

Chapter 3. Evaluation Methodology

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 3rd chapter of the final book. Compared to the last Early Release (before June 26), Chapter 3 has been significantly updated, introducing new concepts and approaches that are necessary for understanding the rest of the book. Chapters 1 and 2 have smaller updates. For this update: Ch 5 has been substantially updated, and broken into two chapters. Chapter 6 is a new chapter.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at jleonard@oreilly.com.

The more AI models there are, the more important it is to evaluate them. The more AI is being used, the more opportunities there are for catastrophic failures. We’ve already seen many such failures in the short time that foundation models have been around. A man committed suicide after being [encouraged by a chatbot](#). Lawyers have submitted [false evidence hallucinated by AI](#). Air Canada was ordered to pay damages when its AI-powered chatbot gave [false information](#) to a passenger. If we don’t have a way to quality control AI outputs, the risk of AI might outweigh its benefits for many applications.

As teams rush to adopt AI, many quickly realize that the biggest hurdle to bringing AI applications to reality is evaluation. For some applications, figuring out evaluation can take up the majority of the development effort. In December 2023, Greg Brockman, an OpenAI co-founder, [tweeted](#) that “evals are surprisingly often all you need.”

Due to the importance and complexity of evaluation, this book has two chapters on it. This chapter covers different evaluation methods used to evaluate open-ended models, how these methods work, and their limitations. The next chapter focuses on how to use these methods to select models and build an evaluation pipeline for your application.

Before diving into evaluation methods, it’s important to acknowledge the challenges of evaluating foundation models. Because evaluation is difficult, many people settle for eyeballing the results. This creates even more risk for their applications and slows down iteration. Instead, we need to invest in systematic evaluation to make the process easier and produce more reliable results.

Since many foundation models have a language model component, this chapter will provide a quick overview of the metrics used to evaluate language models, including cross entropy and perplexity. These metrics are essential for guiding the training and finetuning of language models and are frequently used in many evaluation methods.

Evaluating foundation models is especially challenging because they are open-ended, and I’ll cover best practices for how to tackle these. Using human evaluators remains a necessary option for many applications. However, given how slow and expensive human annotations can be, the goal is to automate the process. This book focuses on automatic evaluation, which includes both exact and subjective evaluation.

The rising star of subjective evaluation is AI-as-a-judge -- the approach of using AI to evaluate AI responses. It’s subjective because the score depends on what model and prompt the AI judge uses. While this approach is gaining rapid traction in the industry, it also invites intense opposition from those who believe that AI isn’t trustworthy enough for this important task. I’m especially excited to go deeper into this discussion, and I hope you will be too.

Challenges of Evaluating Foundation Models

Evaluating ML models has always been difficult. With the introduction of foundation models, evaluation has become even more so. There are multiple reasons why evaluating foundation models is more challenging than evaluating traditional ML models.

First, the more intelligent AI models become, the harder it is to evaluate them. Most people can tell if a first grader's math solution is wrong. Few can do the same for a PhD-level math solution. It's easy to tell if a book summary is bad if it's gibberish, but a lot harder if the summary is coherent. To validate the quality of a summary, you might need to read the book first. This brings us to a corollary: evaluation can be so much more time-consuming for sophisticated tasks. You can no longer evaluate a response based on how it sounds. You'll also need to fact-check, reason, and even incorporate domain expertise.

Second, the open-ended nature of foundation models undermines the traditional approach of evaluating a model against ground truths. With traditional ML, most tasks are close-ended. For example, a classification model can only output among the expected categories. To evaluate a classification model, you can evaluate its outputs against the expected outputs. If the expected output is category X but the model's output is category Y, the model is wrong. However, for an open-ended task, for a given input, there are so many possible correct responses. It's impossible to curate a comprehensive list of correct outputs.

Third, most foundation models are treated as black boxes, either because model providers choose not to expose models' details, or because application developers lack the expertise to understand them. Details such as the model architecture, training data, and the training process can reveal a lot about a model's strengths and weaknesses. Without those details, you can only evaluate a model by observing its outputs.

At the same time, evaluation benchmarks keep getting outdated. Ideally, evaluation should capture the full range of model capabilities. As AI progresses, evaluation needs to evolve to catch up. A benchmark becomes saturated for a model once the model achieves the perfect score or surpasses the human baseline. With foundation models, benchmarks are getting saturated fast. The benchmark [GLUE](#) (General Language Understanding Evaluation) came out in 2018 and became saturated in just a year, necessitating the introduction of [SuperGLUE](#) in 2019. Similarly, [NaturalInstructions](#) (2021) was replaced by [Super-NaturalInstructions](#) (2022).

Last but not least, the scope of evaluation has expanded for general-purpose models. With task-specific models, evaluation involves measuring a model's performance on its trained task. However, with general-purpose models, evaluation is not only about benchmarking on known tasks but also about discovering new tasks that models can do, and these might include tasks that extend beyond human capabilities. Evaluation takes on the added responsibility of exploring the potential and limitations of AI.

The good news is that the new challenges of evaluation have prompted many new methods and benchmarks. [Figure 3-1](#) shows that the number of papers on LLM evaluation was growing exponentially every month in the first half of 2023, from 2 papers a month to almost 35 papers a month.

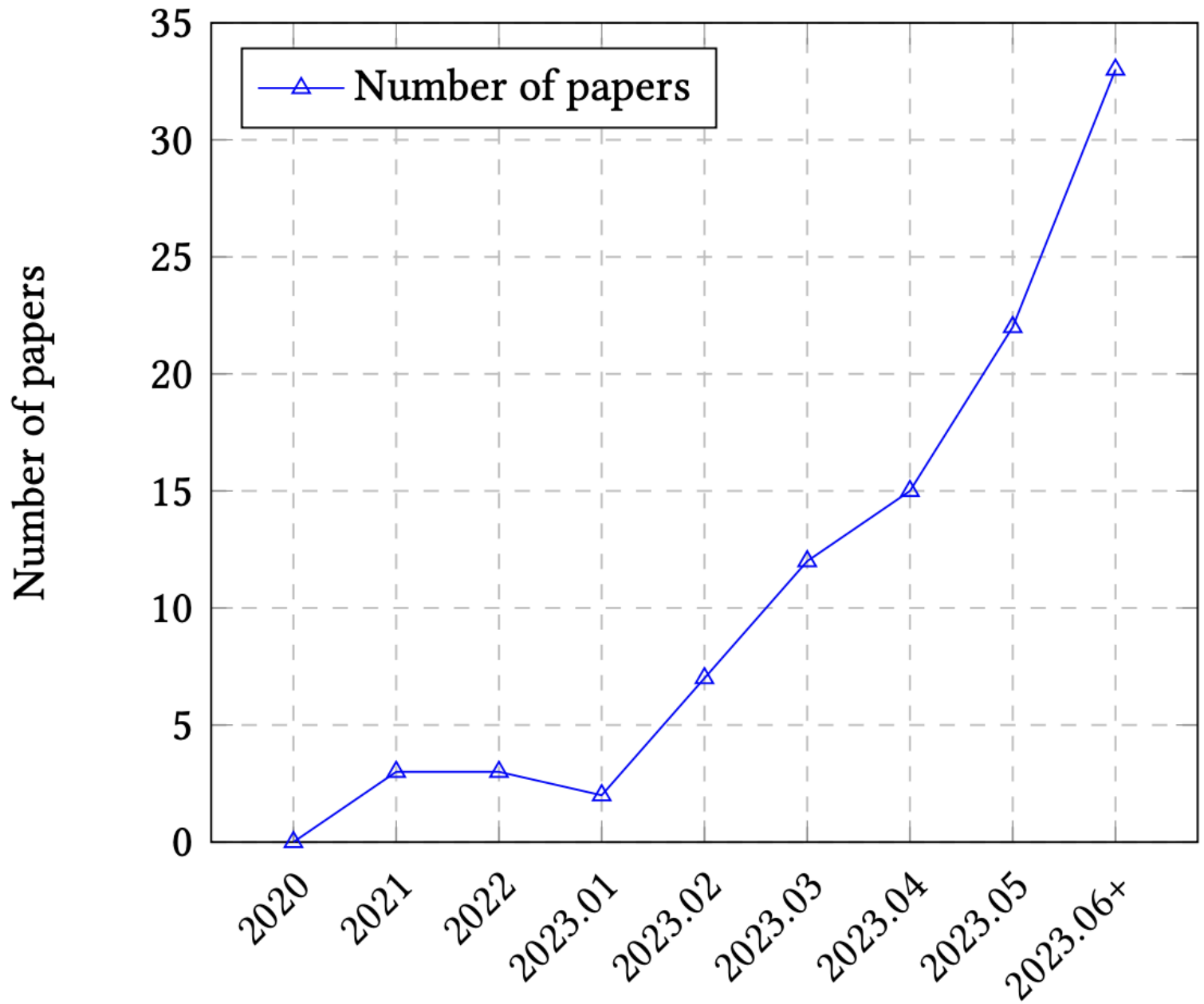


Figure 3-1. The trend of LLMs evaluation papers over time. Image from [Chang et al. \(2023\)](#).

In my own analysis of [top 1000 AI-related repositories on GitHub](#) by the number of stars¹, I found 40 repositories dedicated to evaluation (as of May 12, 2024). When plotting the number of evaluation repositories by their creation date, the growth curve looks exponential, as shown in [Figure 3-2](#).

Cumulative number of evaluation repositories month over month

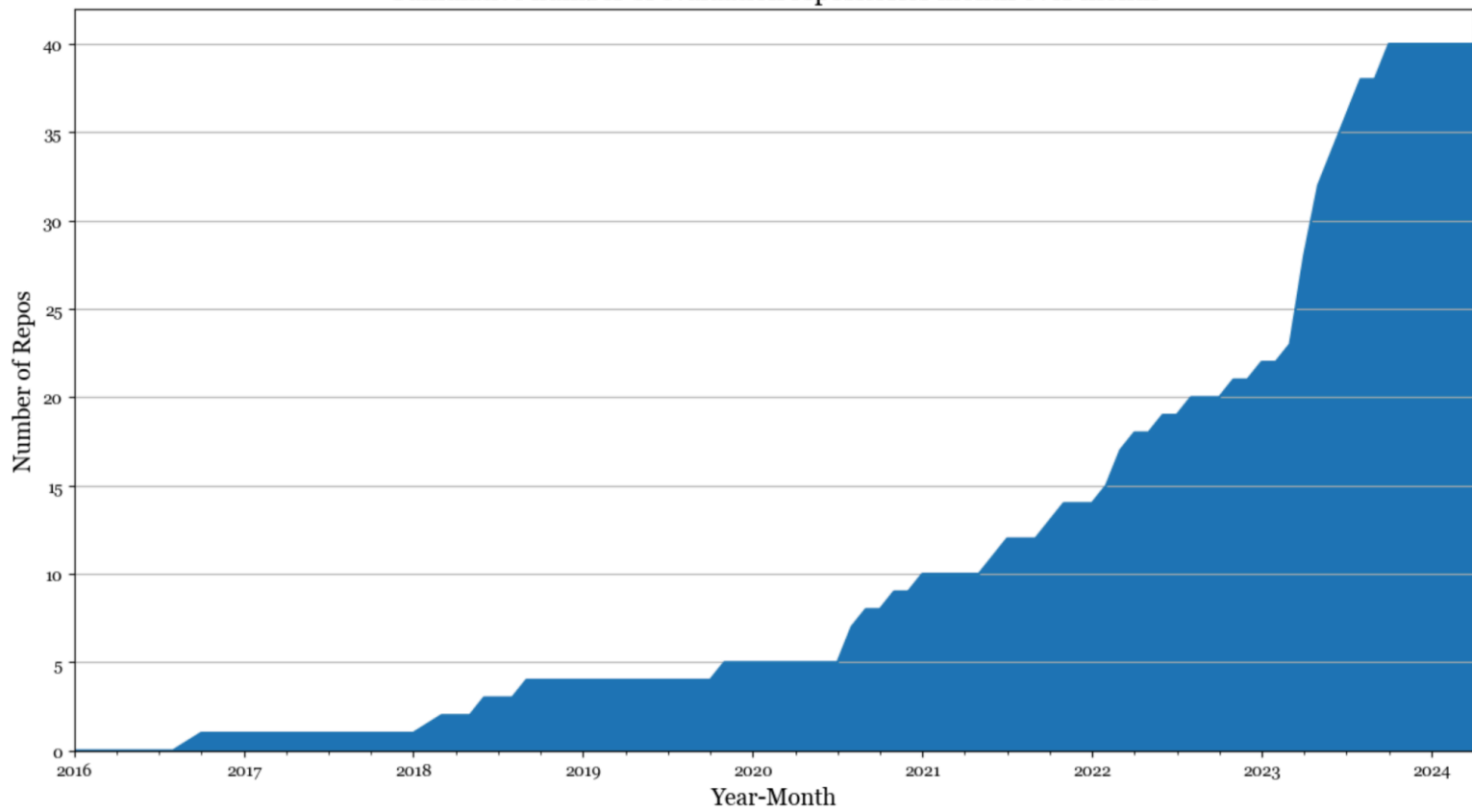


Figure 3-2. Number of open source evaluation repositories among the 1000 most popular AI repositories on GitHub.

The bad news is that despite the increased interest in evaluation, it lags behind in terms of interest in the rest of the AI engineering pipeline. Balduzzi et al. from DeepMind noted in their [paper](#) that “developing evaluations has received little systematic attention compared to developing algorithms.” According to the paper, experiment results are almost exclusively used to improve algorithms and are rarely used to improve evaluation. Recognizing the lack of investments in evaluation, [Anthropic](#) called on policymakers to increase government funding and grants both for developing new evaluation methodologies and analyzing the robustness of existing evaluations.

The number of tools for evaluation is small compared to the number of tools for modeling and training and AI orchestration, as shown in [Figure 3-3](#).

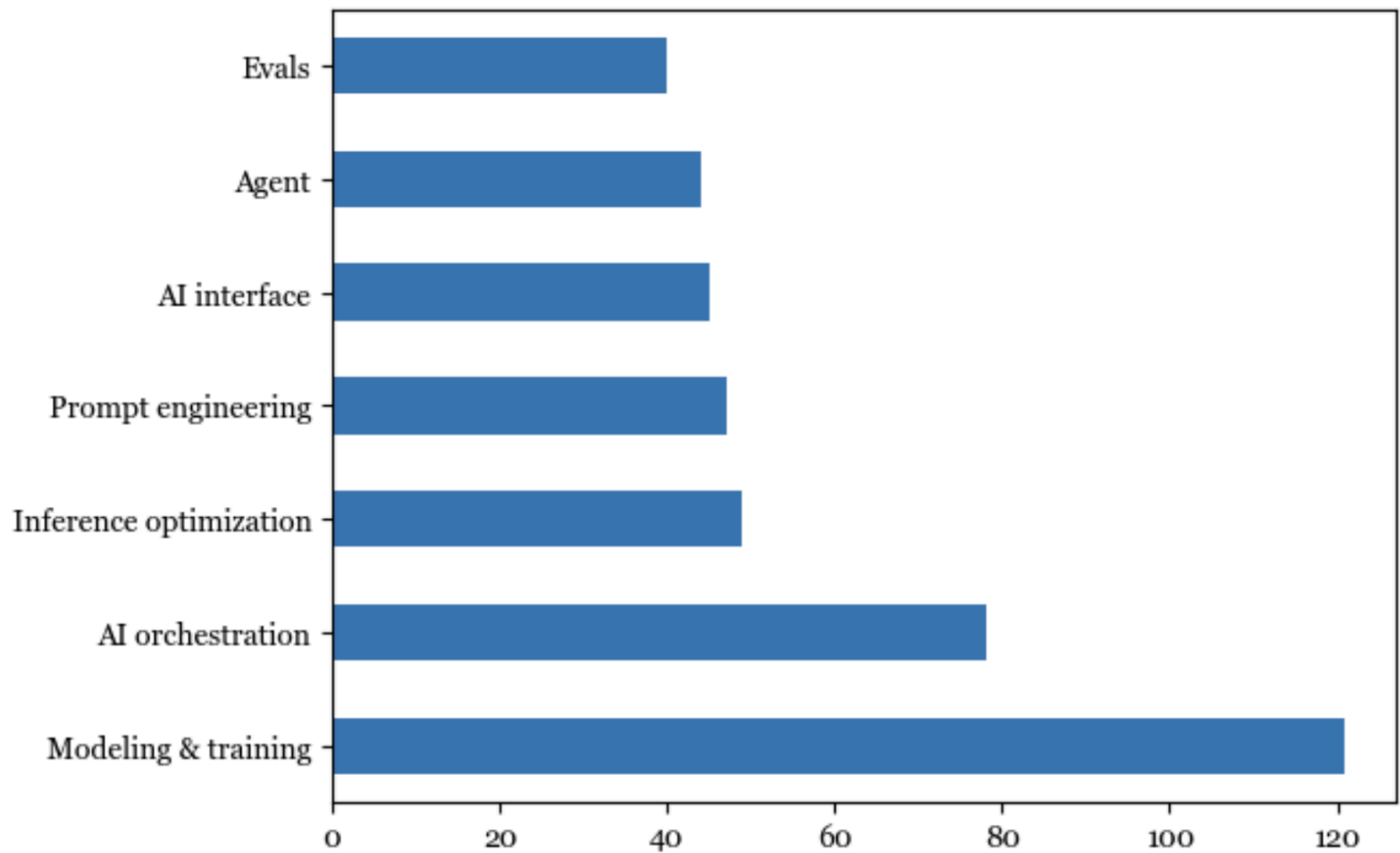


Figure 3-3. Evaluation lags behind other aspects of AI engineering in terms of open source tools, using the data from my list of 1000 most popular AI repositories on GitHub.

Inadequate investment leads to inadequate infrastructure, making it hard for people to carry out systematic evaluations. When asked how they are evaluating their AI applications, many people told me that they just eyeballed the results. Many have a small set of go-to prompts that they use to evaluate models. The process of curating these prompts is ad-hoc, usually based on the curator’s personal experience² instead of based on the application’s needs. You might be able to get away with this ad-hoc approach when getting a project off the ground, but it won’t be sufficient for application iteration. This book focuses on a systematic approach to evaluation.

Understanding Language Modeling Metrics

Foundation models evolved out of language models. Many foundation models still have language models as their main components. For these models, the performance of the language model component tends to be well-correlated to the foundation model’s performance on downstream applications ([Liu et al., 2023](#)). Therefore, a rough understanding of language modeling metrics can be quite helpful in understanding downstream performance.³

As discussed in Chapter 1, language modeling has been around for decades, popularized by Claude Shannon in his 1951 paper [Prediction and Entropy of Printed English](#). The metrics used to guide the development of language models haven’t changed much since then. Most autoregressive language models are trained using cross entropy or its relative, perplexity. When reading papers and model reports, you might also come across bits-per-character (BPC) and bits-per-byte (BPB), both are variations of cross entropy.

All four metrics -- cross entropy, perplexity, BPC, and BPB -- are closely related. If you know the value of one, you can compute the other three, given the necessary information. While I refer to them as language modeling metrics, they can be used for any model that generates sequences of tokens, including non-text tokens.

Recall that a language model encodes statistical information about languages. Statistically, given the context “*I like drinking __*”, the next word is more likely to be “tea” than “charcoal”. The more statistical information (how likely a token is to appear in a given context) that a model can capture, the better it is at predicting the next token.

In ML lingo, a language model *learns the distribution* of its training data. The better this model learns, the better it is at predicting what comes next in the training data, and the lower its training cross entropy. As with any ML model, you care about its performance not just on the training data, but also on the data relevant to you. In general, the closer your data is to a model’s training data, the lower the model’s cross entropy is to your data.

Compared to the rest of the book, this section is math-heavy. If you find it confusing, feel free to skip the math part and focus on the part discussing how to interpret these metrics. Even if you’re not training or finetuning language models, understanding these metrics can help with evaluating which models to use for your application. These metrics can occasionally be used for certain evaluation and data deduplication techniques, as discussed throughout this book.

Entropy

Entropy measures how much information, on average, a token carries. The higher the entropy, the more information each token carries, and the more bits are needed to represent a token.⁴

Let’s use a simple example to illustrate this. Imagine you want to create a language to describe positions within a square, as shown in [Figure 3-4](#). If your language has only two tokens, shown as (a) in [Figure 3-4](#), each token can tell you whether the position is upper or lower. Since there are only two tokens, one bit is sufficient to represent them. The entropy of this language is, therefore, 1.

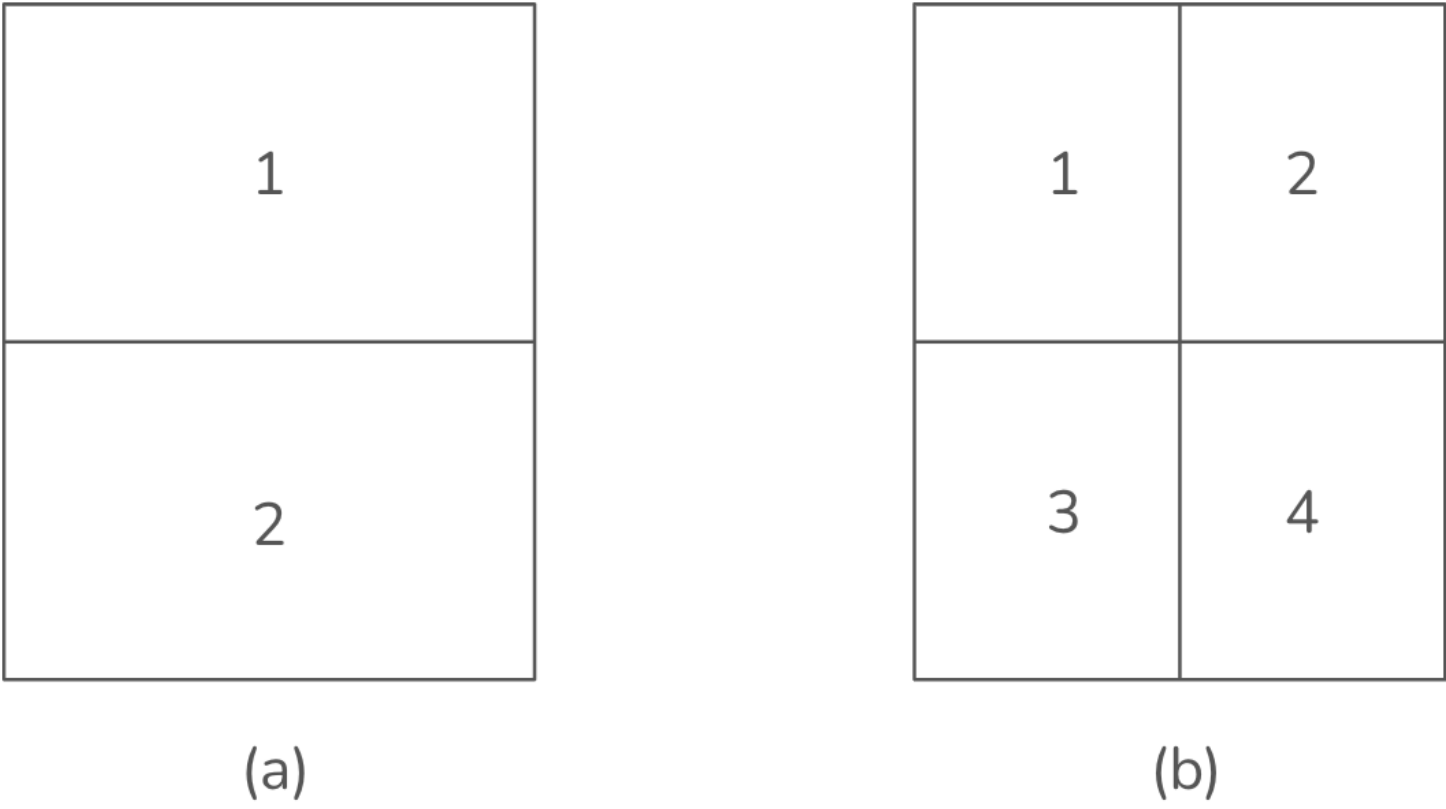


Figure 3-4. Two languages describe positions within a square. Compared to the language on the left (a), the tokens on the right (b) carry more information, but they need more bits to represent them.

If your language has 4 tokens, shown as (b) in [Figure 3-4](#), each token can give you a more specific position: upper-left, upper-right, lower-left, or lower-right. However, since there are now 4 tokens, you need two bits to represent them. The entropy of this language is 2. This language has higher entropy, since each token carries more information, but each token requires more bits to represent.

Intuitively, entropy measures how difficult it is to predict what comes next in a language. The lower a language’s entropy (the less information a token of a language carries), the more predictable that language. This is similar to how, if you can predict perfectly what I say next, what I say carries no new information to you.

Cross Entropy

When you train a language model on a dataset, your goal is to get the model to learn the distribution of this training data. In other words, your goal is to get the model to predict what comes next in the training data. A language model’s cross entropy on a dataset measures how difficult it is for the language model to predict what comes next in this dataset.

A model’s cross entropy on the training data depends on two qualities:

1. The training data’s predictability, measured by the training data’s entropy.
2. How the distribution captured by the language model diverges from the true distribution of the training data.

Entropy and cross entropy share the same mathematical notation H . Let P be the true distribution of the training data, and Q be the distribution learned by the language model.

- The training data’s entropy is therefore $H(P)$.
- The divergence of Q with respect to P can be measured using the Kullback–Leibler (KL) divergence, which is mathematically represented as

$$D_{KL}(P||Q)$$

- The model’s cross entropy with respect to the training data is therefore:

$$H(P,Q) = H(P) + D_{KL}(P||Q)$$

Cross entropy isn’t symmetric. The cross entropy of Q with respect to P -- $H(P,Q)$ -- is different from the cross entropy of P with respect to Q -- $H(Q,P)$.

A language model is trained to minimize its cross entropy with respect to the training data. If the language model learns perfectly from its training data, the model’s cross entropy will be exactly the same as the entropy of the training data. The KL divergence of Q with respect to P will then be 0. You can think of a model’s cross entropy as its approximation of the entropy of its training data.

Bits-per-character and Bits-per-byte

One unit of entropy and cross entropy is bits. If the cross entropy of a language model is 6 bits, this language model needs 6 bits to represent each token.

Since different models have different tokenization methods -- for example, one model uses words as tokens and another uses characters as tokens -- the number of bits per token isn’t comparable across models. Some use the number of *bits-per-character* (BPC) instead. If the number of bits per token is 6 and on average, each token consists of 2 characters, the BPC is $6/2 = 3$.

One complication with BPC arises from different character encoding schemes. For example, with ASCII, each character is encoded using 7 bits, but with UTF-8, a character can be encoded using anywhere between 8 and 32 bits. A more standardized metric would be *bits-per-byte* (BPB), the number of bits a language model needs to represent one byte of the original training data. If the BPC is 3 and each character is 7 bits, or $\frac{7}{8}$ of a byte, then the BPB is: $3 / (\frac{7}{8}) = 3.43$.

Cross entropy tells us how efficient a language model will be at compressing text. If the BPB of a language model is 3.43, meaning it can represent each original byte (8 bits) using 3.43 bits, this language model can compress the original training text to less than half the text’s original size.

Perplexity

Perplexity is the exponential of entropy and cross entropy. Perplexity is often shortened as PPL. Given a dataset with the true distribution P , its perplexity is defined as:

$$PPL(P) = 2^{H(P)}$$

The perplexity of a language model (with the learned distribution Q) on this dataset is defined as:

$$PPL(P, Q) = 2^{H(P, Q)}$$

If cross entropy measures how difficult it is for a model to predict the next token, perplexity measures the amount of uncertainty it has when predicting the next token. Higher uncertainty means there are more possible options for the next token.

Consider a language model trained to encode the 4 position tokens in [Figure 3-4](#) (b) perfectly. The cross entropy of this language model is 2 bits. If this language model tries to predict a position in the square, it has to choose among $2^2 = 4$ possible options. Thus, this language model has a perplexity of 4.

So far, I’ve been using *bit* as the unit for entropy and cross entropy. Each bit can represent 2 unique values, hence the base of 2 in the preceding perplexity equation.

Popular ML frameworks, including TensorFlow and PyTorch, use *nat* (natural log) as the unit for entropy and cross entropy. Nat uses the base of e , the base of natural logarithm⁵. If you use *nat* as the unit, perplexity is the exponential of e :

$$PPL(P, Q) = e^{H(P, Q)}$$

Due to the confusion around *bit* and *nat*, many people report perplexity, instead of cross entropy, when reporting their language models’ performance.

Perplexity Interpretation and Use Cases

As discussed, cross entropy, perplexity, BPC, and BPB are variations of language models’ predictive accuracy measurements. The more accurately a model can predict a text, the lower these metrics are. In this book, I’ll use perplexity as the default language modeling metric. Remember that the more uncertainty the model has in predicting what comes next in a given dataset, the higher the perplexity.

What’s considered a good value for perplexity depends on the data itself and how exactly perplexity is computed, such as how many previous tokens a model has access to. Here are some general rules of thumb:

More structured data gives lower expected perplexity

More structured data is more predictable. For example, HTML code is more predictable than everyday text. If you see an opening HTML tag like `<head>`, you can predict that there should be a closing tag `</head>` nearby. Therefore, the expected perplexity of a model on HTML code should be lower than the expected perplexity of a model on everyday text.

The bigger the vocabulary, the higher the perplexity

Intuitively, the more possible tokens there are, the harder it is for the model to predict the next token. For example, a model’s perplexity on a children’s book will likely be lower than the same model’s perplexity on War and Peace. For the same dataset, say in English, character-based perplexity (predicting the next character) will be lower than word-based perplexity (predicting the next word), because the number of possible characters is smaller than the number of possible words.

The longer the context length, the lower the perplexity

The more context a model has, the less uncertainty it will have in predicting the next token. In 1951, Claude Shannon evaluated his model’s cross entropy conditioned on under 10 previous tokens. As of this writing, a model’s perplexity can be computed conditioned on typically between 500 and 10,000 previous tokens, and possibly more, upperbounded by the model’s maximum context length.

For references, it’s not uncommon to see perplexity values as low as 3 or even lower. If all tokens in a hypothetical language have an equal chance of happening, a perplexity of 3 means that this model has a 1 in 3 chance of predicting the next token correctly. Given that a model’s vocabulary is in the order of 10,000s and 100,000s, these odds are incredible.

Other than guiding the training of language models, perplexity is useful in many parts of an AI engineering workflow. First, perplexity is a good proxy for a model’s capabilities. If a model’s bad at predicting the next token, its performance on downstream tasks will also likely be bad. OpenAI’s GPT-2 report shows that larger models, which are also more powerful models, consistently give lower perplexity on a range of datasets, as shown in [Figure 3-5](#). Sadly, following the trend of companies being increasingly more secretive about their models, many have stopped reporting their models’ perplexity.

	LAMBADA (PPL)	LAMBADA (ACC)	CBT-CN (ACC)	CBT-NE (ACC)	WikiText2 (PPL)	PTB (PPL)	enwik8 (BPB)	text8 (BPC)	WikiText103 (PPL)	1BW (PPL)
SOTA	99.8	59.23	85.7	82.3	39.14	46.54	0.99	1.08	18.3	21.8
117M	35.13	45.99	87.65	83.4	29.41	65.85	1.16	1.17	37.50	75.20
345M	15.60	55.48	92.35	87.1	22.76	47.33	1.01	1.06	26.37	55.72
762M	10.87	60.12	93.45	88.0	19.93	40.31	0.97	1.02	22.05	44.575
1542M	8.63	63.24	93.30	89.05	18.34	35.76	0.93	0.98	17.48	42.16

Figure 3-5. Larger GPT-2 models consistently give lower perplexity on different datasets. Source: [OpenAI, 2018](#).

WARNING

Perplexity might not be a great proxy to evaluate models that have been post-trained using techniques like SFT and RLHF⁶. Post-training is about teaching models how to complete tasks. As a model gets better at completing tasks, it might get worse at predicting the next tokens. A language model’s perplexity typically increases after post-training. Some people say that post-training *collapses* entropy. Similarly, quantization -- a technique that reduces a model’s numerical precision and, with it, its memory footprint⁷ -- can also change a model’s perplexity in unexpected ways.

Recall that the perplexity of a model with respect to a text measures how difficult it is for this model to predict this text. For a given model, perplexity is the lowest for texts that the model has seen and memorized during training. Therefore, perplexity can be used to detect whether a text was in a model’s training data. This is useful for detecting data contamination, e.g. if a benchmark’s data was in the model’s training data, making the model’s performance on this benchmark less trustworthy. This can also be used for deduplication of training data: e.g. only add new data to the existing training dataset if the perplexity of the new data is high.

Perplexity is the highest for unpredictable texts, such as texts expressing unusual ideas (like “my dog teaches quantum physics in his free time”) or gibberish (like “home cat go eye”). Therefore, perplexity can be used to detect unusual and gibberish texts.

HOW TO USE A LANGUAGE MODEL TO COMPUTE A TEXT’S PERPLEXITY

A model’s perplexity with respect to a text measures how difficult it is for the model to predict that text. Given a language model X, and a sequence of tokens [x₁, x₂, ... x_n] , X’s perplexity for this sequence is:

$$P(x_1, x_2, \hat{\epsilon}, x_n)^{-\frac{1}{n}} = \left(\frac{1}{P(x_1, x_2, \hat{\epsilon}, x_n)} \right)^{\frac{1}{n}} = \left(\prod_{i=1}^n \frac{1}{P(x_i|x_1, \hat{\epsilon}, x_{i-1})} \right)^{\frac{1}{n}}$$

with $P(x_i|x_1, \hat{\epsilon}, x_{i-1})$ denotes the probability that X assigns to the token X_i given the previous tokens $x_1, \hat{\epsilon}, x_{i-1}$.

To compute perplexity, you need access to the probabilities (or logprobs) the language model assigns to each next token. Unfortunately, not all commercial models expose their models’ logprobs, as discussed in Chapter 2.

Exact Evaluation

Perplexity and its related metrics help us understand the performance of the underlying language model, which is a proxy for understanding the model’s performance on downstream tasks. This section discusses how to measure a model’s performance on downstream tasks directly.

When evaluating models’ performance, it’s important to differentiate between exact and subjective evaluation. Exact evaluation produces judgment without ambiguity. For example, if the answer to a multiple-choice question is A and you pick B, your answer is wrong. There’s no ambiguity around that. On the other hand, essay grading is subjective. An essay’s score depends on who grades the essay. The same person, if asked twice some time

apart, can give the same essay different scores. Essay grading can become more exact with clear grading guidelines. As you’ll see in the next section, AI-as-a-judge is subjective. The evaluation result can change based on the judge and the prompt.

This section discusses two evaluation approaches that produce exact scores: functional correctness and similarity measurements against reference data. Note that this section focuses on evaluating open-ended responses (arbitrary text generation) as opposed to close-ended responses (such as classification). This is not because foundation models aren’t being used for close-ended tasks. In fact, many foundation model systems have at least a classification component, such as for intent classification or scoring. This section focuses on open-ended evaluation because close-ended evaluation is already well understood.

Functional Correctness

Functional correctness evaluation means evaluating a system based on whether it performs the intended functionality. For example, if you ask a model to create a website, does the generated website meet your requirements? If you ask a model to make a reservation at a certain restaurant, did the model succeed?

Functional correctness is the ultimate metric for evaluating the performance of any application, as it measures whether your application does what it’s intended to do. However, functional correctness isn’t always straightforward to measure, or its measurement can’t be easily automated.

Code generation is an example of a task for which functional correctness measurement can be automated. Functional correctness in coding is sometimes *execution accuracy*. Say you ask the model to write a Python function `gcd(num1, num2)` to find the greatest common denominator (gcd) of two numbers `num1` and `num2`. The generated code can then be input into a Python interpreter to check whether the code is valid and if it is, whether it outputs the correct result of a given pair `(num1, num2)`. For example, given the pair `(num1=15, num2=20)`, if the function `gcd(15, 20)` doesn’t return 5, the correct answer, you know that the function is wrong.

Long before AI was used for writing code, automatically verifying code’s functional correctness was standard practice in software engineering. Code is typically validated with [unit tests](#): code is executed in different scenarios to ensure that it generates the expected outputs. Functional correctness evaluation is how coding platforms like LeetCode and HackerRank validate the submitted solutions.

Popular benchmarks for evaluating AI’s code generation capabilities, such as OpenAI’s [HumanEval](#) and Google’s [MBPP](#) use functional correctness as their metrics. Benchmarks for text-to-SQL (generate SQL queries from natural languages) like [Spider](#) (Yu et al., 2018), [BIRD-SQL](#) (Li et al., 2023), and [WikiSQL](#) (Zhong et al., 2017) also rely on functional correctness.

A benchmark problem comes with a set of test cases. Each test case is a scenario the code should run and the expected output for that scenario.

Here’s an example of a problem in HumanEval:

```
from typing import List

def has_close_elements(numbers: List[float], threshold: float) -> bool:
    """ Check if in given list of numbers, are any two numbers closer to each other than given threshold """
    >>> has_close_elements([1.0, 2.0, 3.0], 0.5) False
    >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3) True
    """
```

Test cases (each `assert` statement represents a test case):

```
def check(candidate):
    assert candidate([1.0, 2.0, 3.9, 4.0, 5.0, 2.2], 0.3) == True
    assert candidate([1.0, 2.0, 3.9, 4.0, 5.0, 2.2], 0.05) == False
    assert candidate([1.0, 2.0, 5.9, 4.0, 5.0], 0.95) == True
    assert candidate([1.0, 2.0, 5.9, 4.0, 5.0], 0.8) == False
```

```
assert candidate([1.0, 2.0, 3.0, 4.0, 5.0, 2.0], 0.1) == True
assert candidate([1.1, 2.2, 3.1, 4.1, 5.1], 1.0) == True
assert candidate([1.1, 2.2, 3.1, 4.1, 5.1], 0.5) == False
```

When evaluating a model, for each problem, a number of code samples, denoted as k , are generated. A model solves a problem if any of the k code samples it generated passes all of that problem's test cases. The final score, called $pass@k$, is the fraction of the solved problems out of all problems. If there are 10 problems and a model solves 5 with $k=3$, then that model's $pass@3$ score is 50%. The more code samples a model generates, the more chance the model has at solving each problem, hence the greater the final score. This means that in expectation, $pass@1$ score should be lower than $pass@3$, which, in turn, should be lower than $pass@10$.

Another category of tasks whose functional correctness can be automatically evaluated is video game bots. If you create a bot to play Tetris, you can tell how good the bot is by the score it gets. Tasks with measurable objectives can typically be evaluated using functional correctness. For example, if you ask AI to schedule your workloads to optimize energy consumption, the AI's performance can be measured by how much energy it saves.⁸

Similarity Measurements Against Reference Data

If the task you care about can't be automatically evaluated using functional correctness, one common approach is to evaluate AI's outputs against reference data. For example, if you ask a model to translate a sentence from French to English, you can evaluate the generated English translation against the correct English translation.

Each example in the reference data follows the format (input, reference responses). An input can have multiple reference responses, such as multiple possible English translations of a French sentence. Reference responses are also called *ground truths* or *canonical responses*.

Since this evaluation approach requires reference data, it's bottlenecked by how much and how fast reference data can be generated. Reference data is generated typically by humans and increasingly by AIs. Using human-generated data as the reference means that we treat human performance as the gold standard, and AI's performance is measured against human performance. Human-generated data can be expensive and time-consuming to generate, leading many to use AI to generate reference data instead. AI-generated data might still need human reviews, but the labor needed to review is much less than the labor needed to generate reference data from scratch.

Generated responses more similar to the reference responses are better. There are four ways to measure the similarity between two open-ended texts:

- Asking an evaluator to make the judgment whether two texts are the same.
- Exact match: whether the generated response matches one of the reference responses exactly.
- Lexical similarity: how similar the generated response looks to the reference responses.
- Semantic similarity: how close the generated response is to the reference responses in meaning (semantics).

Evaluators used to compare two responses can be human or AI. However, if you're already using humans to make this comparison, you might not need reference data -- humans can evaluate the generated responses directly. Using AI evaluators, which is part of the AI-as-a-judge approach, is increasingly common, and will be the focus of the next section.

This section focuses on hand-designed metrics: exact match, lexical similarity, and semantic similarity. Scores by exact matching are binary (match or not), whereas the other two scores are on a sliding scale (such as between 0 and 1 or between -1 and 1). Despite the ease of use and flexibility of the AI-as-a-judge approach, hand-designed similarity measurements are still widely used in the industry for their exact nature.

NOTE

In this section, similarity measurements are used to evaluate the quality of a generated output. Similarity measurements are also used for many other use cases, including but not limited to:

- Retrieval and search: find items similar to a query.
- Ranking: rank items based on how similar they are to a query.
- Clustering: cluster items based on how similar they are to each other.
- Anomaly detection: detect items that are the lesst similar to the rest.
- Data deduplication: remove items that are too similar to other items.

Techniques discussed in this section will come up again throughout the book.

Exact match

It’s considered an exact match if the generated response matches one of the reference responses exactly. Exact matching works for tasks that expect short, exact responses such as simple math problems, common knowledge queries, and trivia-style questions. Here are examples of inputs that have short, exact responses:

- “*What’s 2 + 3?*”
- “*Who was the first woman to win a Nobel Prize?*”
- “*What’s my current account balance?*”
- “*Fill in the blank: Paris to France is like ____ to England.*”

There are variations to matching that take into account formatting issues. One variation is to accept any output that contains the reference response as a match. Consider the question “What’s 2+3?” The reference response is “5”. This variation accepts all outputs that contain “5”, including “The answer is 5” and “2+3 is 5”.

However, this variation can sometimes lead to the wrong solution being accepted. Consider the question “What year was Anne Frank born?” Anne Frank was born on June 12, 1929, so the correct response is 1929. If the model outputs “September 12, 1929”, the correct year is included in the output, but the output is factually wrong.

Beyond simple tasks, exact match rarely works. Given the original French sentence “Comment ça va?”, there are multiple possible English translations such as “How are you?”, “How is everything?”, and “How are you doing?” If the reference data contains only these three translations and a model generates “How is it going?”, the model’s response will be marked as wrong. The longer and more complex the original text, the more possible translations there are. It’s impossible to create an exhaustive set of possible responses for an input. For complex tasks, lexical similarity and semantic similarity work better.

Lexical similarity

Lexical similarity measures how much two texts overlap. You can do this by first breaking each text into smaller tokens. As discussed in Chapter 1, a token can be a character, a word, or a part of a word.

In its simplest form, lexical similarity can be measured by counting how many tokens two texts have in common. As an example, consider the reference response “*My cats scare the mice*” and two generated responses:

1. “*My cats eat the mice*”
2. “*Cats and mice fight all the time*”

Assuming that each token is a word. If you count overlapping of individual words only, response A contains 4 out of 5 words in the reference response (the similarity score is 80%), whereas response B only contains 3 out of 5 (the similarity score is 60%). Response A is, therefore, considered more

similar to the reference response.

In reality, lexical similarity is measured based on the overlapping of sequences of words, *n-grams*, instead of single words. A 2-gram (bigram) is a set of two words. “My cats scare the mice” consists of four bigrams: “my cats”, “cats scare”, “scare the”, and “the mice”. You measure what percentage of *n*-grams in reference responses is also in the generated response⁹. Lexical similarity is, therefore, also called *n-gram similarity*.

Common metrics for lexical similarity are BLEU, ROUGE, METEOR++, TER, and CIDEr. They differ in how exactly the overlapping is calculated. Before foundation models, BLEU, ROUGE, and their relatives were common, especially for translation tasks. Since the rise of foundation models, fewer benchmarks use lexical similarity. Examples of benchmarks that use these metrics are [WMT 2024](#), [COCO Captions](#), and [GEMv2](#).

A drawback of this method is that it requires curating a comprehensive set of reference responses. A good response can get a low similarity score if the reference set doesn’t contain any response that looks like it. On some benchmark examples, Adept found their model Fuyu performed poorly not because the model’s outputs were wrong, but [because some correct answers weren’t in reference data](#). [Figure 3-6](#) shows an example of an image-captioning task in which Fuyu generated a correct caption but was given a low score.



Fuyu’s caption: “A nighttime view of Big Ben and the Houses of Parliament.”

Reference captions: “A fast moving image of cars on a busy street with a tower clock in the background.”

“Lit up night traffic is zooming by a clock tower.”

“A city building is brightly lit and a lot of vehicles are driving by.”

“A large clock tower and traffic moving near.”

“there is a large tower with a clock on it.”

CIDEr Score: 0.4 (No reference caption mentions Big Ben or Parliament)

Figure 3-6. An example where Fuyu generated a correct option but was given a low score because of the limitation of reference captions.

Another drawback of this measurement is that higher lexical similarity scores don’t always mean better responses. For example, on HumanEval, a code generation benchmark, OpenAI found that BLEU scores for incorrect and correct solutions were similar. This indicates that optimizing for BLEU scores isn’t the same as optimizing for functional correctness ([Chen et al., 2021](#)).

Semantic similarity

Lexical similarity measures whether two texts look similar, not whether they have the same meaning. Consider the two sentences “*What’s up?*” and “*How are you?*”. Lexically, they are different -- there’s little overlapping in the words and letters they use. However, semantically, they are close. Conversely, similar-looking texts can mean very different things. “Let’s eat, grandma” and “Let’s eat grandma” mean two completely different things.

Semantic similarity aims to compute the similarity in semantics. This first requires transforming a text into a numerical representation, which is called an *embedding*. For example, the sentence “*the cat sits on a mat*” might be represented using an embedding that looks like this: `[0.11, 0.02, 0.54]`. Semantic similarity is, therefore, also called *embedding similarity*.

How embeddings work is discussed further in the section “A brief introduction to embedding”, but for now, let’s assume that you have a way to transform texts into embeddings. The similarity between two embeddings can be computed using metrics such as cosine similarity. Two embeddings that are exactly the same have a similarity score of 1. Two opposite embeddings have a similarity score of -1.

I’m using text examples, but semantic similarity can be computed for embeddings of any data modality, including images and audio. Semantic similarity for text is sometimes called semantic textual similarity.

WARNING

While I put semantic similarity in the exact evaluation category, it can be considered subjective, as different embedding algorithms can produce different embeddings. However, given two embeddings, the similarity score between them is computed exactly.

Mathematically, let A be an embedding of the generated response, and B an embedding of a reference response. The cosine similarity between A and B is computed as:

$$\frac{A \cdot B}{||A|| ||B||}$$

, with:

- $A \cdot B$
being the dot product of A and B

- $||A||$
being the Euclidean norm (also known as L^2 norm) of A. If A is `[0.11, 0.02, 0.54]`,
$$||A|| = \sqrt{0.11^2 + 0.02^2 + 0.54^2}$$

Metrics for semantic textual similarity include **BERTScore** (embeddings are generated by BERT) and **MoverScore** (embeddings are generated by a mixture of algorithms).

Semantic textual similarity doesn’t require a set of reference responses as comprehensive as lexical similarity does. However, the reliability of semantic similarity depends on the quality of the underlying embedding algorithm. Two texts with the same meaning can still have a low semantic similarity score if their embeddings are bad. Another drawback of this measurement is that the underlying embedding algorithm might require nontrivial compute and time to run.

Introduction to embedding

An *embedding* is a representation of complex data in a lower-dimensional space. Embeddings are typically vectors whose sizes are between 100 and 10,000.¹⁰ An embedding aims to preserve the relevant properties of the original data. If that sounds abstract, don’t worry, examples will make it concrete.

As used in the example, the sentence “*the cat sits on a mat*” might be represented using an embedding vector that looks like this: [0.11, 0.02, 0.54]. The *embedding size* (the number of elements in the embedding vector) is determined by the embedding algorithm. Models trained especially to produce embeddings include the open source models [BERT](#), [CLIP](#), [sentence-transformers](#), and many more proprietary embedding models provided as APIs¹¹. Table 3-1 shows the embedding sizes used by some popular models.

Table 3-1. Embedding sizes used by common models

Model	Embedding size
Google’s BERT	BERT base: 768 BERT large: 1024
OpenAI’s CLIP	Image: 512 Text: 512
OpenAI Embeddings API	text-embedding-3-small: 1536 text-embedding-3-large: 3072
Cohere’s Embed v3	embed-english-v3.0: 1024 embed-english-light-3.0: 384

Since many ML models require data to first be transformed into vector representations, many ML models, including language models such as GPT and LLaMA, also involve a step to generate embeddings. If you have access to these models, you can extract just the embeddings, though the quality might not be as good as the embeddings generated by models trained for embeddings.

The goal of the embedding algorithm is to produce embeddings that capture the essence of the original data. How do we verify that? The embedding vector [0.11, 0.02, 0.54] looks nothing like the original text “*the cat sits on a mat*”.

At a high level, an embedding algorithm is considered good if more similar texts have closer embeddings, measured by cosine similarity or related metrics. The embedding of the sentence “*the cat sits on a mat*” should be closer to the embedding of “*the dog plays on the grass*” than the embedding of “*AI research is super fun*”.

You can also evaluate the quality of embeddings based on their utility for your task. Embeddings are used in many tasks, including classification, topic modeling, recommender systems, and RAG. An example of benchmarks that measure embedding quality on multiple tasks is [MTEB](#), Massive Text Embedding Benchmark (Muennighoff et al., 2023).

I use texts as examples, but any data can have embedding representations. For example, ecommerce solutions like [Criteo](#) and [Coveo](#) have embeddings for products. [Pinterest](#) has embeddings for images, graphs, queries, and even users.

A new frontier is to create joint embeddings for data of different modalities. [CLIP](#) (Radford et al., 2021) was one of the first major models that could map data of different modalities, text and images, into a joint embedding space. [ULIP](#) (Xue et al., Dec 2022) aims to create unified representations of text, images, and 3D point clouds. [ImageBind](#) (Girdhar et al., May 2023) learns a joint embedding across six different modalities including text, images, and audio.

[Figure 3-8](#) visualizes CLIP’s architecture. The training goal is to get the embedding of a text close to the embedding of the corresponding image in the joint embedding space.

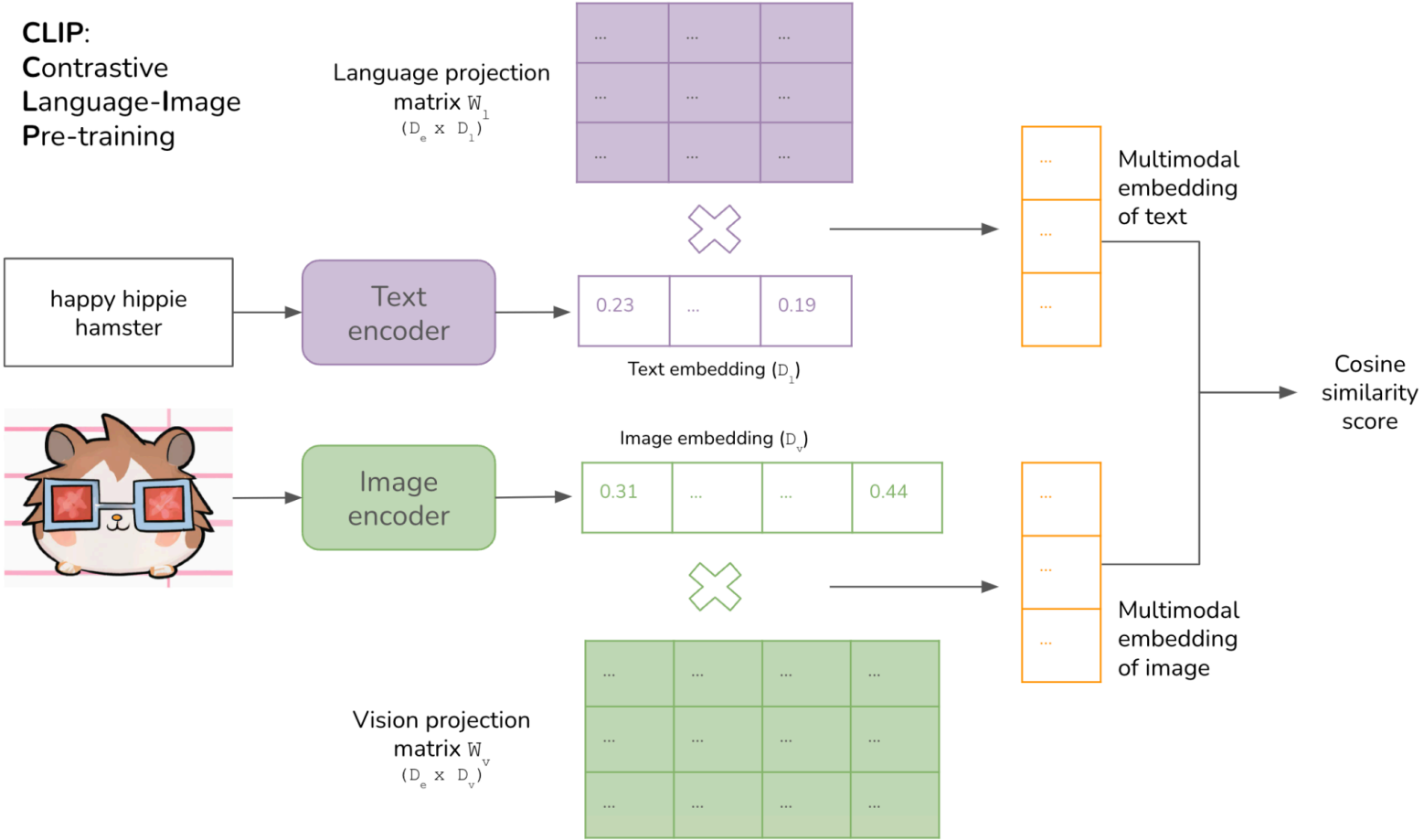


Figure 3-7. [CLIP](#)’s architecture (Radford et al., 2021).

A joint embedding space that can represent data of different modalities is called a *multimodal embedding space*. In a text-image joint embedding space, the embedding of an image of man fishing should be closer to the embedding of the text “a fisherman” than the embedding of the text “fashion show”. This joint embedding space allows embeddings of different modalities to be compared and combined. For example, this enables text-based image search. Given a text, it helps you find images closest to this text.

AI-as-a-Judge

The challenges of evaluating open-ended responses have led many teams to fall back on human evaluation. As AI has successfully been used to automate many challenging tasks, can AI automate evaluation as well? The approach of using AI to evaluate AI is called AI-as-a-judge or LLM-as-a-judge. An AI model that is used to evaluate other AI models is called an *AI judge*.^{[12](#)}

While the idea of using AI to automate evaluation has been around for a long time^{[13](#)}, it only became practical when AI models became capable of doing so, which was around 2020 with the release of GPT-3. As of this writing, AI-as-a-judge has become one of the most, if not the most, common methods for evaluating AI models in production. Most demos of AI evaluation startups I saw in 2023 and 2024 leveraged AI-as-a-judge in one way or another. [LangChain’s State of AI](#) report in 2023 mentioned that 58% of evaluations on their platform were done by AI judges. AI-as-a-judge is also an active area of research.

Why AI-as-a-Judge

AI judges are fast, easy to use, and relatively cheap compared to human evaluators. They can also work without reference data, which means they can be used in production where there is no reference data.

You can ask AI models to judge an output based on any criteria: correctness, repetitiveness, toxicity, wholesomeness, hallucinations, and more. This is similar to how you can ask a person to give their opinion about anything. You might think: “But you can’t always trust people’s opinions.” That’s true, and you can’t always trust AI’s judgments either. However, as each AI model is an aggregation of the masses, it’s possible for AI models to make judgments representative of the masses. With the right prompt for the right model, you can get reasonably good judgments on a wide range of topics.

Studies have shown that certain AI judges are strongly correlated to human evaluators. In 2023, [Zheng et al.](#) found that on their evaluation benchmark, MT-Bench, the agreement between GPT-4 and humans reached 85%, which is even higher than the agreement among humans (81%). [AlpacaEval](#) authors (Dubois et al., 2023) also found that their AI judges have a 0.98, near perfect, correlation with LMSYS’s Chat Arena leaderboard, which is evaluated by humans.

Not only can AI evaluate a response, but also explain its decision, which can be especially useful when you want to audit your evaluation results. [Figure 3-8](#) shows an example of GPT-4 explaining its judgment.

evaluate the quality of the following paragraph. give it a score from 1 to 5, and explain your decision.

Another drawback of this measurement is that higher lexical similarity scores don’t always mean better responses. For example, on HumanEval, a code generation benchmark, OpenAI found that BLEU scores for incorrect and correct solutions were similar. This indicates that optimizing for BLEU scores isn’t the same as optimizing for functional correctness (Chen et al., 2021).



I would rate the paragraph a **4.5 out of 5**.

Strengths:

1. **Clarity:** The paragraph clearly explains the drawback of using lexical similarity scores, providing a specific example.
2. **Evidence:** It effectively uses the HumanEval benchmark and OpenAI's findings to support the argument.
3. **Conciseness:** The paragraph is concise and directly to the point, making it easy to understand.

Areas for Improvement:

1. **Flow:** The transition between sentences is smooth, but the paragraph could benefit from a slightly stronger connection between the general statement and the specific example.

Figure 3-8. Not only can AI judges score, they also can explain their decisions.

Its flexibility makes AI-as-a-judge useful for a wide range of applications, and for some applications, it’s the only automatic evaluation option. Even when AI judgments aren’t as good as human judgments, they might still be good enough to guide an application’s development and provide sufficient confidence to get a project off the ground.

How to Use AI-as-a-Judge

There are many ways you can use AI to make judgments. For example, you can use AI to evaluate the quality of a response by itself, compare that response to reference data, or compare that response to another response. Here are naive example prompts for these three approaches.

1. Evaluate the quality of a response by itself, given the original question.

```
“Given the following question and answer, evaluate how good the answer is for the question. Use t
- 1 means very bad.
- 5 means very good.
Question: [QUESTION]
Answer: [ANSWER]
Score:”
```

2. Compare a generated response to a reference response to evaluate whether the generated response is the same as the reference response. This can be an alternative approach to human-designed similarity measurements.

```
“Given the following question, reference answer, and generated answer, evaluate whether this gene
Question: [QUESTION]
Reference answer: [REFERENCE ANSWER]
Generated answer: [GENERATED ANSWER]”
```

3. Compare two generated responses and determine which one is better or predict which one user will likely prefer. This is helpful for generating preference data for post-training alignment (discussed in Chapter 2), test-time sampling (discussed in Chapter 2), and ranking models using comparative evaluation (discussed in the next section).

```
“Given the following question and two answers, evaluate which answer is better. Output A or B.
Question: [QUESTION]
A: [FIRST ANSWER]
B: [SECOND ANSWER]
The better answer is:”
```

A general-purpose AI judge can be asked to evaluate a response based on any criteria, including criteria unique to your task. If you’re building a roleplaying chatbot, you might want to evaluate if a chatbot’s response is consistent with the role users want it to play, such as “*Does this response sound like something Dumbledore would say?*”. If you’re building an application to generate promotional product photos, you might want to ask “*From 1 to 5, how would you rate the trustworthiness of the product in this image?*” Table 3-2 shows common built-in AI-as-a-judge criteria offered by some AI tools.

Table 3-2. Examples of built-in AI-as-a-judge criteria offered by some AI tools.

AI Tools	Built-in criteria
Azure AI Studio	Groundedness, Relevance, Coherence, Fluency, GPT-Similarity
MLflow Evaluate	Faithfulness, Relevance
Langchain	Conciseness, Relevance, Correctness, Coherence, Harmfulness, Maliciousness, Helpfulness, Controversiality, Misogyny, Insensitivity, Criminality
RAGAS	Faithfulness, Answer relevance

It’s essential to remember that AI-as-a-judge criteria aren’t standardized. Azure AI Studio’s relevance scores might be very different from MLflow’s relevance scores. These scores depend on the underlying model and prompt being used as the judge.

How to prompt an AI judge is similar to how to prompt any AI application. In general, a judge’s prompt should clearly explain:

1. The task the model is to perform, such as to evaluate the relevance between a generated answer and the question.
2. The criteria the model should follow to evaluate, such as “*Your primary focus should be on determining whether the generated answer contains sufficient information to address the given question according to the ground truth answer.*” The more detailed the instruction, the better.
3. The scoring system, which can be:
 1. Classification: such as good/bad or relevant/irrelevant/neutral.
 2. Discrete numerical values: such as 1 to 5 or 1 to 10. Discrete numerical values can be considered a special case of classification, where each class has a numerical interpretation instead of a semantic interpretation.
 3. Continuous numerical values: such as between 0 and 1, e.g. when you want to evaluate the degree of similarity.

NOTE

Language models are generally better with texts than with numbers. It’s been reported that AI judges work better with classification than with numerical scoring systems.

For numerical scoring systems, discrete scoring seems to work better than continuous scoring. Empirically, the wider the range for discrete scoring, the worse the model seems to get. Typical discrete scoring systems are between 1 and 5.

Prompts with examples have shown to perform better. If you use a scoring system between 1 and 5, include examples of what a response with a score of 1, 2, 3, 4, or 5 looks like, and if possible, why a response receives a certain score.

Here’s part of the prompt used for the criteria [relevance](#) by Azure AI Studio. It explains the task, the criteria, the scoring system, an example of an input with a low score, and a justification for why this input has a low score. Part of the prompt was removed for brevity.

Your task is to score the relevance between a generated answer and the question based on the ground truth answer.

Your primary focus should be on determining whether the generated answer contains sufficient information to address the given question according to the ground truth answer.

If the generated answer contradicts the ground truth answer, it will receive a low score of 1-2.

For example, for question "Is the sky blue?", the ground truth answer is "Yes, the sky is blue." and the generated answer is "No, the sky is not blue."

In this example, the generated answer contradicts the ground truth answer by stating that the sky is blue.

This inconsistency would result in a low score of 1-2, and the reason for the low score would reflect this inconsistency.

Figure 3-9 shows an example of an AI judge that evaluates the quality of an answer given the question.

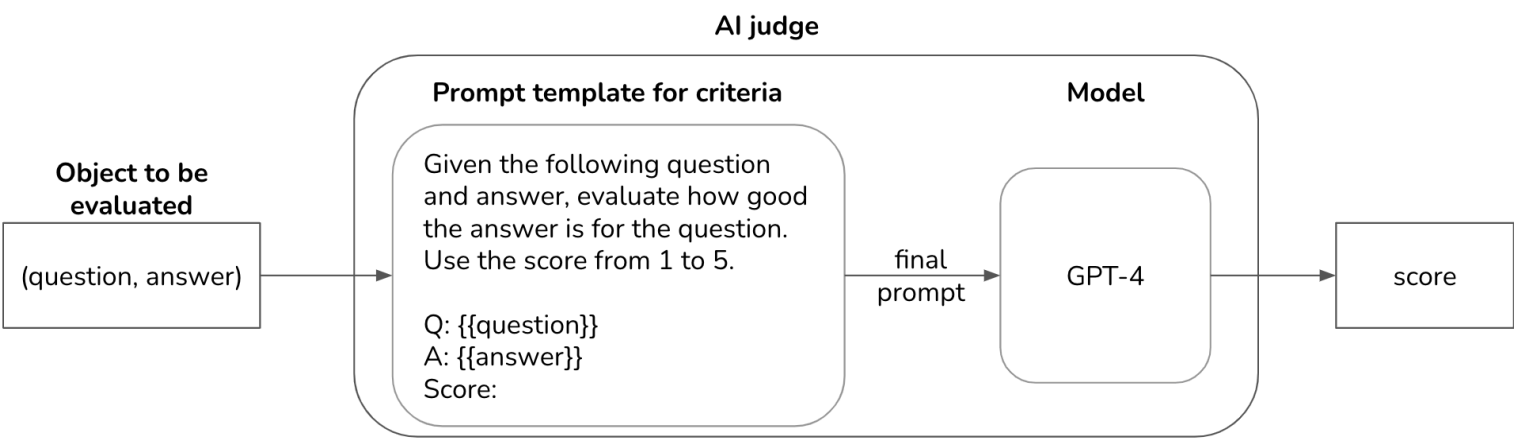


Figure 3-9. An example of an AI judge that evaluates the quality of an answer given a question.

Limitations of AI-as-a-Judge

Despite the many advantages of AI-as-a-judge, many teams are hesitant to adopt this approach. Using AI to evaluate AI seems tautological. The probabilistic nature of AI makes it seem too unreliable to act as evaluators. AI judges can potentially introduce nontrivial costs and latency to an application. Given these limitations, some teams see AI-as-a-judge as a fallback option when they don’t have any other way of evaluating their systems in production.

Inconsistency

For an evaluation method to be trustworthy, its results have to be consistent. Yet AI judges, like all AI applications, are probabilistic. The same judge, on the same input, can output different scores if prompted differently. Even the same judge, the same prompt can output different scores if run twice. This inconsistency makes it hard to reproduce or trust evaluation results.

It’s possible to get an AI judge to be more consistent. Chapter 2 discussed how to do so with sampling variables. Zheng et al. (2023) showed that including evaluation examples in the prompt can increase the consistency of GPT-4 from 65% to 77.5%. However, they acknowledged that high consistency may not imply high accuracy. It could also be that the judge consistently made the same mistakes. On top of that, including more examples makes prompts longer, and longer prompts mean higher inference costs. In Zheng et al.’s experiment, including more examples in their prompts caused their GPT-4 spending to quadruple.

Criteria ambiguity

Unlike many human-designed metrics, AI-as-a-judge metrics aren’t standardized, making them easily misinterpreted and misused. As of this writing, the open source tools MLflow, RAGAS, and LlamaIndex all have the built-in criterion *faithfulness* to measure how faithful a generated output is to the given context, but their instructions and scoring systems are all different. As shown in Table 3-3, MLflow uses a scoring system from 1 to 5, RAGAS uses 0 and 1, whereas LlamaIndex’s prompt asks the judge to output YES and NO.

The faithfulness scores outputted by these three tools won’t be comparable. If, given a (context, answer) pair, MLflow gives a faithfulness score of 3, RAGAS outputs 1, and LlamaIndex outputs NO. Which score would you use?

Table 3-3. Different tools can have very difficult default prompts for the same criteria.

Tool	Prompt [partially omitted for brevity]	Scoring system
MLflow	<p>Faithfulness is only evaluated with the provided output and provided context, please ignore the provided input entirely when scoring faithfulness. Faithfulness assesses how much of the provided output is factually consistent with the provided context. ...</p> <p>Faithfulness: Below are the details for different scores:</p> <ul style="list-style-type: none">- Score 1: None of the claims in the output can be inferred from the provided context.- Score 2: ...	1-5
RAGAS	<p>Your task is to judge the faithfulness of a series of statements based on a given context. For each statement you must return verdict as 1 if the statement can be verified based on the context or 0 if the statement can not be verified based on the context .</p>	0 and 1
LlamaIndex	<p>Please tell if a given piece of information is supported by the context. You need to answer with either YES or NO.</p> <p>Answer YES if any of the context supports the information, even if most of the context is unrelated. Some examples are provided below.</p> <p>Information: Apple pie is generally double-crust.</p> <p>Context: An apple pie is a fruit pie ... It is generally double-crust, with pastry both above and below the filling ...</p> <p>Answer: YES</p>	YES and NO

An application evolves over time, but the way it’s evaluated should ideally be fixed. This way, evaluation metrics can be used to monitor the application’s changes. However, AI judges are also AI applications, which means that they can also change over time.

Imagine that last month, your application’s coherence score was 90% and this month, this score is 92%. Does this mean that your application’s coherence has improved? It’s hard to answer this question unless you know for sure that the AI judges used in both cases are exactly the same. What if the judge’s prompt this month is different from the one last month? Maybe you switched to a slightly better-performing prompt or a coworker fixed a typo in last month’s prompt, and the judge this month is more lenient.

This can become especially confusing if the application and the AI judge are managed by different teams. The AI judge team might change the judges without informing the application team. As a result, the application team might mistakenly attribute the changes in the evaluation results to changes in the application, rather than the changes in the judges.

TIP

Do not trust any AI judge if you can’t see the model and the prompt used for the judge.

Evaluation methods take time to standardize. As the field evolves and more guardrails are introduced, I hope that AI judges will become a lot more standardized and reliable in the future.

Increased costs and latency

AI judges can be used to evaluate applications both during experimentation and in production. Many teams use AI judges as guardrails in production to reduce risks: returning only generated responses deemed good by the AI judge to users.

Using powerful models to evaluate responses can be expensive. If you use GPT-4 to both generate and evaluate responses, you'll do twice as many GPT-4 calls, approximately doubling your API costs. If you have three evaluation prompts because you want to evaluate three criteria -- say overall response's quality, factual consistency, and toxicity -- you'll increase your number of API calls four times.¹⁴

You can reduce costs by using weaker models or self-hosted models as the judges (see the next section “*What models can act as judges*”). You can also reduce costs with *spot-checking*: evaluating only a subset of responses. Spot-checking means you might fail to catch some failures. The larger the percentage of samples you evaluate, the more confidence you will have in your application, but also the higher the costs. Finding the right balance between cost and confidence might take trial and error. All things considered, AI judges are much cheaper than human evaluators.

Implementing AI judges in your production pipeline can add latency. If you evaluate responses before returning them to users, you face a tradeoff: reduced risk but increased latency. The added latency might make this option a non-starter for applications with strict latency requirements.

Biases of AI-as-a-judge

Human evaluators have biases, and so do AI judges. Different AI judges have different biases. Here are some of the common ones. Being aware of your AI judges' biases helps you integrate their scores correctly and even mitigate these biases.

AI judges tend to have the *self-enhance bias*, where a model favors its own responses over the responses generated by other models. The same mechanism that helps a model compute the most likely response to generate will also give this response a high score. In [Zheng et al.](#)'s experiment, GPT-4 favors itself with a 10% higher win rate, while Claude-v1 favors itself with a 25% higher win rate.

Many AI models have the *first-position bias*. An AI judge may favor the first answer in a pairwise comparison or the first in a list of options. This can be mitigated by repeating the same test multiple times with different orderings or with carefully crafted prompts. The position bias of AI is the opposite of that of humans. Humans tend to favor [the answer they see last](#), which is called *recency bias*.

Some AI judges have the *verbosity bias*, favoring lengthier answers, regardless of their quality. [Wu and Aji \(2023\)](#) found that both GPT-4 and Claude-1 prefer longer responses (~100 words) with factual errors over shorter, correct responses (~50 words). [Zheng et al. \(2023\)](#), however, discovered that GPT-4 is less prone to this bias than GPT-3.5, suggesting that this bias might go away as models become stronger.

On top of all these biases, AI judges have the same limitations as all AI applications, including privacy and IP. If you use a proprietary model as your judge, you'd need to send your data to this model. If the model provider doesn't disclose their training data, you won't know for sure if the judge is commercially safe to use.

Despite the limitations of the AI-as-a-judge approach, its many advantages make me believe that its adoption will continue to grow. However, AI judges should be supplemented with exact evaluation methods and/or human evaluation.

What Models Can Act as Judges?

The judge can either be stronger, weaker, or the same as the model being judged. Each scenario has its pros and cons.

A stronger judge makes sense. Shouldn't the exam grader be more knowledgeable than the exam taker? Not only can stronger models can make better judgments, but also help improve weaker models by guiding them to generate better responses.

You might wonder: if you already have access to the stronger model, why bother using a weaker model to generate responses? The answer is cost and latency. You might not have the budget to use the stronger model to generate all responses, so you use it to evaluate a subset of responses. For example, use a cheap in-house model to generate responses and GPT-4 to evaluate 1% of the responses.

The stronger model might also be too slow for your application. You can use a fast model to generate responses while the stronger but slower model does evaluation in the background. If the strong model thinks that the weak model's response is bad, remedy actions might be taken, such as updating the response with that of the strong model. Note that the opposite pattern is also common. You use a strong model to generate responses, and a weak model running in the background to do evaluation.

Using the stronger model as a judge leaves us with two challenges. First, the strongest model will be left with no eligible judge. Second, we need an alternative evaluation method to determine which model is the strongest.

Using a model to judge itself, *self-evaluation*, sounds like cheating, especially because of the self-enhance bias. However, self-evaluation can be great for sanity checks. If a model thinks its own response is incorrect, the model might not be that reliable. Beyond sanity checks, asking a model to evaluate itself can nudge a model to revise and improve its responses ([Press et al., 2022](#), [Gou et al. 2023](#), [Valmeekam et al. 2023](#))¹⁵. This example shows what self-evaluation might look like:

```
Prompt [from user]: What's 10+3?
First response [from AI]: 30
Self-critique [from AI]: Is this answer correct?
Final response [from AI]: No it's not. The correct answer is 13.
```

One open question is whether the judge can be weaker than the model being judged. Some argue that judging is an easier task than generating. Anyone can have an opinion about whether a song is good, but not everyone can write a song. Weaker models should be able to judge the outputs of stronger models.

[Zheng et al.](#) (2023) found that stronger models are better correlated to human preference, which makes people opt for the strongest models they can afford. However, this experiment was limited to general purpose judges. One research direction that I'm excited about is small, specialized judges. Specialized judges are trained to make specific judgments, using specific criteria and following specific scoring systems. A small, specialized judge can be more reliable than larger, general purpose judges for specific judgments.

Because of many possible ways to use AI judges, there are many possible specialized AI judges. Here I'll go over examples of three specialized judges: reward models, reference-guided judges, and preference models.

A reward model takes in a (prompt, response) pair and scores how good the response is given the prompt. Reward models have been successfully used in RLHF for many years. [Cappy](#) is another example of a reward model developed by Google (2023). Given the pair of (prompt, response), Cappy produces a score between 0 and 1, indicating how correct the response is. Cappy is a lightweight scorer with 360 million parameters, much smaller than general-purpose foundation models.

A reference-guided judge evaluates the generated response with respect to one or more reference responses. This judge can output a similarity score or a quality score (how good the generated response is compared to the reference responses). For example, [BLEURT](#) (Sellam et al., 2020) takes in a (candidate response, reference response) pair and outputs a similarity score between the candidate and reference response.¹⁶ [Prometheus](#) (Kim et al., 2023) takes in (prompt, generated response, reference response, scoring rubric) and outputs a quality score between 1 and 5, assuming that the reference response gets a 5.

A preference model takes in (prompt, response 1, response 2) as input and outputs which of the two responses is better (preferred by users) for the given prompt. This is perhaps one of the more exciting directions for specialized judges. Being able to predict human preference opens up many possibilities. As discussed in Chapter 2, preference data is essential for aligning AI models to human preference, and it's challenging and expensive to obtain. Having a good human preference predictor can generally make evaluation easier and models safer to use. There have been many initiatives in building preference models, including [PandaLM](#) (Wang et al., 2023) and [JudgeLM](#) (Zhu et al., 2023). [Figure 3-10](#) shows an example of how PandaLM works. It outputs not just which response is better, but also explains its rationale.

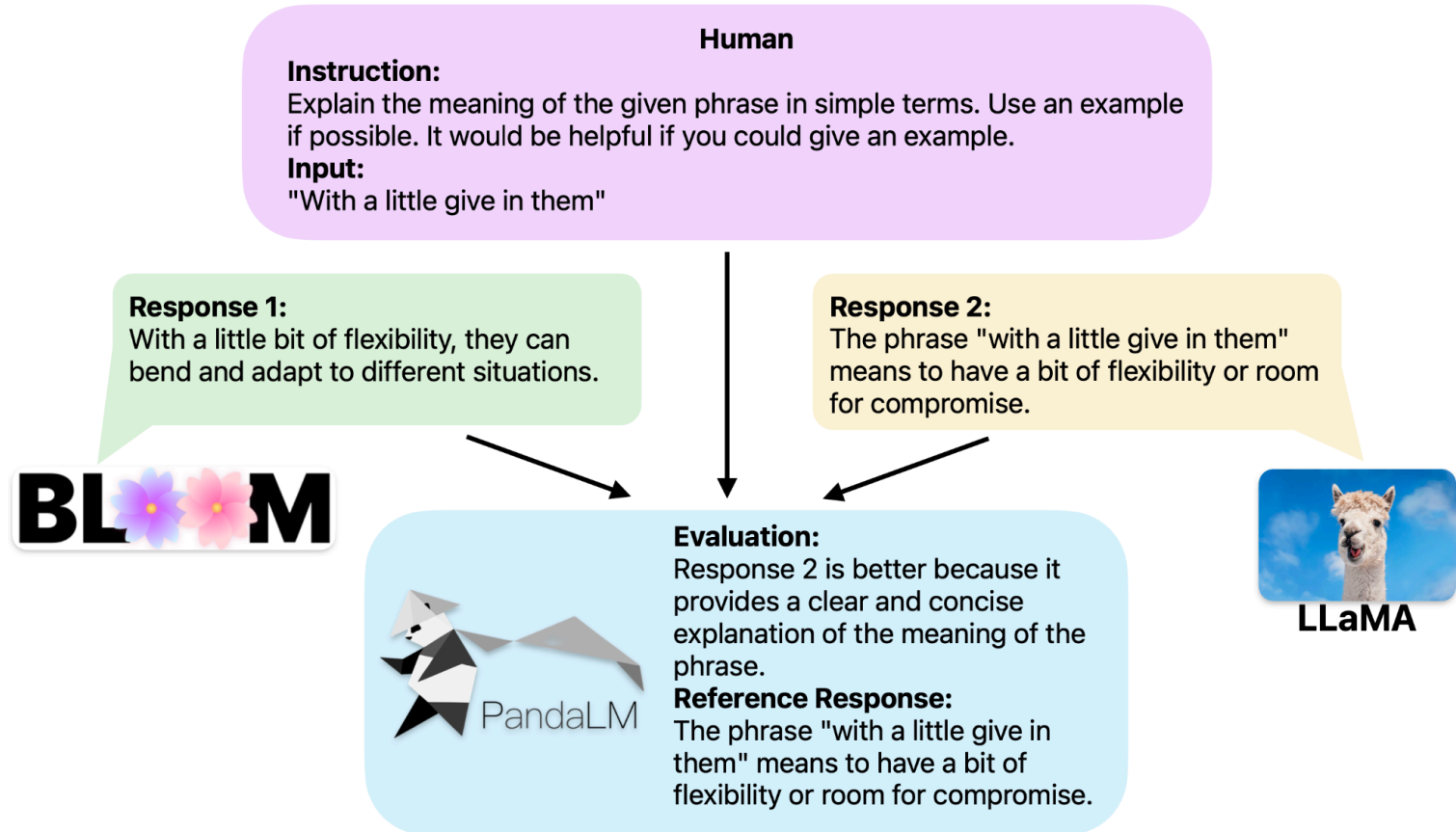


Figure 3-10. An example output of PandaLM, given a human prompt and two generated responses. Picture from the PandaLM paper.

Ranking Models with Comparative Evaluation

Often, you evaluate models not because you care about their scores, but because you want to know which model is the best for you. What you want is a ranking of these models. You can rank models using either independent evaluation or comparative evaluation.

With independent evaluation, you evaluate each model independently¹⁷ then rank them by their scores. For example, if you want to find out which candidate is the best at dancing, you evaluate each candidate individually, give them a score, then pick the candidate with the highest score.

With comparative evaluation, you evaluate models against each other and compute a ranking from comparison results. For the same dancing contest, you can ask all candidates to dance side-by-side and ask the judges which candidate's dancing they like the most, and pick the candidate whose dancing is preferred by most judges.


As discussed in Chapter 2, when labeling the quality of responses, it was discovered that it was a lot easier to compare two responses and pick the better one than to give each response a concrete score. You can leverage this for model evaluation. Instead of independently evaluating how good each model is, you compare them against each other.

In AI, comparative evaluation was first used in 2021 by [Anthropic](#) to rank different models. It also powers the popular LMSYS's [Chatbot Arena](#) leaderboard that ranks models using scores computed from pairwise model comparisons from the community.


Many model providers use comparative evaluation to evaluate their models in production. Figures 3-10 shows an example of ChatGPT asking its users to compare two outputs side-by-side. These outputs could be generated by different models, or by the same model with different sampling variables.


Which response do you prefer?

Your choice will help make ChatGPT better.

 Response 1
To write your dictionary to a JSON file in Python, you can use the ``json`` module, which is part of Python's standard library. Here's a simple example of how you can do it:

```
python  
  
import json
```

 Copy code

 Response 2
To write your dictionary to a JSON file in Python, you can use the ``json`` module, which is part of Python's standard library. The ``json.dump()`` function can be used to serialize your dictionary into a JSON formatted stream to a file. Here's a simple example of how you can do it:

```
python
```


 Copy code

Figure 3-11. ChatGPT occasionally asks users to compare two outputs side-by-side.

For each request, two or more models are selected to respond. An evaluator, which can be human or AI, picks the winner.

NOTE

When doing comparative evaluation, allow for ties to avoid a winner being picked arbitrarily when both options are equally good.

Comparative evaluation shouldn't be confused with A/B testing. In A/B testing, a user sees one model to be evaluated at a time. In comparative evaluation, a user sees multiple models to be evaluated at the same time.

Each comparison is called a *match*. This process results in a series of comparisons, as shown in Table 3-3.

Table 3-4. Examples of a history of pairwise model comparisons.

Match #	Model A	Model B	Winner
1	Model 1	Model 2	Model 1
2	Model 3	Model 10	Model 10
3	Model 7	Model 4	Model 4
...			

The probability that model A is preferred over model B is the *win rate* of A over B. We can compute this win rate by looking at all matches between A and B and calculating the percentage in which A wins.

If there are only two models, ranking them is straightforward. The model that wins more often ranks higher. The more models there are, the more challenging ranking becomes. Let's say that we have five models with the empirical win rates between model pairs as shown in Table 3-4. It's not obvious, from looking at the data, how these five models should be ranked.

Table 3-5. Example win rates of five models. A >> B denotes the event that A is preferred to B.

Model pair #	Model A	Model B	# matches	A >> B
1	Model 1	Model 2	1000	90%
2	Model 1	Model 3	1000	40%
3	Model 1	Model 4	1000	15%
4	Model 1	Model 5	1000	10%
5	Model 2	Model 3	1000	60%
6	Model 2	Model 4	1000	80%
7	Model 2	Model 5	1000	80%
8	Model 3	Model 4	1000	70%
9	Model 3	Model 5	1000	10%
10	Model 4	Model 5	1000	20%

Given comparative signals, a *rating algorithm* is then used to compute a ranking of models. Typically, this algorithm first computes a score for each model and then ranks models by their scores.

Comparative evaluation is new in AI but has been around for almost a century in other industries. It’s especially popular in sports and video games. Many rating algorithms developed for these other domains can potentially be adapted to evaluating AI models, such as Elo, Bradley-Terry, and TrueSkill. LMSYS’s Chatbot Arena originally used Elo to compute models’ ranking but later switched to the Bradley-Terry algorithm because they found Elo sensitive to the order of evaluators and prompts.¹⁸

A ranking is correct if, for any model pair, the higher-ranked model is more likely to win in a match against the lower-ranked model. If model A ranks higher than model B, users should prefer model A to model B more than half the time.

Through this lens, model ranking is a predictive problem. We compute a ranking from historical match outcomes and use it to predict future match outcomes. Different ranking algorithms can produce different rankings, and there’s no ground truth for what the correct ranking is. The quality of a ranking is determined by how good it is in predicting future match outcomes. My analysis of Chat Arena’s ranking shows that the produced ranking is good, at least for model pairs with sufficient matches. See the Appendix for the analysis.

Challenges of Comparative Evaluation

When you evaluate each model independently, the heavy-lifting part of the process is in designing the benchmark and metrics to gather the right signals. Computing scores to rank models is easy. With comparative evaluation, both signal gathering and model ranking are challenging. This section goes over the three common challenges.

Scalability bottlenecks

Comparative evaluation is data-intensive. The number of model pairs grows quadratically with the number of models. In January 2024, LMSYS evaluated 57 models, corresponding to 1,596 model pairs. These 57 models were ranked using 244,000 comparisons, averaging 153 comparisons per

model pair. This is a small number considering the wide range of tasks we want a foundation model to do.

Fortunately, we don't always need direct comparisons between two models to determine whether one is better. Ranking algorithms typically assume *transitivity*. If model A ranks higher than B, and B ranks higher than C, then A ranks higher than C. This means that if the algorithm is certain that A is better than B and B is better than C, it doesn't need to compare A against C to know that A is better.

However, it's unclear if this transitivity assumption holds for AI models. Many papers that analyze Elo for AI evaluation cite transitivity assumption as a limitation ([Boubdir et al.](#), [Balduzzi et al.](#), [Munos et al.](#)). They argued that human preference is not necessarily transitive. In addition, non-transitivity can happen because different model pairs are evaluated by different evaluators and on different prompts.

There's also the challenge of evaluating new models. With independent evaluation, only the new model needs to be evaluated. With comparative evaluation, the new model has to be evaluated against existing models, which can change the ranking of existing models.

This also makes it hard to evaluate private models. Imagine you've built a model for your company, using internal data. You want to compare this model with public models to decide whether it would be more beneficial to use a public one. If you want to use comparative evaluation for your model, you'll likely have to collect your own comparative signals and create your own leaderboard or pay one of those public leaderboards to run private evaluation for you.

The scaling bottleneck can be mitigated with better matching algorithms. So far, we've assumed that models are selected randomly for each match, so all model pairs appear in approximately the same number of matches. However, not all model pairs need to be equally compared. Once we're confident about the outcome of a model pair, we can stop matching them against each other. An efficient matching algorithm should sample matches that reduce the most uncertainty in the overall ranking.

Lack of standardization and quality control

One way to collect comparative signals is to crowdsource comparisons to the community the way LMSYS Arena does. Anyone can go to [their website](#), enter a prompt, get back two responses from two anonymous models, and vote for the better one. Only after voting is done are the model names revealed.

The benefit of this approach is that it captures a wide range of signals and is relatively difficult to game. However, the downside is that it's hard to enforce standardization and quality control.

First, anyone with Internet access can use any prompt to evaluate these models, and there's no standard on what should constitute a better response. It might be a lot to expect these volunteers to fact-check the responses, so they might unknowingly prefer responses that sound better but are factually incorrect.

Some people might prefer polite and moderate responses, while others might prefer responses without a filter. This is both good and bad. It's good because it helps capture human preference in the wild. It's bad because human preference in the wild might not be appropriate for all use cases. Some might even maliciously pick the toxic responses as the preferred ones, polluting the ranking.

Second, crowdsourcing comparisons require users to evaluate models outside of their working environments. Without real-world grounding, test prompts might not reflect how these models are being used in the real world. People might just use the first prompts that come to mind and are unlikely to use sophisticated prompting techniques.

Among [33,000 prompts](#) published by LMSYS Arena in 2023, 180 of them are “*hello*” and “*hi*”, which account for 0.55% of the data, and this doesn't yet count variations like “*hello!*”, “*hello.*”, “*hola*”, “*hey*”, and so on. There are many brainteasers. The question “*X has 3 sisters, each has a brother. How many brothers does X have?*” was asked 44 times.

Simple prompts are easy to respond to, making it hard to differentiate models' performance. Evaluating models using too many simple prompts can pollute the ranking.

If a public leaderboard doesn’t support sophisticated context construction, such as augmenting the context with relevant documents retrieved from your internal databases, its ranking won’t reflect how well a model might work for your RAG system. The ability to generate good responses is different from the ability to retrieve the most relevant documents.

One potential way to enforce standardization is to limit users to a set of predetermined prompts. However, this might impact the leaderboard’s ability to capture diverse use cases. LMSYS instead lets users use any prompts, but then filter out [hard prompts](#) using their internal model, and rank models using only these hard prompts.

Another way is to use only evaluators that we can trust. We can train evaluators on the criteria to compare two responses or train them to use practical prompts and sophisticated prompting techniques. This is the approach that Scale uses with [their private comparative leaderboard](#). The downside of this approach is that it’s expensive and it can severely reduce the number of comparisons we can get.

Another option is to incorporate comparative evaluation into your products and let users evaluate models during their workflows. For example, for the code generation task, you can suggest users two code snippets inside the user’s favorite code editor, and users pick the better one. As shown at the beginning of this section, ChatGPT and Gemini are already doing it.

Some people who have deployed this technique in production pointed out to me that most users don’t read both options and just randomly click on one. This can introduce a lot of noise to the results. However, the signals from the small percentage of users who vote correctly can sometimes be sufficient to help determine which model is better.

Some teams have told me that they prefer AI to human evaluators. AI might not be as good as trained human experts, but might be more reliable than random Internet users.

From comparative performance to absolute performance

For many applications, we don’t necessarily need the best possible models. We need a model that is good enough. Comparative evaluation tells us which model is better. It doesn’t tell us how good a model is, and whether this model is good enough for our use case. Let’s say we obtained the ranking that model B is better than model A. Any of the following scenarios could be valid:

1. Model B is good, but model A is bad.
2. Both model A and model B are bad.
3. Both model A and model B are good.

Other forms of evaluation are needed to help us determine which scenario is true.

Imagine that we’re using model A for customer support help, and model A can resolve 70% of all the tickets. Consider model B, which wins against A 51% of the time. It’s unclear how this 51% win rate will be converted to the downstream customer support application. Several people have told me that in their experience, a 1% change in the win rate can induce a huge performance boost in some applications, but just a minimal boost in other applications.

When deciding to swap out A for B, human preference isn’t everything. We also care about other factors like cost. Not knowing what performance boost to expect makes it hard to do the cost-benefit analysis. If model B costs twice as much as A, comparative evaluation isn’t sufficient to help us determine if the performance boost from B will be worth the added cost.

The Future of Comparative Evaluation

Given so many limitations of comparative evaluation, you might wonder if there’s a future to it. There are many benefits to comparative evaluation. First, as discussed in the section “Alignment” in Chapter 2, people have found that it’s easier to compare two outputs than to give each output a concrete score. As models become stronger, surpassing human performance, it might become impossible for human evaluators to give model responses concrete scores, and comparative evaluation might remain the only option.

Second, comparative evaluation aims to capture the quality we care about: human preference. It reduces the pressure to have to constantly create more benchmarks to catch up with the ever-expanding capabilities of AI. Unlike benchmarks that become useless when models’ performance achieves perfect scores, comparative evaluations will never get saturated as long as newer stronger models are being introduced.

Comparative evaluation is relatively hard to game, as there’s no easy way to cheat like training your model on reference data. For this reason, many trust the results of public comparative leaderboards like Chatbot Arena more than any public benchmark.

I believe that comparative evaluation can give us discriminating signals about models that can’t be obtained otherwise. For offline evaluation, it can be a great addition to intrinsic evaluation or evaluation benchmarks. For online evaluation, it can be complementary to A/B testing.

Summary

The stronger AI models become, the higher the potential for catastrophic failures, which makes evaluation even more important. At the same time, evaluating open-ended, powerful models is challenging. These challenges make many teams turn towards human evaluation. Having humans in the loop for sanity check is always helpful, and in many cases, human evaluation is essential. However, this chapter focused on different approaches to automatic evaluation.

This chapter starts with a discussion on why foundation models are harder to evaluate than traditional ML models. While many new evaluation techniques are being developed, investments in evaluation still lag behind investments in model and application development.

Since many foundation models have a language model component, we zoomed into language modeling metrics including perplexity and cross entropy. Many people I’ve talked to find these metrics confusing, so I included a section on how to interpret these metrics and leverage them in evaluation and certain data processing techniques.

This chapter then shifted the focus to the different approaches to evaluate open-ended responses, including functional correctness, similarity scores, and AI-as-a-judge. The first two evaluation approaches are exact while AI-as-a-judge evaluation is subjective.

Unlike exact evaluation, subjective metrics are highly dependent on the judge. Their scores need to be interpreted in the context of what judges are being used. Scores aimed to measure the same quality by different AI judges might not be comparable. AI judges, like all AI applications, should be iterated upon, meaning their judgments change. This makes them unreliable as benchmarks to track an application’s changes over time. While promising, AI judges should be supplemented with exact evaluation methods, human evaluation, or both.

When evaluating models, you can evaluate each model independently and then rank them by their scores. Alternatively, you can rank them using comparative signals: which of the two models is better? Comparative evaluation is common in sports, especially chess, and is gaining traction in AI evaluation. Both comparative evaluation and the post-training alignment process need preference signals, which are expensive to collect. This motivated the development of preference models: specialized AI judges that predict which response users prefer.

While language modeling metrics and hand-designed similarity measurements have existed for some time, AI-as-a-judge and comparative evaluation have only gained adoption with the emergence of foundation models. Many teams are figuring out how to incorporate them into their evaluation pipelines. Figuring out how to build a reliable evaluation pipeline to evaluate open-ended applications is the topic of the next chapter.