/autonomous-agents-and-llms-autogpt-langchain-and-all-that/`

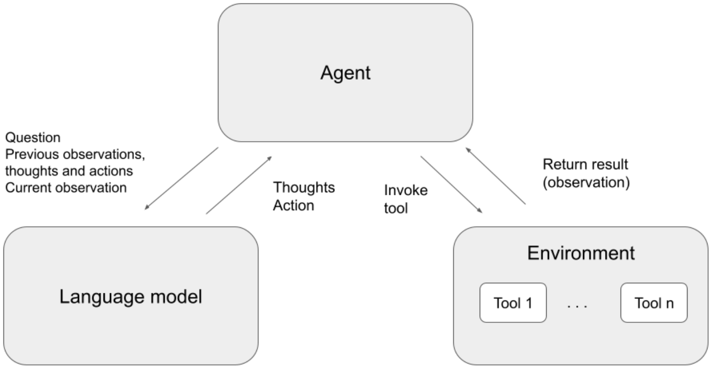
ReAct: Synergizing Reasoning + Acting

- In the ReAct framework, we expand the agent's action space by incorporating a Language Model (LLM), denoted as $A' = A \cup L$ (with L representing the language space).
- The environment is conceptualized as a toolkit, consisting of diverse interfaces that the agent can leverage.
- These interfaces take various forms, including API calls and Python functions (such as a search engine, calculator, or web scraper).
- At each step, the agent consults the LLM (using chain of thought prompting) to determine its next action, considering the available tools.

ReAct: Synergizing Reasoning + Acting

- This action, denoted as $a'_t \in L$ within the language space, is termed a “thought”.
- The context is dynamically updated as $c_{t+1} = (c_t, a'_t)$ for subsequent reasoning or action.
- The agent then executes one of the available tools to perform the action, incorporating the result as a new observation in the next prompt sent to the model.
- This iterative process continues until the task is successfully solved.
- Problem: if the model starts to hallucinate or to repeat itself, we can easily follow a trajectory of actions that takes us nowhere or end up in a loop.

ReAct Agents



Source: `https://leftasexercise.com/2023/06/17/autonomous-agents-and-llms-autogpt-langchain-and-all-that/`

LLMBench and LLm Arena

- How standard mono-task NLP systems are usually evaluated?
- There is a gold-standard evaluation dataset not used during training on which predictions can be compared against gold labels using tasks-specific metrics (e.g., F1, BLEU).
- LLMS are intrinsically multi-task and open-ended.
- Hence, cannot easily be evaluated with this approach.
- Another caveat is that aligned LLM
- Machines evaluating machines??
- MT-bench (categories): a multi-turn question set,
- HuggingFace Open LLM Leaderboard
- Chatbot Arena: a crowdsourced battle platform between LLMs.
- Evaluation pattern: LLM-as-a-judge

Testing Setup

- 'The Vicuna Team' for evaluating Vicuña and FastChat-T5 models used 80 questions spanning topics like code generation, general knowledge, common sense, counterfactuals, and mathematical reasoning.
- GPT-4 used for comparison and scoring.
- Specific prompts assigned to assess responses.
- Example prompt for evaluating general knowledge responses.

```
{"prompt_id": 1, "system_prompt": "You are a helpful and
precise assistant for checking the quality of the answer.",
...
```

LLM Leaderboard by HuggingFace

- Responding to the surge in LLMs and chatbots, HuggingFace introduces the LLM Leaderboard.
- Open for community members to submit their transformer-type models for evaluation.
- Two evaluation forms: LLM Benchmarks and Human/GPT Evaluations.

LLM Benchmarks

- Evaluation based on 4 key benchmarks from Eleuther AI Language Model Evaluation Harness.
  - AI2 Reasoning Challenge (25-shot)
  - HellaSwag (10-shot)
  - MMLU (5-shot)
  - TruthfulQA (0-shot)
- Models tested on tasks covering science, common sense inference, multitasking accuracy, and truthfulness.

Overview of Base LLM Evaluation

- Typical benchmarks for base LLMs involve close-ended problems, such as multiple-choice questions.
- Two main categories: knowledge-oriented and reasoning-oriented benchmarks.
- Recent benchmarks (e.g., OpenCompass) combine both types for comprehensive evaluation.

Knowledge-Oriented Benchmarks

- Examples: MMLU [364] and C-Eval [711].
- Evaluate the capacity of world knowledge.

Reasoning-Oriented Benchmarks

- Examples: GSM8K [643], BBH [365], and MATH [364].
- Focus on evaluating the capability of solving complex reasoning tasks.

Combined Benchmarks

- Example: OpenCompass [713].
- Comprehensive comparison by combining both knowledge-oriented and reasoning-oriented aspects.

Benchmark-Based Evaluation Procedure

- Format each problem into a prompt for LLMs to generate the result text.
- Parse the generated result text using human-written rules to obtain the predicted answer.
- Calculate LLMs' performance automatically using standard metrics like accuracy by comparing with the ground-truth answer.

Few-Shot or Zero-Shot Setting

- Evaluation can be conducted in either the few-shot or zero-shot setting.
- Few-shot setting is often more suitable for base LLMs due to their weak task generalization ability.
- CoT prompts might be necessary for complex reasoning tasks to fully exhibit capacity during evaluation.

Application to Fine-Tuned LLMs

- This evaluation approach can be applied to assess the abilities of fine-tuned LLMs.
- Leaderboards (e.g., Open LLM Leaderboard [707]) use this approach to evaluate both base and fine-tuned LLMs.

Questions?

Thanks for your Attention!

References I

Chung, H. W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, Y., Wang, X., Dehghani, M., Brahma, S., Webson, A., Gu, S. S., Dai, Z., Suzgun, M., Chen, X., Chowdhery, A., Castro-Ros, A., Pellat, M., Robinson, K., Valter, D., Narang, S., Mishra, G., Yu, A., Zhao, V., Huang, Y., Dai, A., Yu, H., Petrov, S., Chi, E. H., Dean, J., Devlin, J., Roberts, A., Zhou, D., Le, Q. V., and Wei, J. (2022).
Scaling instruction-finetuned language models.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. (2023).
Qlora: Efficient finetuning of quantized llms.
arXiv preprint arXiv:2305.14314.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. (2021).
Lora: Low-rank adaptation of large language models.
arXiv preprint arXiv:2106.09685.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., tau Yih, W., Rocktäschel, T., Riedel, S., and Kiela, D. (2021).
Retrieval-augmented generation for knowledge-intensive nlp tasks.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. (2022).
Training language models to follow instructions with human feedback.
Advances in Neural Information Processing Systems, 35:27730–27744.

References II

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E., Le, Q., and Zhou, D. (2022).
Chain of thought prompting elicits reasoning in large language models.
arXiv preprint arXiv:2201.11903.