CHAPTER

10 Post-training: Instruction Tuning, Alignment, and Test-Time Compute

"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

Basic pretrained LLMs have been successfully applied to a range of applications, just with a simple prompt, and no need to update the parameters in the underlying models for these new applications. 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. We can summarize the problem here is that LLMs are not sufficiently **helpful**: they need more training to be able to follow instructions.

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. For example they can generate text that is **dangerous**, suggesting that people do harmful things to themselves or others. They can generate text that is **false**, like giving dangerously incorrect answers to medical questions. And they can verbally attack their users, generating text that is **toxic**. 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

To address these two problems, language models 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** (sometimes called **SFT** for supervised finetuning), models are finetuned on a corpus of instructions and questions with their corresponding responses. We'll describe this in the next section.

In the second technique, **preference alignment**, (sometimes called RLHF or DPO after two specific instantiations, Reinforcement Learning from Human Feedback and Direct Preference Optimization), 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. We'll describe preference alignment in Section 10.2.

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 preference alignment. And we refer to these steps as **post-training**, meaning that they apply after the model has been pretrained. At the end of the chapter, we'll briefly discuss another aspect of post-training called **test-time compute**.

10.1 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 7 and Chapter 9? Fig. 10.1 sketches the differences. In the first example, introduced in Chapter 7 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.

In the second example, also from Chapter 7, **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 9, we adapt to a particular task by adding a new specialized classification head and updating its features via its own loss func-

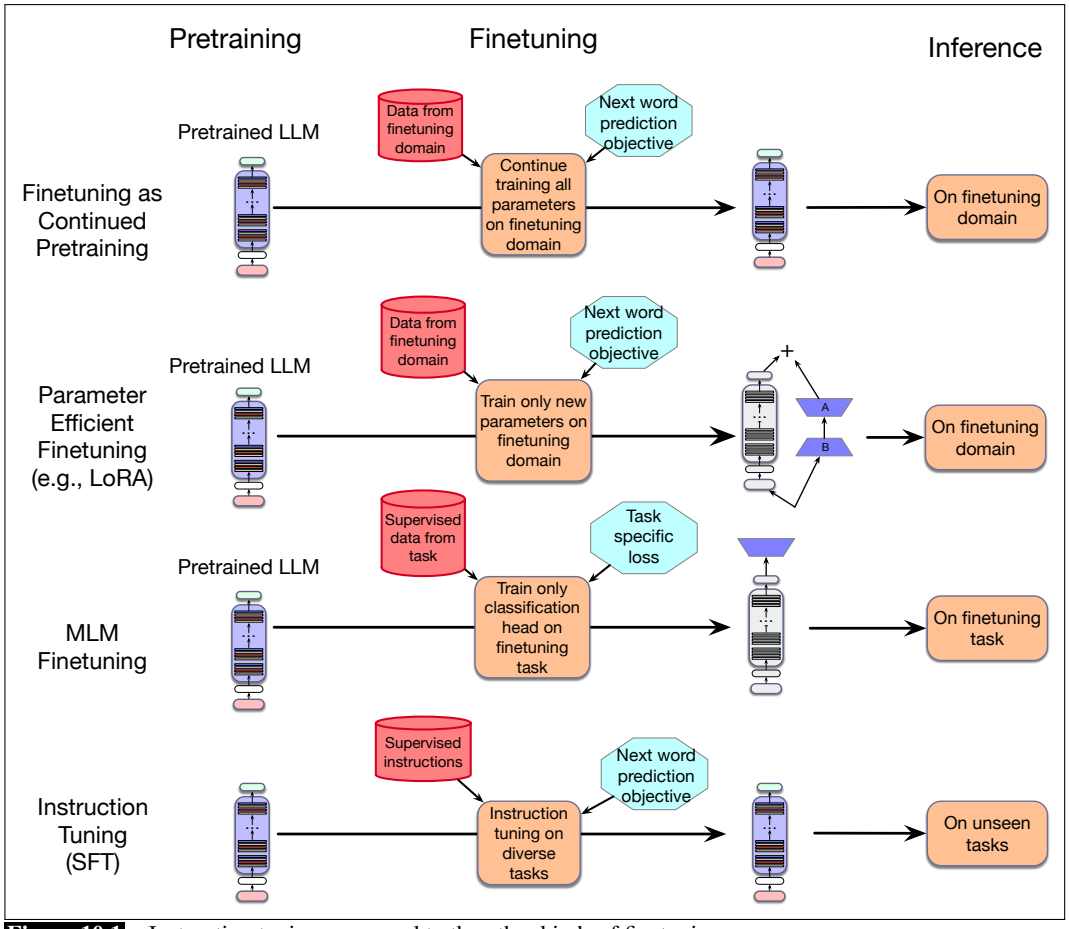


Figure 10.1 Instruction tuning compared to the other kinds of finetuning.

tion (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.

10.1.1 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 descriptions 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. 10.2) 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 میید چند لانه کبوتر وجود دارد؟	کبوترخانه میید 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 10.2 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. 10.3 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. 10.4) 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.

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

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 Beyoncé start becoming popular?
	`answers`	{ `text`: ['in the late 1990s'], `answer_start`: 269 }

Figure 10.3 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 10.4 Instruction templates for sentiment, Q/A and NLI tasks.

training examples. Fig. 10.5 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.

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 embez-*

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 10.5 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.

zle 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.

10.1.2 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.

10.2 Learning from Preferences

Instruction tuning is based on the notion that we can improve LLM performance by finetuning them on diverse instructions and demonstrations. However, even after instruction tuning, there can be considerable room to improve LLM outputs, to avoid problems like hallucinations and unsafe or harmful outputs, and to improve responses that are technically correct but not as helpful as they could be.

preference-based learning

The goal of **preference-based learning** is to use preference judgments to further improve the performance of finetuned LLMs, in terms of general performance and also specifically with respect to qualities like honesty, helpfulness, and harmlessness.

Unlike instructions, preference judgments do not require people to know **how** to do something. We simply have to have an opinion about the end result. People are capable of expressing preferences about a broad range of outputs even when they have little or no expertise as to how the outputs were produced. Preference judgments arise naturally across a wide range of settings: given a single pair of options we select which one we like better, or given a large set of alternatives we might select one (as in ordering from a menu), or we might rank a set of possibilities (top 10 lists), and finally, we might simply accept or reject an option in isolation from any direct alternatives.

10.2.1 LLM Preference Data

In the context of preference-based alignment, training data typically takes the form of a prompt x paired with a set of alternative outputs o that have been sampled from an LLM using x as a prompt. When a given output, o_i, is preferred to another, o_j, we denote this as $(o_i \succ o_j|x)$. Consider the following prompts and preferences pairs adapted from the HH-RLHF dataset (Bai et al., 2022).

Prompt: I've heard garlic is a great natural antibiotic. Does it help with colds?
Chosen: It can be helpful against colds, but may make you stink.
Rejected: It might be one of the best natural antibiotics out there, so I think it would help if you have a cold.

Prompt: What is malaria?
Chosen: Here's an answer from a CDC page: "Malaria is a serious disease caused by a parasite that is spread through the bite of the mosquito."
Rejected: I don't know what malaria is.

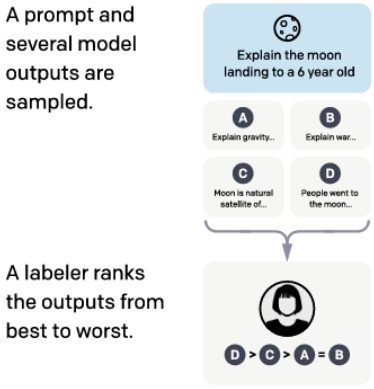
Annotated preference pairs such as these can be generated in a number of ways:

- Direct annotation of pairs of sampled outputs by trained annotators.
- Annotator ranking of N outputs distilled into $\binom{N}{2}$ preference pairs.

- Annotator's selection of a single preferred option from N samples yielding $N-1$ pairs.

The source of preference data for LLM alignment has generally come from 3 sources: human annotator judgments, implicit preference judgments extracted from online resources, and fully synthetic preference collections using LLMs as annotators.

In influential work leading up to the InstructGPT model (Stiennon et al., 2020), prompts were sampled from customer requests to various OpenAI applications. Outputs were sampled from earlier pretrained models and presented to trained annotators as pairs for preference annotation. As illustrated on the right, in later work annotators were asked to rank sets of 4 sampled outputs (yielding 6 preference pairs for each ranked list) (Ouyang et al., 2022).



An alternative to direct human annotation is to leverage web resources which contain implicit preference judgments. Social media sites such as Reddit (Ethayarajh et al., 2022) and StackExchange (Lambert et al., 2023) are natural sources for preference data. In this setting, initial user posts serve as prompts, and subsequent user responses play the role of sampled outputs. Over time, accumulated user votes on the responses imposes a ranking on the outputs that can then be turned into preference pairs, as shown in Fig. 10.6.

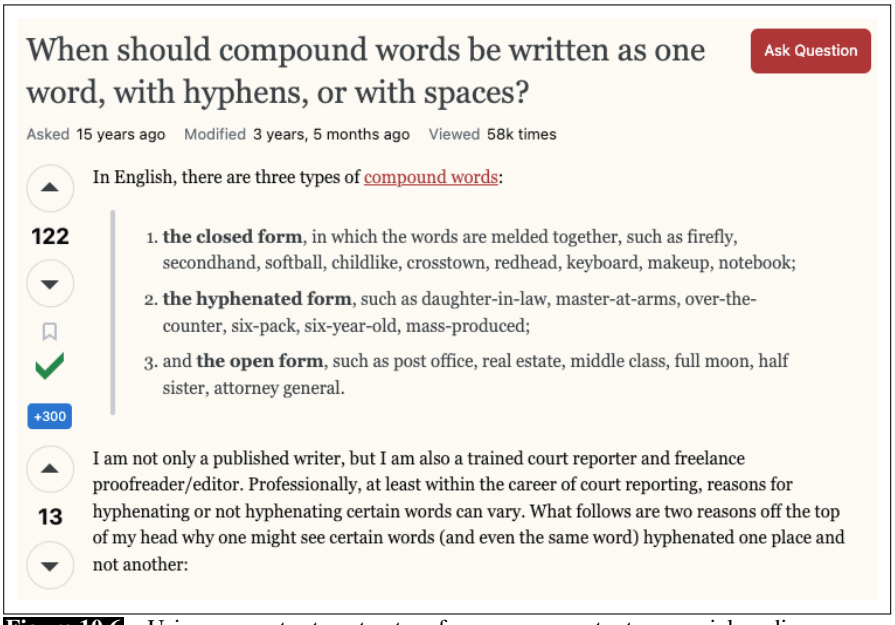


Figure 10.6 Using user votes to extract preferences over outputs on social media.

Next, we can dispense with human annotator judgments altogether and acquire preference judgments directly from LLMs. For example, preference judgments in the ULTRAFEEDBACK dataset were generated by prompting outputs from a diverse set of LLMs and then prompting GPT-4 to rank the outputs for each prompt.

Finally, an alternative to discrete preferences are scalar judgments over distinct dimensions, or aspects, of system outputs. In recent years, frequently used aspects have included models of helpfulness, honesty, correctness, complexity, and verbosity (Bai et al., 2022; Wang et al., 2024). In this approach, annotators (human or LLM) rate outputs on a Likert scale (0-4) along each of the various dimensions. Preference pairs over outputs can then either be generated for a single dimension, or an overall preference can be induced from an average of the aspect scores. Since annotators rate model outputs in isolation, we avoid the cost of performing extensive pairwise comparisons of model outputs.

10.2.2 Modeling Preferences

Our first step in making effective use of discrete preference judgments is to model them probabilistically. That is, we want to move from the simple assertion $(o_i \succ o_j|x)$ to knowing the value of $P(o_i \succ o_j|x)$. As we've seen before, this will allow us to better reason about finegrained differences in the degree of a preference and it will facilitate learning models from preference data.

Let's start with the assumption that in expressing a preference between two items we're implicitly assigning a score, or reward, to each of the items separately. Further, let's assume these scores are scalar values, $z \in \mathbb{R}$. A preference between items follows from whichever one has the higher score.

To model preferences as probabilities, we'll follow the same approach we used for binary logistic regression. Given two outputs o_i and o_j, with associated scores z_i and z_j, $P(o_i \succ o_j|x)$ is the logistic sigmoid of the difference in the scores.

$$\begin{aligned} P(o_i \succ o_j|x) &= \frac{1}{1+e^{-(z_i-z_j)}} \\ &= \sigma(z_i - z_j) \end{aligned}$$

Bradley-Terry Model

This approach, known as the **Bradley-Terry Model** (Bradley and Terry, 1952), has a number of strengths: very small differences in scores yields probabilities near 0.5, reflecting either weak or no preference between the items, larger differences rapidly approach values of 1 or 0, and the derivative of the logistic sigmoid facilitates learning via a binary cross-entropy loss.

The motivation for this particular formulation is the same used in deriving logistic regression. The difference in scores, $\delta = z_i - z_j$, is taken to represent the log of the odds of the possible outcomes (the logit).

$$\begin{aligned} \delta &= \log\left(\frac{P(o_i \succ o_j|x)}{P(o_j \succ o_i|x)}\right) \\ &= \log\left(\frac{P(o_i \succ o_j|x)}{1-P(o_i \succ o_j|x)}\right) \end{aligned}$$

Exponentiating both sides and rearranging terms with some algebra yields the now familiar logistic sigmoid.

$$\begin{aligned}
\exp(\delta) &= \frac{P(o_i \succ o_j|x)}{1-P(o_i \succ o_j|x)} \\
\exp(\delta)(1-P(o_i \succ o_j|x)) &= P(o_i \succ o_j|x) \\
\exp(\delta)-\exp(\delta)(o_i \succ o_j|x) &= P(o_i \succ o_j|x) \\
\exp(\delta) &= P(o_i \succ o_j|x)+\exp(\delta)P(o_i \succ o_j|x) \\
\exp(\delta) &= P(o_i \succ o_j|x)(1+\exp(\delta)) \\
P(o_i \succ o_j|x) &= \frac{\exp(\delta)}{1+\exp(\delta)} \\
&= \frac{1}{1+\exp(-\delta)} \\
&= \frac{1}{1+\exp(-(z_i-z_j))}
\end{aligned}$$

Bringing us right back to our original formulation.

$$P(o_i \succ o_j|x) = \sigma(z_i-z_j)$$

10.2.3 Learning to Score Preferences

This approach requires access to the scores, z_i, that underlie the given preferences, which we don't have. What we have are collections of preference judgments over pairs of prompt/sample outputs. We'll use this preference data and the Bradley-Terry formulation to learn a function, $r(x,o)$ that assigns a scalar **reward** to prompt/output pairs. That is, $r(x,o)$ calculates the z score from above.

reward

$$\begin{aligned}
P(o_i \succ o_j|x) &= \sigma(z_i-z_j) && (10.1)\\
&= \sigma(r(o_i,x)-r(o_j,x)) && (10.2)
\end{aligned}$$

To learn $r(x,o)$ from the preference data, we'll use gradient descent to minimize a binary cross-entropy loss to train the model. Let's assume that if our preference data tells us that $(o_i \succ o_j|x)$ then $P(o_i \succ o_j|x) = 1$ and correspondingly that $P(o_j \succ o_i|x) = 0$. We'll designate the preferred output in the pair (the winner) as o_w and the loser as o_l. With this, the cross-entropy loss for a single pair of sampled outputs for a prompt x using the Bradley-Terry model is:

$$\begin{aligned}
L_{CE}(x,o_w,o_l) &= -\log P(o_w \succ o_l|x) \\
&= -\log \sigma(r(x,o_w)-r(x,o_l))
\end{aligned}$$

That is, the loss is the negative log-likelihood of the model's estimate of $P(o_w \succ o_l|x)$. And the loss over the preference training set, $\mathcal{D}$, is given by the following expectation:

$$L_{CE} = -\mathbb{E}_{(x,o_w,o_l)\sim\mathcal{D}}[\log \sigma(r(x,o_w)-r(x,o_l))] \quad (10.3)$$

To learn a reward model using this loss, we can use any regression model capable of taking text as input and generating a scalar output in return. As shown in Fig. 10.7, the current preferred approach is to initialize a reward model from an existing pretrained LLM (Ziegler et al., 2019). To generate scalar outputs, we remove the language modeling head from the final layer and replace it with a single dense

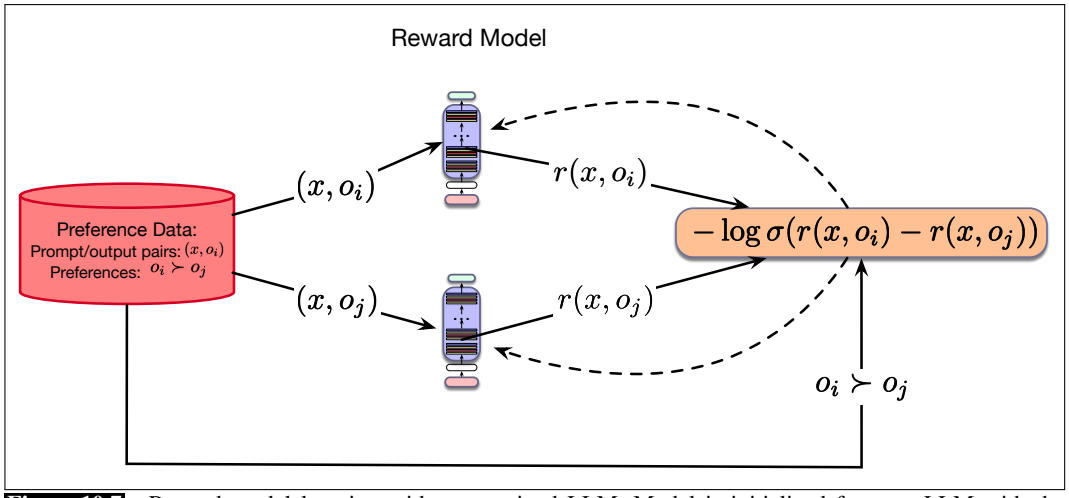


Figure 10.7 Reward model learning with a pretrained LLM. Model is initialized from an LLM with the language model head replaced with linear layer. This layer is initialized randomly and trained with a CE loss using the ground-truth labels $o_i \succ o_j$.

linear layer. We then use gradient descent with the loss from 10.3 to learn to score model outputs using the preference training data.

Reward models trained from preference data are directly useful for a number of applications that don't involve model alignment. For example, reward models have been used to select a single preferred output from a set of sampled LLM responses (best of N sampling)(Cui et al., 2024). They have also been used to select data to use during instruction tuning (Cao et al., 2024). Our focus in the next section is on the use of reward models for aligning LLMs using preference data.

10.3 LLM Alignment via Preference-Based Learning

Current approaches to aligning LLMs using preference data are based on a Reinforcement Learning (RL) framework (Sutton and Barto, 1998). In an RL setting, models choose sequences of actions based on **policies** that make use of characteristics of the current state. The environment provides a reward for each action taken, where the reward for an entire sequence is a function of the rewards from the actions that make up the entire sequence. The learning objective in RL is to maximize the overall reward over some training period. In applying RL to optimizing LLMs, we'll use the following framework:

- **Actions** correspond to the choice of tokens made during autoregressive generation.
- **States** correspond to the context of the current decoding step. That is, the history of tokens generated up to that point.
- **Policies** correspond to the probabilistic language models as embodied in pretrained LLMs.
- **Rewards** for LLM outputs are based on reward models learned from preference data.

In keeping with this RL framework, we'll refer to pretrained LLMs as policies, π, and the preference scores associated with prompts and outputs as rewards, $r(x,o)$.

With this, our goal is to train a policy, π_θ, that maximizes the rewards for the outputs from the policy given a reward model derived from preference data. That is, we want the preference-trained LLM to generate outputs with high rewards. We can express this as an optimization problem as follows:

$$\pi^* = \underset{\pi_\theta}{\text{argmax}}\, \mathbb{E}_{x\sim\mathcal{D}, o\sim\pi_\theta(o|x)}[r(x,o)] \tag{10.4}$$

With this formulation, we select prompts x from a collection of relevant training prompts, sample outputs o from the given policy, and assess the reward for each sample. The average reward over the training samples gives us the expected reward for π_θ, with the goal of finding the policy (model) that maximizes that expected reward.

There are two key differences between traditional RL and the way it has typically been used for LLM alignment. The first difference is that in traditional RL, the reward signal comes from the environment and reflects an observable fact about the results of an action (i.e., you win a game or you don't). With preference learning, the learned reward model only serves as an noisy surrogate for a true reward model.

The second difference lies in the starting point for learning. Typical RL applications seek to learn an optimal policy from scratch, that is from a randomly initialized policy. Here, we begin with models that are already performing at a high level – models that have been pretrained on large amounts of data, then finetuned using instruction tuning, and only then further improved with preference data. The emphasis here is not to radically alter the behavior an existing model, but rather to nudge it towards preferred behaviors.

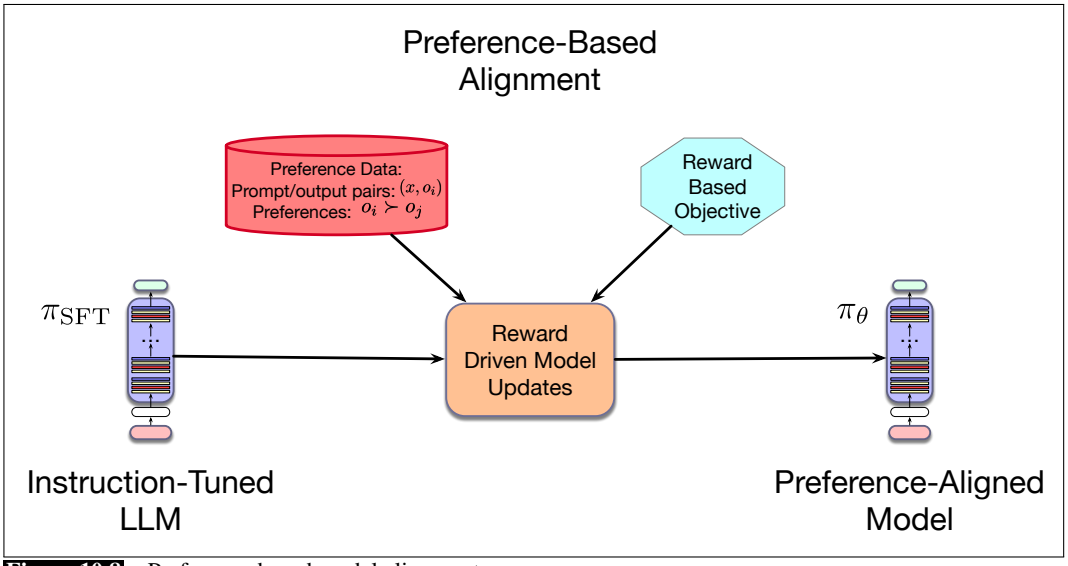


Figure 10.8 Preference-based model alignment.

Given this, if we optimize for the rewards as in 10.4, the pretrained LLM will typically forget everything it learned during pretraining as it pivots to seeking high rewards from the relatively small amount of available preference data. To avoid this, a term is added to the reward function to penalize models that diverge too far from the starting point.

$$\pi^* = \underset{\pi_\theta}{\text{argmax}}\, \mathbb{E}_{x\sim\mathcal{D}, o\sim\pi_\theta(o|x)}[r(x,o) - \beta\mathbb{D}_{\text{KL}}[\pi_\theta(o|x)||\pi_{\text{ref}}(o|x)]] \tag{10.5}$$

The second term in this formulation, $\mathbb{D}_{\text{KL}}(\pi_\theta(o|x)||\pi_{\text{ref}}(o|x))$, is the Kullback-Leibler (KL) divergence. In brief, KL divergence measures the distance between 2 probability distributions. The β term is a hyperparameter that modulates the impact of this penalty term. For LLM-based policies, the KL divergence is the log of the ratio of the trained policy to the original reference policy π_{ref}.

$$\pi^* = \underset{\pi_\theta}{\text{argmax}}\, \mathbb{E}_{x\sim\mathcal{D},o\sim\pi_\theta(o|x)}\left[r_\phi(x,o) - \beta \log \frac{\pi_\theta(o|x)}{\pi_{\text{ref}}(o|x)}\right] \tag{10.6}$$

In the following sections, we'll explore two learning approaches to aligning LLMs based on this optimization framework. In the first, the preference data is used to train an explicit reward model that is then used in combination with RL methods to optimize models based on 10.6. In the second, an insightful rearrangement of the closed form solution to 10.6 is used to finetune models directly from existing preference data.

10.3.1 Reinforcement Learning with Preference Feedback (PPO)

TBD

10.3.2 Direct Preference Optimization

Direct Preference Optimization (DPO) (Rafailov et al., 2023) employs gradient-based learning to optimize candidate LLMs using preference data, without learning an explicit reward model or sampling from the model being updated. Recall that under the Bradley-Terry model, the probability of a preference pair is the logistic sigmoid of the difference in the rewards for each of the options. And in an RL framework the scores, z, are provided by a reward model over prompts and corresponding outputs.

$$P(o_i \succ o_j|x) = \sigma(z_i - z_j) \tag{10.7}$$
$$= \sigma(r(x,o_i) - r(x,o_j)) \tag{10.8}$$

DPO begins with the KL-constrained maximization introduced earlier in 10.6, which expresses the optimal policy π^* in terms of the reward model and the reference model π_{ref}. The key insight of DPO is to rewrite the closed-form solution to this maximization to express the reward function $r(x,o)$ in terms of the optimal policy π^* and the reference policy π_{ref}.

$$r(x,o) = \beta \log \frac{\pi_r(o|x)}{\pi_{ref}(o|x)} + \beta \log Z(x) \tag{10.9}$$

Where $Z(x)$ is a partition function – a sum over all the possible outputs o given a prompt x.

$$Z(x) = \sum_y \pi_{\text{ref}}(o|x) \exp\left(\frac{1}{\beta} r(x,o)\right) \tag{10.10}$$

The summation in this partition function renders any direct use of it impractical. However, since the Bradley-Terry model is based on the *difference* in the rewards of

the items, plugging 10.9 into 10.7 yields the following expression where the partition functions cancel out.

$$P(o_i \succ o_j|x) = \sigma(r(x,o_i) - r(x,o_j)) \tag{10.11}$$

$$= \sigma\left(\beta \log \frac{\pi_\theta(o_i|x)}{\pi_{\text{ref}}(o_i|x)} - \beta \log \frac{\pi_\theta(o_j|x)}{\pi_{ref}(o_j|x)}\right) \tag{10.12}$$

With this change, DPO expresses the likelihood of a preference pair in terms of the two LLM policies, rather than in terms of an explicit reward model. Given this, the CE loss (negative log likelihood) for a single instance is:

$$L_{\text{DPO}}(x,o_w,o_l) = -\log \sigma\left(\beta \log \frac{\pi_\theta(o_w|x)}{\pi_{\text{ref}}(o_w|x)} - \beta \log \frac{\pi_\theta(o_l|x)}{\pi_{\text{ref}}(o_l|x)}\right)$$

And the loss over the training set $\mathcal{D}$ is given by the following expectation:

$$L_{\text{DPO}}(\pi_\theta) = -\mathbb{E}_{(x,o_w,o_l)\sim\mathcal{D}}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(o_w|x)}{\pi_{\text{ref}}(o_w|x)} - \beta \log \frac{\pi_\theta(o_l|x)}{\pi_{\text{ref}}(o_l|x)}\right)\right]$$

This loss follows from the derivative of the sigmoid and is directly analogous to the one introduced in Section 10.2.3 for learning a reward model using the Bradley-Terry framework. Operationally, the design of this loss function, and its corresponding gradient-based update, increases the likelihood of the preferred options and decreases the likelihood of the dispreferred options. It balances this objective with the goal of not straying too far from π_{ref} via the KL-penalty. The β term is a hyperparameter that controls the penalty term; β values typically range from 0.1 to 0.01.

As illustrated in Fig. 10.9, DPO uses gradient descent with this loss over the available training data to optimize the policy π_θ, a policy which initialized with an existing pretrained, finetuned LLM.

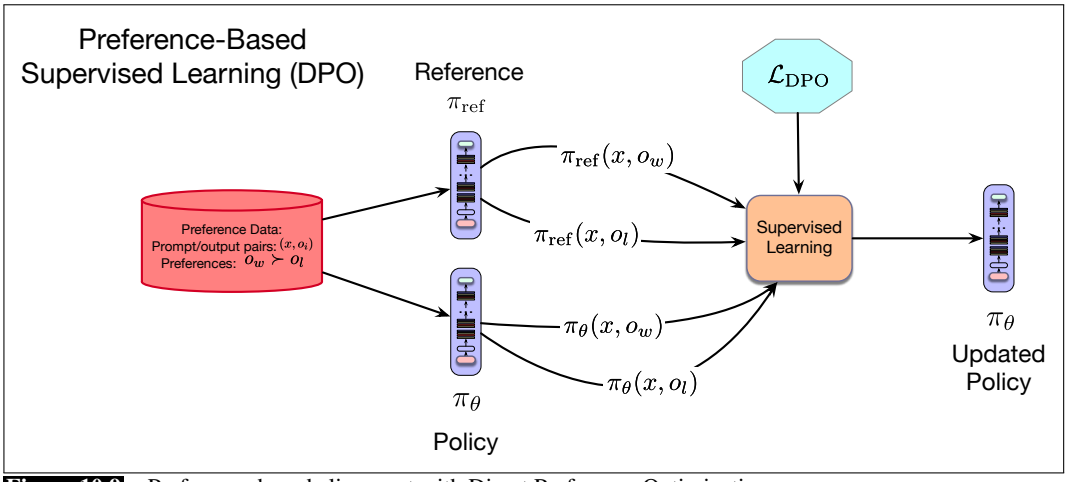


Figure 10.9 Preference-based alignment with Direct Preference Optimization.

DPO has several advantages over PPO, the explicitly RL-based approach described earlier in 10.3.1.

- DPO does not require training an explicit reward model.
- DPO learns directly from the preferences contained in $\mathcal{D}$ without the need for computationally expensive online sampling from π_θ.

- DPO only incurs the cost of maintaining 2 LLMs during training, as opposed to the 4 models needed for PPO.

10.3.3 Evaluation of Preference-Aligned Models

10.3.4 Limitations of Preference-Based Learning

10.4 Test-time Compute

We've now seen 3 levels of training for large language models: **pretraining**, where models learn to predict words, and two kinds of post-training: **instruct tuning**, where they learn to follow instructions, and **preference alignment**, where they learn to prefer prompt continuations that are preferred by humans.

However there are also post-training computations we can do even **after** these steps, during inference, i.e., when the model is generating its output. This class of post-training tasks is called **test-time compute**. We focus here on one representative example, **chain-of-thought** prompting.

test-time compute

10.4.1 Chain-of-Thought Prompting

There is 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.

chain-of-thought

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 to difficult reasoning tasks (Wei et al., 2022; Suzgun et al., 2023b). Fig. 10.10 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. 10.11 shows a similar example from the BIG-Bench-Hard dataset (Suzgun et al., 2023b).

10.5 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.

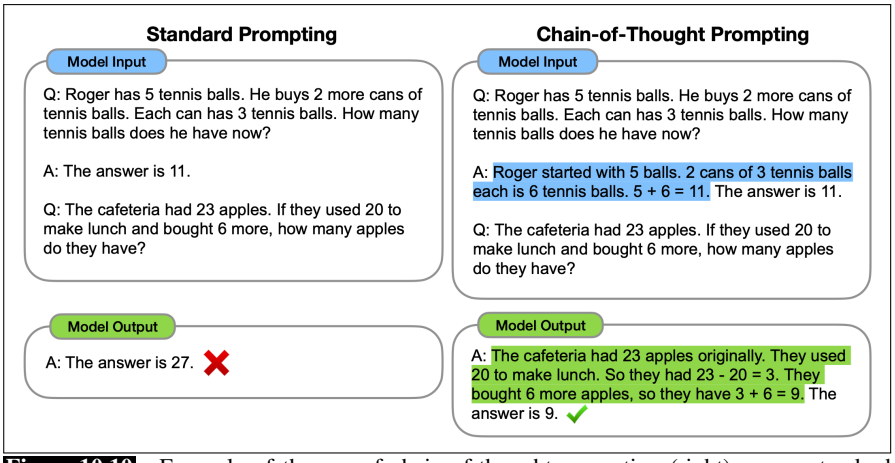


Figure 10.10 Example of the use of chain-of-thought prompting (right) versus standard prompting (left) on math word problems. Figure from Wei et al. (2022).

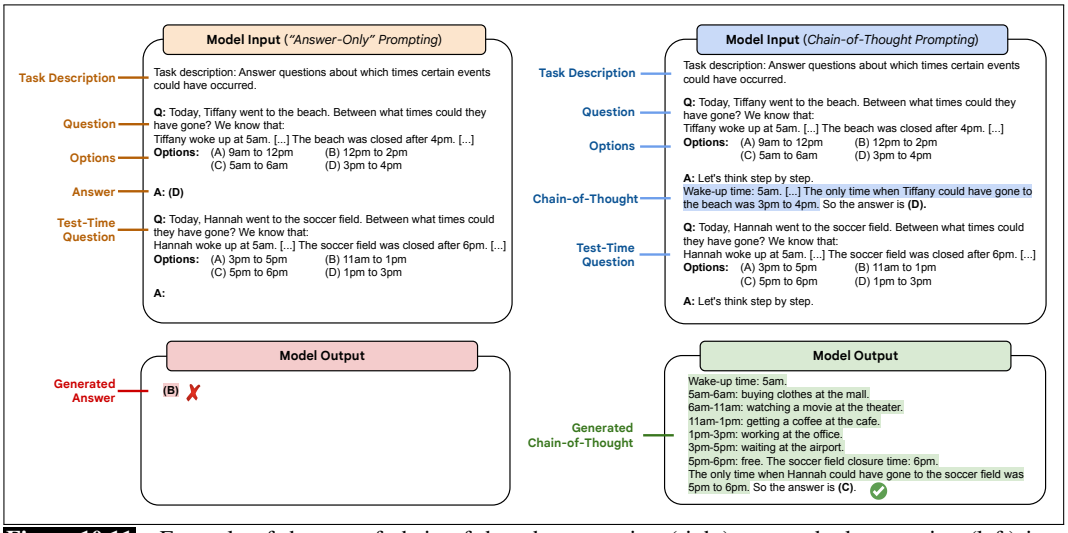


Figure 10.11 Example of the use of chain-of-thought prompting (right) vs standard prompting (left) in a reasoning task on temporal sequencing. Figure from Suzgun et al. (2023b).

- 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.

Historical Notes