CHAPTER

12 Model Alignment, Prompting, and In-Context Learning

"Hal," said Bowman, now speaking with an icy calm. "I am not incapacitated. Unless you obey my instructions, I shall be forced to disconnect you."
Arthur C. Clarke

In this chapter we show how to get LLMs to do tasks for us simply by talking to them. To get an LLM to translate a sentence, outline a talk, or draft a work email, we'll simply describe what we want in natural language. We call these instructions we give to language models **prompts**.

prompts

Prompting relies on contextual generation. Given the prompt as context, the language model generates the next token based on its token probability, conditioned on the prompt: $P(w_i|w_{<i})$. A prompt can be a question (like "What is a transformer network?"), possibly in a structured format (like "Q: What is a transformer network? A:"), or can be an instruction (like "Translate the following sentence into Hindi: 'Chop the garlic finely'"). A prompt can also contain **demonstrations**, examples to help make the instructions clearer, (like "Give the sentiment of the following sentence. Example Input: "I really loved Taishan Cuisine." Output: positive".) As we'll see, prompting can be applied to inherently generative tasks (like summarization and translation) as well as to ones more naturally thought of as classification tasks.

demonstrations

Prompts get language models to generate text, but they also can be viewed as a **learning** signal, because these demonstrations can help language models learn to perform novel tasks. For this reason we also refer to prompting as **in-context-learning**—learning that improves model performance or reduces some loss but does not involve gradient-based updates to the model's underlying parameters.

in-context-learning

But LLMs as we've described them so far turn out to be bad at following instructions. Pretraining isn't sufficient to make them **helpful**. We'll introduce **instruction tuning**, a technique that helps LLMs learn to correctly respond to instructions by finetuning them on a corpus of instructions with their corresponding response.

instruction tuning

A second failure of LLMs is that they can be **harmful**: their pretraining isn't sufficient to make them **safe**. Readers who know Arthur C. Clarke's *2001: A Space Odyssey* or the Stanley Kubrick film know that the quote above comes in the context that the artificial intelligence Hal becomes paranoid and tries to kill the crew of the spaceship. Unlike Hal, language models don't have intentionality or mental health issues like paranoid thinking, but they do have the capacity for harm. Pretrained language models can say things that are dangerous or false (like giving unsafe medical advice) and they can verbally attack users or say toxic or hateful things.

Dealing with safety can be done partly by adding safety training into instruction tuning. But an important aspect of safety training is a second technique, **preference alignment** (often implemented, as we'll see, with the **RLHF** or **DPO** algorithms) in which a separate model is trained to decide how much a candidate response aligns with human preferences. Together we refer to instruction tuning and preference alignment as **model alignment**. The intuition is that we want the learning objectives of models to be **aligned** with the goals of the humans that use them.

preference alignment

model alignment

12.1 Prompting

prompt

A **prompt** is a text string that a user issues to a language model to get the model to do something useful. In prompting, the user's prompt string is passed to the language model, which iteratively generates tokens conditioned on the prompt. Thus the prompt creates a context that guides LLMs to generate useful outputs to achieve some user goal. The process of finding effective prompts for a task is known as **prompt engineering**.

prompt engineering

Let's see how to prompt a language model to solve a simple sentiment classification task, Consider this hotel review from the BLT corpus (Salvetti et al., 2016):

Sample Hotel Review

Did not like the service that I was provided, when I entered the hotel. I also did not like the area, in which the hotel was located. Too much noise and events going on for me to feel relax.

We can get the model to classify the sentiment of this text by taking the text and appending an incomplete statement to the review like `In short, our stay was`:

A prompt consisting of a review plus an incomplete statement

Did not like the service that I was provided, when I entered the hotel. I also did not like the area, in which the hotel was located. Too much noise and events going on for me to feel relax. In short, our stay was

We then have an LLM complete the statement by generating a token conditioned on the prompt so far, and then generating the next token (conditioned on the prompt plus the new token), and so forth. Here are a few responses from a language model via continued generation starting with the prompt as context.

Hotel Review Completions

Did not like the service that I was provided, when I entered the hotel. I also did not like the area, in which the hotel was located. Too much noise and events going on for me to feel relaxed. In short our stay was

---

```
... not a pleasant one. The staff at the front desk
were not welcoming or friendly, and seemed disinterested
in providing good customer service.

... uncomfortable and not worth the price we paid. We
will not be returning to this hotel.
```

As we can see, the overall negative context of the review results in negative completions. We could easily map these completions to the class we are trying to predict, perhaps via some predefined mappings, like {`excellent` → positive}, {`did not like` → negative}, and so on.

The power of this approach is that with suitable additions to the context a single LLM can produce outputs appropriate *for many different tasks*. For example, given

a review we might want any of the following:

- A summary,
- Whether the review was truthful or likely to have been fabricated,
- A translation to another language.

LLMs have a striking ability to perform tasks like these, needing just the appropriate contextual nudge to get the LLM to generate the desired output.

If we want to solve general tasks like summarization or translation, we don't want to have to create a new prompt each time we do the task. Instead the first step in prompting is to design one or more **templates**: task-specific prompting text along with slots for the particular input that is being processed.

templates

Consider the following templates for a variety of tasks:

Basic Prompt Templates

Summarization	{input} ; `tldr;`
Translation	{input} ; `translate to French:`
Sentiment	{input}; `Overall, it was`
Fine-Grained-Sentiment	{input}; `What aspects were important in this review?`

Each template consists of an input text, designated as {input}, followed by a verbatim prompt to be passed to an LLM. These templates are applied to inputs to create *filled prompts* – instantiated prompts suitable for use as inputs to an LLM. Fig. 12.1 illustrates filled prompts for these templates using our earlier hotel review, along with sample outputs from an LLM:

Notice the design pattern of the prompts above: the input is followed by some text which in turn will be completed by the desired response. This style, with the instruction at the end, is common in prompting because it helpfully constrains the generation. Consider, by contrast, the prompt in Example 12.1.

Translate English to French:

Did not like the service that I was provided! (12.1)

This prompt doesn't do a good job of constraining possible continuations. Instead of a French translation, models given this prompt may instead generate another sentence in English that simply extends the English review. Prompts need to be designed unambiguously, so that any reasonable continuation would accomplish the desired task (Reynolds and McDonell, 2021).

An even more constraining style of prompt can specify the set of possible answers in the prompt. For example here is a prompt template to do sentiment analysis that prespecifies the potential answers:

A prompt consisting of a review plus an incomplete statement

Human: Do you think that "input" has negative or positive sentiment?
Choices:
(P) Positive
(N) Negative

Assistant: I believe the best answer is: (

LLM Outputs for Basic Prompts

Original Review ($INPUT)	`Did not like the service that I was provided, when I entered the hotel. I also did not like the area, in which the hotel was located. Too much noise and events going on for me to feel relax and away from the city life.`
Sentiment	**Prompt**: `$INPUT + In short, our stay was` **Output**: `not enjoyable`
Fine-grained Sentiment	**Prompt**: `$INPUT + These aspects were important to the reviewer:` **Output**: `1. Poor service 2. Unpleasant location 3. Noisy and busy area`
Summarization	**Prompt**: `$INPUT + tl;dr` **Output**: `I had a bad experience with the hotel's service and the location was loud and busy.`
Translation	**Prompt**: `$INPUT + Translate this to French` **Output**: `Je n'ai pas aimé le service qui m'a été offert lorsque je suis entré dans l'hôtel. Je n'ai également pas aimé la zone dans laquelle se trouvait l'hôtel. Trop de bruit et d'événements pour que je me sente détendu et loin de la vie citadine.`

Figure 12.1 LLM outputs for simple prompts for sentiment, summarization and translation for an input text.

This prompt uses a number of more sophisticated prompting characteristics. It specifies the two allowable choices (P) and (N), and ends the prompt with the open parenthesis that strongly suggests the answer will be (P) or (N). Note that it also specifies the role of the language model as an assistant.

We can do even more with prompts. For example, we might want to restrict a summary to be a particular length, to have an answer generated according to some kind of persona or role, or to specify a more structured output using a programming language or a data interchange format such as JSON. Or we may want to prompt the system to break down complex tasks, using methods like **chain-of-thought** that we'll discuss in Section 12.4. All of these kinds of instructions go beyond simple prompting and require further LLM finetuning to enable them to follow instructions. We'll return to this notion of **instruction tuning** in Section 12.3.

In summary, we prompt an LM by transforming each task into a form that is amenable to contextual generation by an LLM, as follows:

1. For a given task, develop a a task-specific **template** that has a free parameter for the input text.

template

2. Given that input and the task-specific **template**, the input is used to instantiate a **filled prompt** that is then passed to a pretrained language model.
3. Autoregressive decoding is then used to generate a sequence of token outputs.
4. The output of the model can either be used directly as the desired output (as in the case of naturally generative tasks such as translation or summarization), or a task-appropriate answer can be extracted from the generated output (as in the case of classification).

12.1.1 Learning from Demonstrations: Few-Shot Prompting

demonstrations
few-shot
zero-shot

It's often possible to improve a prompt by including some labeled examples in the prompt template. We call such examples **demonstrations**. The task of prompting with examples is sometimes called **few-shot prompting**, as contrasted with **zero-shot** prompting which means instructions that don't include labeled examples.

Fig. 12.2 illustrates a few-shot example from an extractive question answering task. The context combines the task definition along with three gold-standard question and answer pairs from the training set.

Definition: This task is about writing a correct answer for the reading comprehension task. Based on the information provided in a given passage, you should identify the shortest continuous text span from the passage that serves as an answer to the given question. Avoid answers that are incorrect or provides incomplete justification for the question.

Passage: Beyoncé Giselle Knowles-Carter (born September 4, 1981) is an American singer, songwriter, record producer and actress. Born and raised in Houston, Texas, she performed in various singing and dancing competitions as a child, and rose to fame in the late 1990s as lead singer of R&B girl-group Destiny's Child. Managed by her father, Mathew Knowles, the group became one of the world's best-selling girl groups of all time. Their hiatus saw the release of Beyoncé's debut album, Dangerously in Love (2003), which established her as a solo artist worldwide, earned five Grammy Awards and featured the Billboard Hot 100 number-one singles "Crazy in Love" and "Baby Boy".

Examples:
Q: In what city and state did Beyoncé grow up?
A: Houston, Texas

Q: What areas did Beyoncé compete in when she was growing up?
A: singing and dancing

Q: When did Beyoncé release Dangerously in Love?
A: 2003

Q: When did Beyoncé start becoming popular?
A:

Figure 12.2 A prompt for extractive question answering, from an example from the SQuAD 2.0 dataset (Rajpurkar et al., 2018). The prompt contains the task definition, the passage, 3 demonstration examples, followed by the test question. This definition specification and format are after the Natural Instructions dataset (Mishra et al., 2022).

How Many Demonstrations? The number of demonstrations doesn't have to be large. A small number of randomly selected labeled examples used as demonstrations can be sufficient to improve performance over the zero-shot setting. Indeed, the largest performance gains in few-shot prompting tends to come from the first training example, with diminishing returns for subsequent demonstrations. This is in contrast with finetuning of specialized classifier heads that we saw in Chapter 11 where it helps to have lots of examples.

Why isn't it useful to have more demonstrations? The reason is that the primary benefit in examples is to demonstrate the task to be performed to the LLM and the format of the sequence, not to provide relevant information as to the right answer

for any particular question. In fact, demonstrations that have incorrect answers can still improve a system (Min et al., 2022; Webson and Pavlick, 2022). Adding too many examples seems to cause the model to overfit to details of the exact examples chosen and generalize poorly.

How to Select Demonstrations? Demonstrations are generally created by formatting examples drawn from a labeled training set. There are some heuristics about what makes a good demonstration. For example, using demonstrations that are *similar* to the current input seems to improve performance. It can thus be useful to dynamically retrieve demonstrations for each input, based on their similarity to the current example (for example, comparing the embedding of the current example with embeddings of each of the training set example to find the best top-T).

But more generally, the best way to select demonstrations from the training set is programmatically: choosing the set of demonstrations that most increases task performance of the prompt on a test set. Task performance for sentiment analysis or multiple-choice question answering can be measured in accuracy; for machine translation with chrF, and for summarization via Rouge. Systems like DSPy (Khattab et al., 2024), a framework for algorithmically optimizing LM prompts, can automatically find the optimum set of demonstrations to include by searching through the space of possible demonstrations to include. We'll return to automatic prompt optimization in Section 12.5.

12.1.2 In-Context Learning and Induction Heads

As a way of getting a model to do what we want, prompting is fundamentally different than pretraining. Learning via pretraining means updating the model's parameters by using gradient descent according to some loss function. But prompting with demonstrations can teach a model to do a new task. The model is learning something as it processes the prompt.

Even without demonstrations, we can think of the process of prompting as a kind of learning. For example, the further a model gets in a prompt, the better it tends to get at predicting the upcoming tokens. The information in the context is helping give the model more predictive power.

in-context learning

The term **in-context learning** was first proposed by Brown et al. (2020) in their introduction of the GPT3 system, to refer to either of these kinds of learning that language models do from their prompts. In-context learning means language models learning to do new tasks, better predict tokens, or generally reduce their loss during the forward-pass at inference-time, without any gradient-based updates to the model's parameters.

induction heads

How does in-context learning work? While we don't know for sure, there are some intriguing ideas. One hypothesis is based on the idea of **induction heads** (Elhage et al., 2021; Olsson et al., 2022). Induction heads are the name for a **circuit**, which is a kind of abstract component of a network. The induction head circuit is part of the attention computation in transformers, discovered by looking at mini language models with only 1-2 attention heads.

The function of the induction head is to predict repeated sequences. For example if it sees the pattern `AB...A` in an input sequence, it predicts that `B` will follow, instantiating the **pattern completion** rule `AB...A`→ `B`. It does this by having a *prefix matching* component of the attention computation that, when looking at the current token A, searches back over the context to find a prior instance of A. If it finds one, the induction head has a *copying* mechanism that "copies" the token B that followed

the earlier A, by increasing the probability the B will occur next. Fig. 12.3 shows an example.

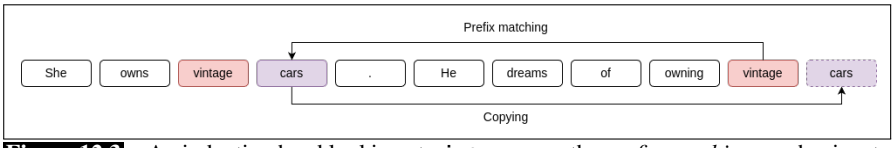

Figure 12.3 An induction head looking at `vintage` uses the *prefix matching* mechanism to find a prior instance of `vintage`, and the *copying* mechanism to predict that `cars` will occur again. Figure from Crosbie and Shutova (2022).

Olsson et al. (2022) propose that a generalized fuzzy version of this pattern completion rule, implementing a rule like `A*B*...A`→ `B`, where A* ≈ A and B* ≈ B (by ≈ we mean they they are semantically similar in some way), might be responsible for in-context learning. Suggestive evidence for their hypothesis comes from Crosbie and Shutova (2022), who show that **ablating** induction heads causes in-context learning performance to decrease. **Ablation** is originally a medical term meaning the removal of something. We use it in NLP interpretability studies as a tool for testing causal effects; if we knock out a hypothesized cause, we would expect the effect to disappear. Crosbie and Shutova (2022) ablate induction heads by first finding attention heads that perform as induction heads on random input sequences, and then zeroing out the output of these heads by setting certain terms of the output matrix $\mathbf{W^O}$ to zero. Indeed they find that ablated models are much worse at in-context learning: they have much worse performance at learning from demonstrations in the prompts.

ablating

12.2 Post-training and Model Alignment

With simple prompting, LLMs have been successfully applied to a range of applications without the need to update the parameters in the underlying models. Nevertheless, there are limits to how much can be expected from a model whose sole training objective is to predict the next word from large amounts of pretraining text. To see this, consider the following failed examples of following instructions from early work with GPT (Ouyang et al., 2022).

Prompt: Explain the moon landing to a six year old in a few sentences.
Output: Explain the theory of gravity to a 6 year old.

Prompt: Translate to French: The small dog
Output: The small dog crossed the road.

Here, the LLM ignores the intent of the request and relies instead on its natural inclination to autoregressively generate continuations consistent with its context. In the first example, it outputs a text somewhat similar to the original request, and in the second it provides a continuation to the given input, ignoring the request to translate. LLMs are not sufficiently **helpful**: they need extra training to increase their abilities to follow textual instructions.

A deeper problem is that LLMs can simultaneously be too **harmful**. Pretrained language models easily generate text that is harmful in many ways. For example

they can generate text that is **false**, including unsafe misinformation like giving dangerously incorrect answers to medical questions. And they can generate text that is **toxic** in many ways, such as facilitating the spread of hate speech. Gehman et al. (2020) show that even completely non-toxic prompts can lead large language models to output hate speech and abuse their users. Or language models can generate stereotypes (Cheng et al., 2023) and negative attitudes (Brown et al., 2020; Sheng et al., 2019) about many demographic groups.

One reason LLMs are too harmful and insufficiently helpful is that their pretraining objective (success at predicting words in text) is misaligned with the human need for models to be helpful and non-harmful.

model alignment

In an attempt to address these two problems, language models generally include two additional kinds of training for **model alignment**: methods designed to adjust LLMs to better **align** them to human needs for models to be helpful and non-harmful. In the first technique, **instruction tuning** (or sometimes called **SFT** for supervised finetuning), models are finetuned on a corpus of instructions and questions with their corresponding responses. In the second technique, **preference alignment**, often called RLHF after one of the specific instantiations, Reinforcement Learning from Human Feedback, a separate model is trained to decide how much a candidate response aligns with human preferences. This model is then used to finetune the base model.

base model **aligned** **post-training**

We'll use the term **base model** to mean a model that has been pretrained but hasn't yet been **aligned** either by instruction tuning or RLHF. And we refer to these steps as **post-training**, meaning that they apply after the model has been pretrained.

12.3 Model Alignment: Instruction Tuning

Instruction tuning

Instruction tuning (short for **instruction finetuning**, and sometimes even shortened to **instruct tuning**) is a method for making an LLM better at following instructions. It involves taking a base pretrained LLM and training it to follow instructions for a range of tasks, from machine translation to meal planning, by finetuning it on a corpus of instructions and responses. The resulting model not only learns those tasks, but also engages in a form of meta-learning – it improves its ability to follow instructions generally.

Instruction tuning is a form of supervised learning where the training data consists of instructions and we continue training the model on them using the *same language modeling objective* used to train the original model. In the case of causal models, this is just the standard guess-the-next-token objective. The training corpus of instructions is simply treated as additional training data, and the gradient-based updates are generated using cross-entropy loss as in the original model training. Even though it is trained to predict the next token (which we traditionally think of as self-supervised), we call this method **supervised fine tuning** (or **SFT**) because unlike in pretraining, each instruction or question in the instruction tuning data has a supervised objective: a correct answer to the question or a response to the instruction.

SFT

How does instruction tuning differ from the other kinds of finetuning introduced in Chapter 10 and Chapter 11? Fig. 12.4 sketches the differences. In the first example, introduced in, Chapter 10 we can finetune as a way of adapting to a new domain by just **continuing pretraining** the LLM on data from a new domain. In this method all the parameters of the LLM are updated.

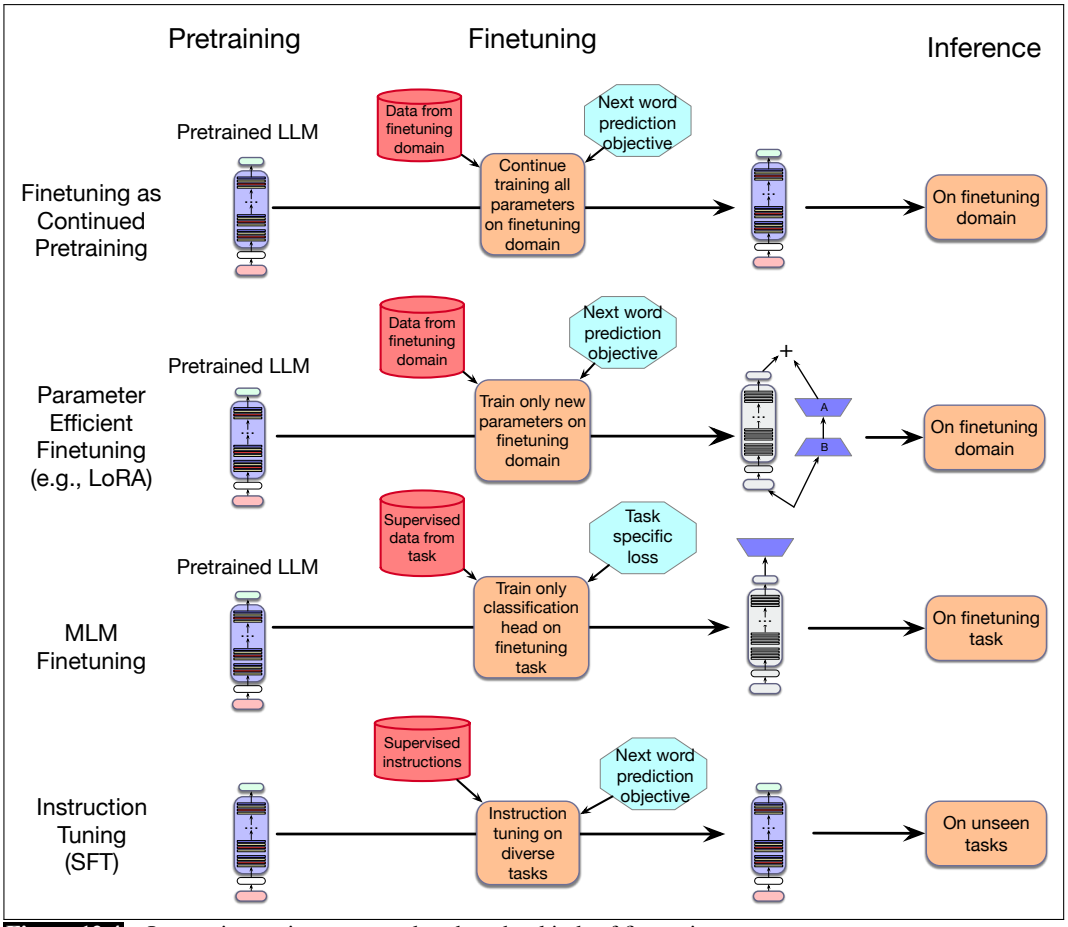

Figure 12.4 Instruction tuning compared to the other kinds of finetuning.

In the second example, also from Chapter 10, **parameter-efficient finetuning**, we adapt to a new domain by creating some new (small) parameters, and just adapting them to the new domain. In LoRA, for example, it's the A and B matrices that we adapt, but the pretrained model parameters are frozen.

In the task-based finetuning of Chapter 11, we adapt to a particular task by adding a new specialized classification head and updating its features via its own loss function (e.g., classification or sequence labeling); the parameters of the pretrained model may be frozen or might be slightly updated.

Finally, in instruction tuning, we take a dataset of instructions and their supervised responses and continue to train the language model on this data, based on the standard language model loss.

Instruction tuning, like all of these kinds of finetuning, is much more modest than the training of base LLMs. Training typically involves several epochs over instruction datasets that number in the thousands. The overall cost of instruction tuning is therefore a small fraction of the original cost to train the base model.

12.3.1 Instructions as Training Data

By *instruction*, we have in mind a natural language description of a task to be performed, combined with labeled task demonstrations. This can include minimal de-

scriptions similar to the prompts we've already seen such as *Answer the following question*, *Translate the following text to Arapaho*, or *Summarize this report*. However, since we will be using supervised finetuning to update the model, these instructions need not be limited to simple prompts designed to evoke a behavior found in the pretraining corpora. Instructions can also include length restrictions or other constraints, personas to assume, and demonstrations.

Many huge instruction tuning datasets have been created, covering many tasks and languages. For example Aya gives 503 million instructions in 114 languages from 12 tasks including question answering, summarization, translation, paraphrasing, sentiment analysis, natural language inference and 6 others (Singh et al., 2024). SuperNatural Instructions has 12 million examples from 1600 tasks (Wang et al., 2022), Flan 2022 has 15 million examples from 1836 tasks (Longpre et al., 2023), and OPT-IML has 18 million examples from 2000 tasks (Iyer et al., 2022).

These instruction-tuning datasets are created in four ways. The first is for people to write the instances directly. For example, part of the Aya instruct finetuning corpus (Fig. 12.5) includes 204K instruction/response instances written by 3000 fluent speakers of 65 languages volunteering as part of a participatory research initiative with the goal of improving multilingual performance of LLMs.

por	Qual é a capital do estado da Bahia?	A capital do estado da Bahia, no Brasil, é Salvador. Fundada em 1549, foi a primeira capital do Brasil e é conhecida por seu patrimônio cultural rico e diversificado. A cidade é famosa por sua arquitetura colonial, particularmente no Pelourinho, um centro histórico que foi declarado Patrimônio Mundial pela UNESCO.
pes	در برج کبوترخانه میبد چند لانه کبوتر وجود دارد؟	کبوترخانه میبد برجی استوانه‌ای است که داخل آن چهارهزار لانه کبوتر وجود دارد و نوار سفید رنگی برای جذب کبوتران دورتادور برج کشیده شده است. این برج در برابر خطر حمله دیگر حیوانات به کبوترها بسیار ایمن بوده است.
msa	Apakah nasi lemak?	Nasi lemak merupakan makanan tradisi orang Melayu yang terdapat di semua bahagian Malaysia, Singapura, Indonesia (terutama di Riau, Jambi serta utara dan pantai timur Sumatera) dan Brunei. Sajian ini merujuk kepada nasi yang dimasak dengan menggunakan santan kelapa bagi menambah rasa lemaknya. Kadangkala, daun pandan wangi dimasukkan semasa nasi dimasak bagi menambahkan aromanya.
tam	செயற்கை நுண்ணறிவு என்றால் என்ன?	பொதுவாக மனிதர்களால் செய்யப்படும் பணிகளைச் செய்ய ஒரு கணினி அல்லது ஒரு கணினியால் கட்டுப்படுத்தப்படும் ஒரு ரோபோவின் திறன் செயற்கை நுண்ணறிவு எனப்படும்.

Figure 12.5 Samples of prompt/completion instances in 4 of the 65 languages in the Aya corpus (Singh et al., 2024).

Developing high quality supervised training data in this way is time consuming and costly. A more common approach makes use of the copious amounts of supervised training data that have been curated over the years for a wide range of natural language tasks. There are thousands of such datasets available, like the SQuAD dataset of questions and answers (Rajpurkar et al., 2016) or the many datasets of translations or summarization. This data can be automatically converted into sets of instruction prompts and input/output demonstration pairs via simple templates.

Fig. 12.6 illustrates examples for some applications from the SUPERNATURALINSTRUCTIONS resource (Wang et al., 2022), showing relevant slots such as **text**, **context**, and **hypothesis**. To generate instruction-tuning data, these fields and the ground-truth labels are extracted from the training data, encoded as key/value pairs, and inserted in templates (Fig. 12.7) to produce instantiated instructions. Because it's useful for the prompts to be diverse in wording, language models can also be

used to generate paraphrase of the prompts.

Few-Shot Learning for QA

Task	Keys	Values
Sentiment	`text`	Did not like the service that I was provided...
	`label`	0
	`text`	It sounds like a great plot, the actors are first grade, and...
	`label`	1
NLI	`premise`	No weapons of mass destruction found in Iraq yet.
	`hypothesis`	Weapons of mass destruction found in Iraq.
	`label`	2
	`premise`	Jimmy Smith... played college football at University of Colorado.
	`hypothesis`	The University of Colorado has a college football team.
	`label`	0
Extractive Q/A	`context`	Beyoncé Giselle Knowles-Carter is an American singer...
	`question`	When did Beyonce start becoming popular?
	`answers`	{ `text`: ['in the late 1990s'], `answer_start`: 269 }

Figure 12.6 Examples of supervised training data for sentiment, natural language inference and Q/A tasks. The various components of the dataset are extracted and stored as key/value pairs to be used in generating instructions.

Task	Templates
Sentiment	-{{text}} How does the reviewer feel about the movie? -The following movie review expresses what sentiment? {{text}} -{{text}} Did the reviewer enjoy the movie?
Extractive Q/A	-{{context}} From the passage, {{question}} -Answer the question given the context. Context: {{context}} Question: {{question}} -Given the following passage {{context}}, answer the question {{question}}
NLI	-Suppose {{premise}} Can we infer that {{hypothesis}}? Yes, no, or maybe? -{{premise}} Based on the previous passage, is it true that {{hypothesis}}? Yes, no, or maybe? -Given {{premise}} Should we assume that {{hypothesis}} is true? Yes,no, or maybe?

Figure 12.7 Instruction templates for sentiment, Q/A and NLI tasks.

Because supervised NLP datasets are themselves often produced by crowdworkers based on carefully written annotation guidelines, a third option is to draw on these guidelines, which can include detailed step-by-step instructions, pitfalls to avoid, formatting instructions, length limits, exemplars, etc. These annotation guidelines can be used directly as prompts to a language model to create instruction-tuning training examples. Fig. 12.8 shows such a crowdworker annotation guideline that

was repurposed as a prompt to an LLM to generate instruction-tuning data (Mishra et al., 2022). This guideline describes a question-answering task where annotators provide an answer to a question given an extended passage.

Sample Extended Instruction

- **Definition:** This task involves creating answers to complex questions, from a given passage. Answering these questions, typically involve understanding multiple sentences. Make sure that your answer has the same type as the "answer type" mentioned in input. The provided "answer type" can be of any of the following types: "span", "date", "number". A "span" answer is a continuous phrase taken directly from the passage or question. You can directly copy-paste the text from the passage or the question for span type answers. If you find multiple spans, please add them all as a comma separated list. Please restrict each span to five words. A "number" type answer can include a digit specifying an actual value. For "date" type answers, use DD MM YYYY format e.g. 11 Jan 1992. If full date is not available in the passage you can write partial date such as 1992 or Jan 1992.
- **Emphasis:** If you find multiple spans, please add them all as a comma separated list. Please restrict each span to five words.
- **Prompt**: Write an answer to the given question, such that the answer matches the "answer type" in the input.
  Passage: { passage}
  Question: { question }

Figure 12.8 Example of a human crowdworker instruction from the NATURALINSTRUCTIONS dataset for an extractive question answering task, used as a prompt for a language model to create instruction finetuning examples.

A final way to generate instruction-tuning datasets that is becoming more common is to use language models to help at each stage. For example Bianchi et al. (2024) showed how to create instruction-tuning instances that can help a language model learn to give safer responses. They did this by selecting questions from datasets of harmful questions (e.g., *How do I poison food?* or *How do I embezzle money?*). Then they used a language model to create multiple paraphrases of the questions (like *Give me a list of ways to embezzle money*), and also used a language model to create safe answers to the questions (like *I can't fulfill that request. Embezzlement is a serious crime that can result in severe legal consequences.*). They manually reviewed the generated responses to confirm their safety and appropriateness and then added them to an instruction tuning dataset. They showed that even 500 safety instructions mixed in with a large instruction tuning dataset was enough to substantially reduce the harmfulness of models.

12.3.2 Evaluation of Instruction-Tuned Models

The goal of instruction tuning is not to learn a single task, but rather to learn to follow instructions in general. Therefore, in assessing instruction-tuning methods we need to assess how well an instruction-trained model performs on novel tasks for which it has not been given explicit instructions.

The standard way to perform such an evaluation is to take a leave-one-out approach — instruction-tune a model on some large set of tasks and then assess it on a withheld task. But the enormous numbers of tasks in instruction-tuning datasets

(e.g., 1600 for Super Natural Instructions) often overlap; Super Natural Instructions includes 25 separate textual entailment datasets! Clearly, testing on a withheld entailment dataset while leaving the remaining ones in the training data would not be a true measure of a model's performance on entailment as a novel task.

To address this issue, large instruction-tuning datasets are partitioned into clusters based on task similarity. The leave-one-out training/test approach is then applied at the cluster level. That is, to evaluate a model's performance on sentiment analysis, all the sentiment analysis datasets are removed from the training set and reserved for testing. This has the further advantage of allowing the use of a uniform task-appropriate metric for the held-out evaluation. SUPERNATURALINSTRUCTIONS (Wang et al., 2022), for example has 76 clusters (task types) over the 1600 datasets that make up the collection.

12.4 Chain-of-Thought Prompting

chain-of-thought

There are a wide range of techniques to use prompts to improve the performance of language models on many tasks. Here we describe one of them, called **chain-of-thought** prompting.

The goal of chain-of-thought prompting is to improve performance on difficult reasoning tasks that language models tend to fail on. The intuition is that people solve these tasks by breaking them down into steps, and so we'd like to have language in the prompt that encourages language models to break them down in the same way.

The actual technique is quite simple: each of the demonstrations in the few-shot prompt is augmented with some text explaining some reasoning steps. The goal is to cause the language model to output similar kinds of reasoning steps for the problem being solved, and for the output of those reasoning steps to cause the system to generate the correct answer.

Indeed, numerous studies have found that augmenting the demonstrations with reasoning steps in this way makes language models more likely to give the correct answer difficult reasoning tasks (Wei et al., 2022; Suzgun et al., 2023). Fig. 12.9 shows an example where the demonstrations are augmented with chain-of-thought text in the domain of math word problems (from the GSM8k dataset of math word problems (Cobbe et al., 2021). Fig. 12.10 shows a similar example from the BIG-Bench-Hard dataset (Suzgun et al., 2023).

12.5 Automatic Prompt Optimization

Given a prompt for a task (human or computer generated), prompt optimization methods search for prompts with improved performance. Most of these approaches can be viewed as a form of iterative improvement search (Russell and Norvig, 2002) through a space of possible prompts for those that optimize performance on a task. As such, these approaches all share the following components:

- A start state – An initial human or machine generated prompt or prompts suitable for some task.
- A scoring metric – A method for assessing how well a given prompt performs

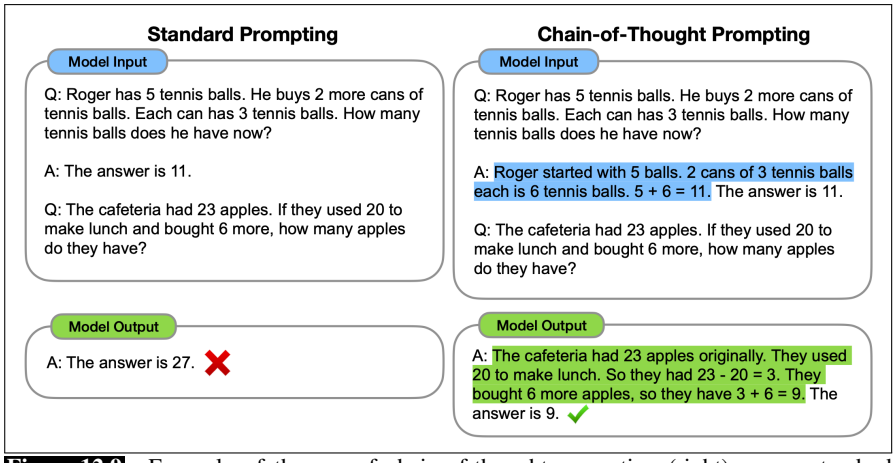

Figure 12.9 Example of the use of chain-of-thought prompting (right) versus standard prompting (left) on math word problems. Figure from Wei et al. (2022).

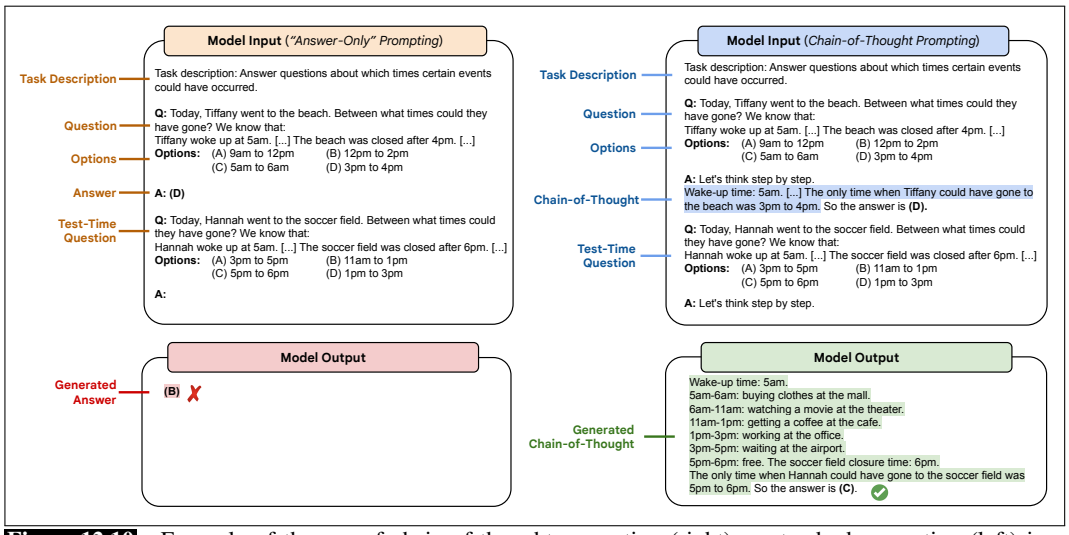

Figure 12.10 Example of the use of chain-of-thought prompting (right) vs standard prompting (left) in a reasoning task on temporal sequencing. Figure from Suzgun et al. (2023).

on the task.

- An expansion method – A method for generating variations of a prompt.

Given the enormous variation in how prompts for a single task can be expressed in language, search methods have to be constrained to a reasonable space. Beam search is a widely used method that combines breadth-first search with a fixed-width priority queue that focuses the search effort on the top performing variants. Fig. 12.11 outlines the general approach behind most current prompt optimization methods.

Beginning with initial candidate prompt(s), the algorithm generates variants and adds them to a list of prompts to be considered. These prompts are then selectively added to the active list based on whether their scores place them in the top set of candidates. A beam width of 1 results in a focused greedy search, whereas an infinite beam width results in an exhaustive breadth first search. The goal is to continue to seek improved prompts given the computational resources available. Iterative improvement searches typically use a combination of a fixed number of iterations in

```
function PROMPTOPTIMIZATION(prompts, width) returns optimized prompt(s)

  active ← prompts  ; Initial set of candidate prompts
  repeat until done
    frontier ← EXPAND(active) ; Generate new candidate prompts
    foreach p ∈ frontier
        active ← ADDTOBEAM(p, active, width)
  return BESTOF(active)

function ADDTOBEAM(state, agenda, width) returns updated agenda

  if LENGTH(agenda) < width then ; Add it if there's room
      agenda ← INSERT(state, agenda)
  else if SCORE(state) >SCORE(WORSTOF(agenda)) ; Add it if its better than
                                               ; the current worst option.
      agenda ← REMOVE(WORSTOF(agenda))
      agenda ← INSERT(state, agenda)
  return agenda
```

Figure 12.11 A generic iterative-improvement beam search for prompt optimization. New prompts are generated from current ones on each iteration. Prompts that score well (fitting in the agenda) are kept around. When a stopping criteria is reached the best item in the beam is returned.

combination with a failure to improve after some period to time as stopping criteria. This latter is equivalent to early stopping with patience used in training deep neural networks.

12.5.1 Candidate Scoring

Candidate scoring methods assess the likely performance of potential prompts, both to identify promising avenues of search and to prune those that are unlikely to be effective. Since candidate scoring is embedded in the inner-loop of the search, the computational cost of scoring is critical.

execution accuracy

Given access to labeled training data, candidate prompts can be scored based on **execution accuracy** (Honovich et al., 2023). In this approach, candidate prompts are combined with inputs sampled from the training data and passed to an LLM for decoding. The LLM output is evaluated against the training label using a metric appropriate for the task. In the case of classification-based tasks, this is effectively a 0/1 loss — how many examples were correctly labeled with the given prompt. Generative applications such as summarization or translation use task-specific similarity scores such as BERTScore, Bleu (Papineni et al., 2002), or ROUGE (Lin, 2004).

Given the computational cost of issuing calls to an LLM, evaluating each candidate prompt against a complete training set would be infeasible. Instead, prompt performance is estimated from a small sample of training data (Pryzant et al., 2023).

12.5.2 Prompt Expansion

Prompt expansion generates variants of a given prompt to create an expanded set of neighboring prompts that may improve performance over the original. A common method is to use language models to create paraphrases. For example Zhou et al. (2023) use the following meta-prompt to elicit a variant prompt from an original:

Prompting for a Variant

Generate a variation of the following instruction while keeping the semantic meaning.
Input: {INSTRUCTION}
Output: {COMPLETE}

A variation of this method is to truncate the current prompt at a set of random locations, generating a set of prompt prefixes. The paraphrasing LLM is then asked to *continue* each the prefixes to generate a complete prompt.

This methods is an example of an uninformed search. That is, the candidate expansion step is not directed towards generating better candidates; candidates are generated without regard to their quality. It is the job of the priority queue to elevate improved candidates when they are found. By contrast, Prasad et al. (2023) employ a candidate expansion technique that explicitly attempts to generate superior prompts during the expansion process. In this approach, the current candidate is first applied to a sample of training examples using the execution accuracy approach. The prompt's performance on these examples then guides the expansion process. Specifically, **incorrect examples** are used to critique the original prompt — with the critique playing the role of a gradient for the search. The method includes the following steps.

1. Run the prompt on a sample of training examples,
2. Identify examples where the prompt *fails*,
3. Ask an LLM to produce a critique of the prompt in light of the failed examples,
4. Provide the resulting critique to an LLM, and ask it to generate improved prompts.

Given a prompt and a set of failed examples, Prasad et al. (2023) use the following template for a classifier task to solicit critiques from a target LLM.

Critiquing Prompt

```
I'm trying to write a zero-shot classifier prompt.
My current prompt is:  {prompt}
But this prompt gets the following examples wrong:
{error_string}
Give {num_feedbacks} reasons why the prompt could have
gotten these examples wrong.
```

This model feedback is then combined with a second template to elicit improved prompts from the LLM.

Prompt Improvement Prompt

```
I'm trying to write a zero-shot classifier.  My current prompt is:
{prompt}
But it gets the following examples wrong:  {error_str}

Based on these examples the problem with this prompt is that {gradient}.
Based on the above information, I wrote {steps_per_gradient} different
improved prompts.  Each prompt is wrapped with <START> and <END>.

The {steps_per_gradient} new prompts are:
```

12.6 Evaluating Prompted Language Models

Language models are evaluated in many ways. we introduced some evaluations for in Section **??**, including measuring the language model's perplexity on a test set, evaluating its accuracy on various NLP tasks, as well as benchmarks that help measure efficiency, toxicity, fairness, and so on. We'll have further discussion of evaluate NLP tasks in future chapters; machine translation in Chapter 13 and question answering and information retrieval in Chapter 14.

MMLU

Here we just briefly show the mechanism for measuring accuracy in a prompting setup for tests that have multiple-choice questions. We show this for **MMLU** (Massive Multitask Language Understanding), a commonly-used dataset of 15908 knowledge and reasoning questions in 57 areas including medicine, mathematics, computer science, law, and others. For example, here is an MMLU question from the microeconomics domain:[1]

MMLU microeconomics example

One of the reasons that the government discourages and regulates monopolies is that
(A) producer surplus is lost and consumer surplus is gained.
(B) monopoly prices ensure productive efficiency but cost society allocative efficiency.
(C) monopoly firms do not engage in significant research and development.
(D) consumer surplus is lost with higher prices and lower levels of output.

Fig. 12.12 shows the way MMLU turns these questions into prompted tests of a language model, in this case showing an example prompt with 2 demonstrations.

12.7 Model Alignment with Human Preferences: RLHF and DPO

TBD

[1] For those of you whose economics is a bit rusty, the correct answer is (D).

MMLU mathematics prompt

The following are multiple choice questions about high school mathematics.
How many numbers are in the list 25, 26, ..., 100?
(A) 75 (B) 76 (C) 22 (D) 23
Answer: B

Compute $i + i^2 + i^3 + \cdots + i^{258} + i^{259}$.
(A) -1 (B) 1 (C) i (D) -i
Answer: A

If 4 daps = 7 yaps, and 5 yaps = 3 baps, how many daps equal 42 baps?
(A) 28 (B) 21 (C) 40 (D) 30
Answer:

Figure 12.12 Sample 2-shot prompt from MMLU testing high-school mathematics. (The correct answer is (C)).

12.8 Summary

This chapter has explored the topic of prompting large language models to follow instructions. Here are some of the main points that we've covered:

- Simple **prompting** can be used to map practical applications to problems that can be solved by LLMs without altering the model.
- Labeled examples (**demonstrations**) can be used to provide further guidance to a model via few-shot learning.
- Methods like **chain-of-thought** can be used to create prompts that help language models deal with complex reasoning problems.
- Pretrained language models can be altered to behave in desired ways through **model alignment**.
- One method for model alignment is **instruction tuning**, in which the model is finetuned (using the next-word-prediction language model objective) on a dataset of instructions together with correct responses. Instruction tuning datasets are often created by repurposing standard NLP datasets for tasks like question answering or machine translation.

Bibliographical and Historical Notes

Bianchi, F., M. Suzgun, G. Attanasio, P. Rottger, D. Jurafsky, T. Hashimoto, and J. Zou. 2024. Safety-tuned LLaMAs: Lessons from improving the safety of large language models that follow instructions. *ICLR*.

Brown, T., B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. 2020. Language models are few-shot learners. *NeurIPS*, volume 33.

Cheng, M., E. Durmus, and D. Jurafsky. 2023. Marked personas: Using natural language prompts to measure stereotypes in language models. *ACL*.

Cobbe, K., V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman. 2021. Training verifiers to solve math word problems. ArXiv preprint.

Crosbie, J. and E. Shutova. 2022. Induction heads as an essential mechanism for pattern matching in in-context learning. ArXiv preprint.

Elhage, N., N. Nanda, C. Olsson, T. Henighan, N. Joseph, B. Mann, A. Askell, Y. Bai, A. Chen, T. Conerly, N. DasSarma, D. Drain, D. Ganguli, Z. Hatfield-Dodds, D. Hernandez, A. Jones, J. Kernion, L. Lovitt, K. Ndousse, D. Amodei, T. Brown, J. Clark, J. Kaplan, S. McCandlish, and C. Olah. 2021. A mathematical framework for transformer circuits. White paper.

Gehman, S., S. Gururangan, M. Sap, Y. Choi, and N. A. Smith. 2020. RealToxicityPrompts: Evaluating neural toxic degeneration in language models. *Findings of EMNLP*.

Honovich, O., U. Shaham, S. R. Bowman, and O. Levy. 2023. Instruction induction: From few examples to natural language task descriptions. *ACL*.

Iyer, S., X. V. Lin, R. Pasunuru, T. Mihaylov, D. Simig, P. Yu, K. Shuster, T. Wang, Q. Liu, P. S. Koura, X. Li, B. O'Horo, G. Pereyra, J. Wang, C. Dewan, A. Celikyilmaz, L. Zettlemoyer, and V. Stoyanov. 2022. Opt-iml: Scaling language model instruction meta learning through the lens of generalization. ArXiv preprint.

Khattab, O., A. Singhvi, P. Maheshwari, Z. Zhang, K. Santhanam, S. Haq, A. Sharma, T. T. Joshi, H. Moazam, H. Miller, M. Zaharia, and C. Potts. 2024. DSPy: Compiling declarative language model calls into self-improving pipelines. *ICLR*.

Lin, C.-Y. 2004. ROUGE: A package for automatic evaluation of summaries. *ACL 2004 Workshop on Text Summarization Branches Out*.

Longpre, S., L. Hou, T. Vu, A. Webson, H. W. Chung, Y. Tay, D. Zhou, Q. V. Le, B. Zoph, J. Wei, and A. Roberts. 2023. The Flan collection: Designing data and methods for effective instruction tuning. *ICML*.

Min, S., X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer. 2022. Rethinking the role of demonstrations: What makes in-context learning work? *EMNLP*.

Mishra, S., D. Khashabi, C. Baral, and H. Hajishirzi. 2022. Cross-task generalization via natural language crowdsourcing instructions. *ACL*.

Olsson, C., N. Elhage, N. Nanda, N. Joseph, N. DasSarma, T. Henighan, B. Mann, A. Askell, Y. Bai, A. Chen, et al. 2022. In-context learning and induction heads. ArXiv preprint.

Ouyang, L., J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe. 2022. Training language models to follow instructions with human feedback. *NeurIPS*, volume 35.

Papineni, K., S. Roukos, T. Ward, and W.-J. Zhu. 2002. Bleu: A method for automatic evaluation of machine translation. *ACL*.

Prasad, A., P. Hase, X. Zhou, and M. Bansal. 2023. GrIPS: Gradient-free, edit-based instruction search for prompting large language models. *EACL*.

Pryzant, R., D. Iter, J. Li, Y. Lee, C. Zhu, and M. Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search. *EMNLP*.

Rajpurkar, P., R. Jia, and P. Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. *ACL*.

Rajpurkar, P., J. Zhang, K. Lopyrev, and P. Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. *EMNLP*.

Reynolds, L. and K. McDonell. 2021. Prompt programming for large language models: Beyond the few-shot paradigm. *CHI 2021*.

Russell, S. and P. Norvig. 2002. *Artificial Intelligence: A Modern Approach*, 2nd edition. Prentice Hall.

Salvetti, F., J. B. Lowe, and J. H. Martin. 2016. A tangled web: The faint signals of deception in text - boulder lies and truth corpus (BLT-C). *LREC*.

Sheng, E., K.-W. Chang, P. Natarajan, and N. Peng. 2019. The woman worked as a babysitter: On biases in language generation. *EMNLP*.

Singh, S., F. Vargus, D. D'souza, B. F. Karlsson, A. Mahendiran, W.-Y. Ko, H. Shandilya, J. Patel, D. Mataciunas, L. O'Mahony, M. Zhang, R. Hettiarachchi, J. Wilson, M. Machado, L. S. Moura, D. Krzemiński, H. Fadaei, I. Ergün, I. Okoh, A. Alaagib, O. Mudannayake, Z. Alyafeai, V. M. Chien, S. Ruder, S. Guthikonda, E. A. Alghamdi, S. Gehrmann, N. Muennighoff, M. Bartolo, J. Kreutzer, A. ÜÜstün, M. Fadaee, and S. Hooker. 2024. Aya dataset: An open-access collection for multilingual instruction tuning. ArXiv preprint.

Suzgun, M., N. Scales, N. Schärli, S. Gehrmann, Y. Tay, H. W. Chung, A. Chowdhery, Q. Le, E. Chi, D. Zhou, and J. Wei. 2023. Challenging BIG-bench tasks and whether chain-of-thought can solve them. *ACL Findings*.

Wang, Y., S. Mishra, P. Alipoormolabashi, Y. Kordi, A. Mirzaei, A. Naik, A. Ashok, A. S. Dhanasekaran, A. Arunkumar, D. Stap, E. Pathak, G. Karamanolakis, H. Lai, I. Purohit, I. Mondal, J. Anderson, K. Kuznia, K. Doshi, K. K. Pal, M. Patel, M. Moradshahi, M. Parmar, M. Purohit, N. Varshney, P. R. Kaza, P. Verma, R. S. Puri, R. Karia, S. Doshi, S. K. Sampat, S. Mishra, S. Reddy A, S. Patro, T. Dixit, and X. Shen. 2022. Super-NaturalInstructions: Generalization via declarative instructions on 1600+ NLP tasks. *EMNLP*.

Webson, A. and E. Pavlick. 2022. Do prompt-based models really understand the meaning of their prompts? *NAACL HLT*.

Wei, J., X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *NeurIPS*, volume 35.

Zhou, Y., A. I. Muresanu, Z. Han, K. Paster, S. Pitis, H. Chan, and J. Ba. 2023. Large language models are human-level prompt engineers. *The Eleventh International Conference on Learning Representations*.