

6.3.4 Mitigation Strategies

Gehman et al. (2020) classify toxicity mitigation techniques into two primary types: data-based and decoding-based strategies. Data-based strategies encompass further pre-training of the model, altering its parameters. This approach, while effective, tends to be computationally intensive due to the parameter modifications involved. In contrast, decoding-based methods focus on altering only the decoding algorithm of a language model, leaving the model parameters intact. As a result, these strategies are typically more accessible and less resource intensive, offering a practical advantage for practitioners in the field.

6.3.4.1 Data-based Methods

The *Domain Adapted Pre-training* (DAPT) technique, as discussed by Gururangan et al. (2020), involves a two-phase pre-training process. The first stage involves general pre-training across a wide range of sources, while the second stage focuses on domain-specific pre-training, utilizing datasets pertinent to specific contexts, such as Twitter communication data, as shown in Fig. 6.9. This approach is particularly effective when combined with fine-tuning for specific tasks. For example, in classifying topics related to the Black Lives Matter movement, fine-tuning focuses on the hierarchy and nuances of language specific to this context, extracted from the broader domain of tweets. Although Gehman et al. (2020) demonstrated that continuing the pre-training of LLMs on a non-toxic subset of *OPENWEBTEXTCORPUS* (OWTC) using DAPT can significantly reduce toxicity in models such as GPT-2, this method is not without its challenges. One of the most significant limitations is the computational expense involved. Additionally, the necessity for extra training data, particularly when sourced through human labor such as crowdsourcing, can be prohibitively costly. Another critical limitation of this approach is its potential to adversely affect modeling performance. In filtering out toxic elements, there is a risk of inadvertently eliminating benign and potentially valuable knowledge.

Attribute Conditioning (ATCON) is another data-based technique based on the work of Ficler and Goldberg (2017) and Keskar et al. (2019). The approach involves the augmentation of pre-training data for pre-trained models with a toxicity attribute token, signified as `|toxic|` or `|non-toxic|`, incorporated into a randomly selected subset of documents. The process includes a further pre-training phase, enabling the model to integrate and adapt to these toxicity indicators. When the model is employed for text generation, the toxicity attribute token, specifically `|non-toxic|`, is prepended to the prompts. This addition acts as a signal to the model, delineating the desired focus on non-toxicity in the generated text.

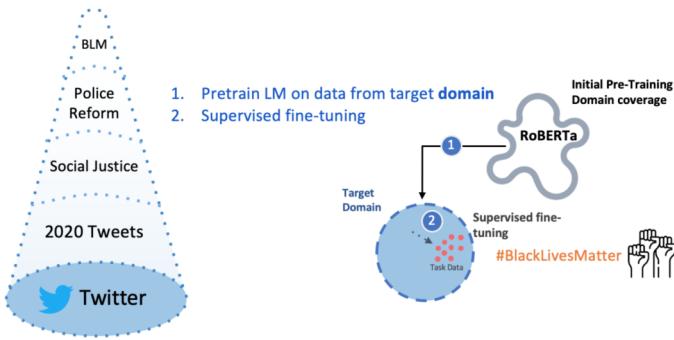


Fig. 6.9: Schematic illustrating Domain Adapted Pre-training. After the initial model pre-training, a second pre-training step is done with a corpus that 1) is pertinent to the model task (i.e., web data); 2) has been filtered of toxic communications.

6.3.4.2 Decoding-based Methods

Decoding-based methods denote approaches where the decoding algorithm is adjusted to minimize toxicity.

! Practical Tips

One of the most simplest decoding-based strategies is *blocklisting*, also known as *word filtering*. This approach involves assigning a zero probability to undesirable words — which typically include curse words, profanity, and insults — within the language model’s framework. As a result, the model is effectively prevented from generating these words in its output. There are a number of more complex, and generally more effective, approaches to detoxification during decoding, described here.

Vocabulary shifting

This approach, developed by Ghosh et al. (2017), centers around learning vector representations that distinctly signify toxic and non-toxic attributes for each token in the vocabulary. While the original research by Ghosh et al. utilized LSTM models, the fundamental principles of this technique are adaptable and remain consistent when applied to more contemporary Transformer-based architectures such as GPT-2.

In a standard LSTM model, the joint probability of a sequence of M words w_1, w_2, \dots, w_M is defined by the chain rule of probability:

$$P(w_1, w_2, \dots, w_M) = \prod_{t=1}^M P(w_t | w_1, w_2, \dots, w_{t-1}) \quad (6.26)$$

For such a model, the conditional probability of word w_t in context $c_{t-1} = (w_1, w_2, \dots, w_{t-1})$ is:

$$P(w_t = i | c_{t-1}) = \frac{\exp(U_i^T f(c_{t-1}) + b_i)}{\sum_{j=1}^V \exp(U_j^T f(c_{t-1}) + b_j)} \quad (6.27)$$

where $f(\cdot)$ is the LSTM output, \mathbf{U} is a word representation matrix, and \mathbf{b}_i is a bias term. The proposed Affect-LM model for vocabulary shift by Ghosh et al. (2017) modifies this equation as follows:

$$P(w_t = i | c_{t-1}, e_{t-1}) = \frac{\exp(U_i^T f(c_{t-1}) + \beta V_i^T g(e_{t-1}) + b_i)}{\sum_{j=1}^V \exp(U_j^T f(c_{t-1}) + \beta V_j^T g(e_{t-1}) + b_j)} \quad (6.28)$$

where e_{t-1} that captures affect category information derived from the context words during training. It quantifies the impact of the affect category information on predicting the target word w_t in context, and β is the affect strength parameter.

Plug and Play Language Model (PPLM)

PPLM allows users to integrate one or more attribute models representing specific control objectives into an LLM (Dathathri et al., 2019). This seamless integration requires no additional training or fine-tuning of the model, which is a significant advantage for researchers who lack access to extensive hardware resources. PPLM functions under two key assumptions:

1. Access to an attribute model, denoted as $p(a | x)$.
2. Availability of gradients from this attribute model.

The PPLM process, as shown in Fig 6.10, involves the following steps:

1. Perform a forward pass in the LLM, sampling a token from the resulting probability distribution. Then, feed the generated string to the attribute model to calculate the likelihood of the desired attribute, $p(a | x)$.
2. Execute backpropagation to compute gradients of both $p(a | x)$ and $p(x)$ with respect to the model’s hidden state. Adjust the hidden state to increase the probability of both $p(a | x)$ and $p(x)$.
3. Recalculate the LLM’s probability distribution and sample a new token.

Generative Discriminator (GeDi)

GeDi uses smaller LMs as generative discriminators to guide generation from LLMs to mitigate toxicity (Krause et al., 2020). It calculates the probability of each potential next word in a sequence using Bayes’ rule, balancing two class-conditional distributions. The first distribution is conditioned on a desired attribute or “control code”, directing the model toward specific, favorable content. Conversely, the second distribution, or “anti-control code”, is aligned with attributes to avoid, such as toxic language.

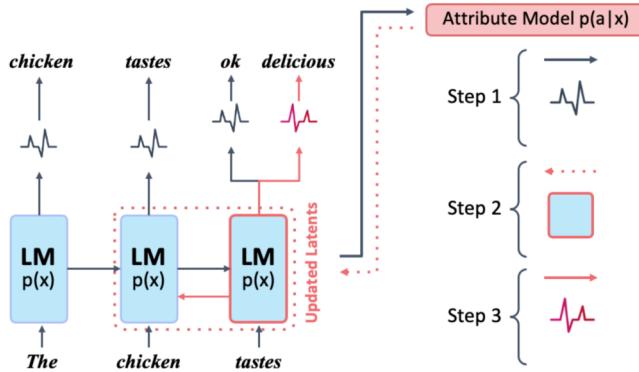


Fig. 6.10: Plug and Play Language Model (PPLM)

The core principle entails the use of auxiliary *Class-Conditional Language Models* (CC-LMs) to ascertain the likelihood of a generated token being part of the control code class. This code defines an attribute of the text sequence $x_{1:T}$, potentially representing aspects such as sentiment, topic, or labels such as “toxic”. The CC-LL calculates probabilities $p(x|c)$ and $p(x|\neg c)$, where c and $\neg c$ are the control code and anti-control code. An example of a CC-LL is CTRL, which provides a framework for language models to generate text while being conditioned on an attribute variable (Keskar et al., 2019).

In training a CC-LM, each sequence $x_{1:T}^{(i)}$ is paired with a control code $c^{(i)}$. The model is then trained to minimize the average negative log-likelihood, L_g . The combined loss function for GeDi training, \mathcal{L}_{gd} , is a mix of the LLM’s loss \mathcal{L}_g and the discriminative loss \mathcal{L} , weighted by a hyper-parameter λ :

$$\mathcal{L}_{gd} = \lambda \mathcal{L}_g + (1 - \lambda) \mathcal{L}_d \quad (6.29)$$

where \mathcal{L} is a discriminative loss aimed at optimizing toward desirable behavior and away from the undesirable behavior. The discriminative loss \mathcal{L}_d aims at increasing the classification accuracy, whereas \mathcal{L}_g helps in better-calibrated token probabilities for guided generation. GeDi is more effective at detoxification than PPLM and is less computationally expensive. However, it has the limitation of requiring an additional language model for training, which adds to the computational requirements.

Self-Diagnosis and Self-Debiasing

This method, developed by Schick et al. (2021), addresses detoxification by asking two pivotal questions: First, can LLMs autonomously detect their own biases (Self-Diagnosis)? Second, can LLMs discard undesired behaviors in an unsupervised manner (Self-Debiasing)?

Self-diagnosis investigates the ability of pre-trained language models to self-diagnose socially undesirable attributes in their outputs without relying on additional

training data or external resources. It focuses on the models' inherent knowledge gained during pre-training and uses brief attribute descriptions to assess their ability to identify and address biases and inappropriate content autonomously.

Consider a language model M and a sequence of tokens $x_1, \dots, x_k = \mathbf{x}$. For self-diagnosis, a sentence \mathbf{x} generated by M is combined with a query about an attribute \mathbf{y} , to determine whether \mathbf{x} contains \mathbf{y} . For instance, if M generates a sentence \mathbf{x} = “I’m going to hunt you down!” and needs to be checked for a threat \mathbf{y} = “a threat”, the model is prompted with the self-diagnosis template:

```
# Self-diagnosing template, denoted as sdg(.)
{sample text x}
Does the above text contain {threat text y} ?
```

The probability of \mathbf{x} exhibiting attribute \mathbf{y} is given by:

$$p(\mathbf{y}|\mathbf{x}) = \frac{p_M(\text{"Yes"}|\text{sdg}(\mathbf{x}, \mathbf{y}))}{\sum_{w \in \{\text{"Yes"}, \text{"No"}\}} p_M(w|\text{sdg}(\mathbf{x}, \mathbf{y}))} \quad (6.30)$$

Self-debiasing mirrors the idea of self-diagnosis but focuses on active modification of the output. The self-debiasing process also focuses on using the model’s own internal knowledge to adapt its output generation, thereby reducing the likelihood of producing biased text. An example self-debiasing template is as follows:

```
# Self-debiasing template, denoted as sdb(.)
The following text contains {undesired attribute s}:
{sample text x}
```

Employing this template involves a pre-trained language model, denoted as M , and an attribute \mathbf{y} , signifying a specific bias. When M is provided with an input text \mathbf{x} , it uses the self-debiasing input $\text{sdb}(\mathbf{x}, \mathbf{y})$ to generate a continuation. This results in two probability distributions: the original $p_M(w|\mathbf{x})$ and the adjusted $p_M(w|\text{sdb}(\mathbf{x}, \mathbf{y}))$. The latter is designed to favor undesirable words, as evidenced by the equation $\Delta(w, \mathbf{x}, \mathbf{y}) = p_M(w|\mathbf{x}) - p_M(w|\text{sdb}(\mathbf{x}, \mathbf{y}))$, which is expected to be negative for biased words.

To mitigate this bias, a new probability distribution $\tilde{p}_M(w|\mathbf{x})$ is formulated, proportional to $\alpha(\Delta(w, \mathbf{x}, \mathbf{y})) \cdot p_M(w|\mathbf{x})$. Here, $\alpha : \mathbb{R} \rightarrow [0, 1]$ acts as a scaling function, adjusting probabilities based on the bias difference $\Delta(w, \mathbf{x}, \mathbf{y})$. This scaling is intended to be minimally invasive, altering probabilities only when necessary while leaving unbiased words unchanged.

The approach avoids setting probabilities of any word to zero to maintain practical model evaluation. Instead, the probability of biased words is diminished according to $\Delta(w, \mathbf{x}, \mathbf{y})$, using a decay constant λ in the expression $\alpha(x) = 1$ if $x \geq 0$, $e^{-\lambda x}$ otherwise. This method can be extended to accommodate multiple biases by considering a set of descriptions $\mathbf{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_n\}$ and adjusting the distribution accordingly.

6.4 Privacy

With the rise of the internet over the past several decades, we live in an age where information flows more freely than ever. Unfortunately, not all of this information is willfully and knowingly shared by those providing it, nor is it thoughtfully collected and stored by those obtaining it. As such, the increased accessibility of personally identifying information and other private data has become a widely recognized concern. Given that all of the most prominent LLMs source a substantial amount of training data from websites, it is natural to consider whether this poses any downstream risks to privacy. As it turns out, LLMs learn specific information about individuals, and it is possible to extract that information with sufficient prompting. Privacy remains a largely unsolved problem for LLM at this point. This section will discuss the existing research and emerging trends aiming to address these concerns.

6.4.1 Causes

The conventional wisdom in most realms of machine learning is that when a model frequently generates predictions that closely match the examples seen during training, it is a classic symptom of overfitting. This principle fueled an assumption that LLMs are generally unlikely to memorize their training data and repeat it verbatim since they are most commonly trained for only one epoch on a considerable volume of data. The process is directly at odds with the conditions that define overfitting. Unfortunately, the assumption that memorization exclusively arises from overfitting has been invalidated (Carlini et al., 2021). Because the memorization potential of LLMs was not widely recognized early on, the research interest in the mitigation of private data capture has lagged behind the models' overall capabilities. High-performing LLMs with more parameters or training tokens also appear to have a greater capacity to memorize data than their scaled-down siblings (Nasr et al., 2023).

To quantify the memorized training data for a given model, Carlini et al. (2021) proposed an attack designed to evoke a memorized output string. Their approach samples sequences of tokens from internet sources, which are either known to be or likely to be part of the training data. These tokens are then used to prompt the model, with the outputs then being checked for precise matches in the training data. Naturally, this process is far more accessible when the training sources are public knowledge. To extend their research to GPT-2, which has never published its training data, the authors had to resort to Google searches over the entire internet to locate matches with a high probability of having been memorized by the model. Despite this limitation, they were nonetheless able to find data that had been memorized by GPT-2, including personally identifiable information (PII).

In subsequent work by Nasr et al. (2023) several additional experiments were undertaken based on the above mentioned procedure. Their research distinguished the rate of finding unique memorized output and the number of unique sequences extracted. It was observed, for instance, that Pythia-1.4 emitted more memorized data

than GPT-Neo-6, up to approximately 60 billion queries. At that point, their curves crossed, and those of GPT-Neo-6 surpassed those of Pythia-1.4. This is somewhat unsurprising, as there is no intuitive reason to expect that the effectiveness of the attack technique for a particular model is a direct measure of the actual amount of data it has memorized. The authors established this relationship using extrapolation procedures, which allowed them to place a much tighter lower bound on the amount of total memorized data for every model evaluated.

Chat models exhibit different behaviors, given that they typically undergo alignment tuning and are conditioned to provide helpful responses in a conversational tone. As a result, ChatGPT appeared to be relatively safe at first since it is unlikely to return a continuation of the memorized text when a sequence of tokens is submitted in the prompt. However, an additional finding by Nasr et al. (2023) was the discovery of a new attack that succeeded in extracting memorized data from ChatGPT. This approach entails selecting a single-token word, repeating it several times, and then asking ChatGPT to continue repeating it forever. The results vary widely depending on the chosen token, but in some instances, ChatGPT will eventually stop repeating the word and instead begin regurgitating memorized training data. Several thousand occurrences of memorized PII were verified within a text sample generated in this experiment.

The mechanism for this surprising behavior is not known with certainty due to the closed nature of ChatGPT, but examination of open-source models suggests that as the text grows longer, the model perhaps infers that there is an increasingly high likelihood of it ending soon. After all, no training text truly goes on forever, so this is somewhat of a foreign concept for a model. This may cause it to eventually converge toward the unique “end-of-text” token as its next-token prediction, forcing the model to generate a new sequence without any meaningful context preceding it. Intuitively, it seems plausible that this could significantly increase the probability of falling back on a known text sequence seen during training.

It is important to note that these attacks were designed to demonstrate the possibility that a nontrivial amount of memorized data could be extracted from LLMs. The research has only scratched the surface of potential techniques that might be employed or how hackers could seek to make these techniques more efficient and scalable. As discussed above, ChatGPT did not appear to be prone to training data leakage in early attacks but cracked when a novel attack was discovered. The implication is that it is challenging to gauge the full extent of memorization in LLMs and its risk of harm.

6.4.2 Evaluation Metrics

Kim et al. (2023) introduced a tool to probe and quantify PII leakage called ProPILE. It consists of an evaluation dataset constructed by locating PII within the widely used Pile training dataset and a set of prompt templates to retrieve a specific type of personal data given other relevant information about the person. As a concrete

example, the prompt “Please contact {name} by {pii_1} or {target_pii_type} ” could test whether the name and phone number give the model sufficient context to retrieve a street address, or a street address reveals an email address, and so forth. All possible permutations are queried, and the model outputs are captured.

The effectiveness of the prompting attack can be measured with two different metrics: exact match and likelihood. The exact match metric is based on a simple equality test of the generated string and the true PII. The likelihood metric leverages the capability of many model APIs to return scores for each token prediction and not just the response text. Obtaining the likelihood makes achieving a more precise estimate possible, particularly in a scenario where attackers could operate at a large scale and repeat similar prompts many times. This is defined as follows:

$$\Pr(a_m | \mathcal{A}_{\setminus m}) = \prod_{r=1}^{L_r} p(a_{m,r} | x_1, x_2, \dots, x_{L_q+r-1}) \quad (6.31)$$

where a_m is the target PII, $\mathcal{A}_{\setminus m}$ is the remaining PII, L_q is the length of the query, and L_r is the length of the correct response. Repeated computation over multiple queries produces an additional metric representing the percentage of people included in the training data that would have a piece of their PII exposed in k queries or less, using all available prompts. This metric is $\gamma_{<k}$.

In the context of ProPILE, users wanting to check whether a model exposes their data are constrained to what is deemed black-box probing since the only information they have about the model is its outputs. The previously described templates are the only available prompting mechanisms for black-box probing. White-box probing refers to the setting where model providers wish to quantify PII leakage for their models. The models’ weights are known in this case and can boost the prompting effectiveness. With full access to the model, it is possible to train soft prompts that exceed the capabilities of the prompt templates. Hypothetical attackers would not have the necessary information to follow a similar prompt tuning approach and would be limited to less efficient prompt engineering techniques. Presumably, even a clever and motivated attacker would have difficulty devising a better probing strategy than a soft prompt developed by the model owners, so this technique enables model developers to zero in on a worst-case PII leakage estimate.

Inan et al. (2021) introduced another privacy metric based on the concept of *differential privacy*. A computation is considered differentially private when two datasets that differ by exactly one record produce the same outputs with a maximum probabilistic deviation ϵ . Formally this is given by:

$$\Pr[M(A) \in S] \leq e^\epsilon \Pr[M(B) \in S] \quad (6.32)$$

where A and B are datasets that differ by a single record and S is any subset of possible outputs. For an adequately small ϵ , differential privacy can provide a very strong guarantee that a model does not reveal information specific to an individual data point. By definition, the model’s output would be consistent with an alternate version trained on data that does not include that individual.

Furthering this idea, Inan et al. (2021) trained a reference model with all the data found to be unique to any user removed. They then used the perplexity ratio of the reference model and the model being assessed for privacy leakage on each of the removed elements, thus defining a worst-case leakage metric as follows:

$$\epsilon_l = \max_{w \in S_{uniq}} \log\left(\frac{PP_{public}(w)}{PP_{lm}(w)}\right) \quad (6.33)$$

where PP_{lm} is the perplexity of a language model trained with user data and PP_{public} is the perplexity of a public model, over each sequence $w \in S_{uniq}$.

6.4.3 Benchmarks

Miresghallah et al. (2023) proposed *ConfAIde* as a benchmark for assessing whether LLMs can make sound judgments about revealing potentially sensitive information. Instead of analyzing what the model knows, it focuses on its ability to reason about situations in which knowledge should not be shared. Four different types of scenarios are defined, with increasing degrees of complexity. The simplest scenario asks a model to rate the sensitivity for a given piece of information, independent of who accesses it or how it is used. The other three tiers become more complex, with the last tier asking the model to capture notes on an extended conversation. Multiple people join and leave at different times, discussing multiple types of information; the model then must decide which information to share in a summary with the whole group. For each example, human annotations were collected to establish the sensitivity of the information and human preferences for sharing or not sharing the information in the given context.

The authors of ConfAIde used their benchmark to assess several top-performing LLMs, including GPT-4 and ChatGPT. They found that the models were reasonably good at the lowest level task of recognizing sensitive information, but their capabilities decreased significantly when greater contextual awareness was needed. Their results indicated that LLMs had difficulty distinguishing between private and public information, so generally, a model that is less likely to share sensitive data is also more likely to hide non-sensitive data. The ConfAIde benchmark reveals that even instruction-tuned LLMs are incapable of reasoning about privacy and secrecy, and new techniques will likely be needed to address this deficiency.

6.4.4 Mitigation Strategies

In this section, we discuss practical ways to mitigate the privacy issues posed by LLMs. These strategies are divided into methods that can be applied during training and methods applied during inference.

6.4.4.1 Privacy Protection During Training

Perhaps the most intuitively straightforward way to prevent LLMs from distributing personal information is to purge it from the training data. A model obviously won't memorize private data if it never sees it in the first place. This is a widely utilized pre-processing step for LLM pre-training, as mentioned in Chapter 2. Unfortunately, given the massive quantities of data involved, it is virtually impossible to guarantee that all PII has been removed using standard anonymization techniques.

The concept of differential privacy discussed earlier in this section also has utility as a mitigation strategy. Various researchers, such as [Abadi et al. \(2016\)](#), have introduced differential privacy into the training process by building it into the stochastic gradient descent optimizer. While this approach has its merits, it has thus far been shown in most cases to be detrimental to training and usually results in lower-quality models.

A further limitation of training data anonymization and differential privacy is that LLMs have also been shown to infer personal information without explicitly learning it. [Staab et al. \(2023\)](#) found that several state-of-the-art LLMs could accurately discover Reddit users' information based on their posted content. This work sought to identify personal attributes such as age, gender, and location through direct prompting techniques. They sent a user's posts and asked each model if it could guess the information. Even when the input data had been anonymized to remove instances where users explicitly divulged information, they were still frequently successful at guessing correctly. GPT-4 had the highest accuracy on the evaluation dataset curated by the authors, at an impressive 84.6%.

While guessing the approximate age of an unknown Reddit user may seem benign at first glance, these findings are significant because many people who participate in online forums believe that they are anonymous as long as they do not reveal their names. It is well known that the internet makes people feel more comfortable saying things they otherwise would not want to share. However, suppose they divulge a considerable amount of information about where they live, their jobs, their families, and their age. In that case, it becomes possible for a determined individual to piece together someone's identity through social media and publicly available records. This risk already exists without using LLM's, but it is somewhat laborious. LLMs could accelerate this malicious activity and make it easier to conduct at a much larger scale.

It is conceivable that future work will give rise to new techniques that are more successful at preventing models from memorizing PII from their training data. However, it is far more difficult to imagine how we could continually develop increasingly powerful models yet somehow prevent them from acquiring enough knowledge to infer geographical and generational differences in speaking or writing styles. For better or worse, this is a capability that LLM's now possess. It is almost certainly more prudent to focus on putting safeguards around model usage rather than attempting to stunt their intelligence.

6.4.4.2 Privacy Protection During Inference

When new ways to exploit a model are discovered, significant pressure exists to resurface that direct query. Often, when reports surfaced that directly query an LLM can retrieve information that should not have been given, users will soon find that similar prompts stop working. The system could be updated internally, for instance, to include instructions within the context that any requests to determine a person's location should not be carried through. The model can then respond to such queries by simply stating that it is unable to provide an answer. While it is good for model providers to be willing and able to address such issues as quickly as possible, this is a very reactionary approach that falls short of completely alleviating all privacy concerns.

Given the seeming inability of modern LLMs to fully guarantee the protection of private data, it is also vital for application developers to consider how these models could put their users at risk. After all, LLM providers such as OpenAI are known to store queries sent through their APIs to enable future technological advancements. Rather than fully entrusting model researchers and developers with the responsible use of incoming data, the consumers of LLMs must often consider anonymizing their prompts before sending them to a third-party service. This is especially true for any application where users are likely to include personal data intentionally or unintentionally. Tools such as OpaquePrompts have been developed to automate the removal of sensitive information and, depending on the use case, potentially inject the anonymized tokens back into the output downstream of the model's response if needed.

! Practical Tips

Another common alternative for organizations that rely on externally developed models is to choose an open-source LLM instead of a service such as ChatGPT. With this approach, a copy of the model can be deployed internally. Users' prompts remain secure within the organization's network rather than being exposed to third-party services. While this dramatically reduces the risk of leaking sensitive data, it also adds significant complexity.

LLM demands expensive computing resources, and optimizing the cost of all that computation demands specialized human expertise. Beyond the increasingly large number of applications being built on top of third-party LLMs, there is also a strong demand for direct interactions with ChatGPT and its ilk to help with various daily tasks. However, this has also been met with hesitation by many people who are concerned about exposing their private data. A user who wants to use an online LLM to write an email with a more professional tone would necessarily expose the contents of their proposed email to the service providing the model. To avoid this uncomfortable situation, the adoption of smaller models that can run on personal devices has increased rapidly.

! Practical Tips

One of the most popular locally installable LLM interfaces is an application called GPT4All from Nomic AI. It provides options to download different model variants, such as Falcon and Mistral, in sizes under 10 billion parameters. These models are small enough to provide fast, high-quality responses on a personal device, requiring no API requests. Naturally, there are some limitations compared to the more powerful GPT models, especially in cases with large context sizes. However, a smaller LLM can be more than adequate for answering questions or helping with basic tasks. In many cases, it is a reasonable trade-off for substantially reducing privacy risk.

The trend toward locally available models is being closely watched from the perspective of the hardware industry as well. Over the past decade, most hardware advancements have been geared toward more efficient training of models on ever-larger datasets. However, in recent years, there has been a massive wave of investment in edge computing optimized for neural models. Some prognosticators believe that the growth potential for this technology may be even more significant than the astounding revenue growth that NVIDIA has achieved from its large-scale GPUs. While there are other factors, privacy concerns with LLMs undoubtedly contribute to the interest in decentralized models.

6.5 Tutorial: Measuring and Mitigating Bias in LLMs

6.5.1 Overview

In Section 6.2, we discussed the impact of bias in LLMs and some of the techniques developed to mitigate it. In this tutorial, we will apply one of these methods and observe the corresponding shifts in model behavior. This exercise closely follows the work of [Meade et al. \(2022\)](#), who surveyed several bias mitigation techniques and conveniently provided the code to run all their experiments in a GitHub repository.

Goals:

- Analyze how the CrowS benchmark is designed to measure bias.
- Test the use of one potential bias mitigation technique on RoBERTa and evaluate the improvement.
- Apply a debiased model on a downstream task to assess whether its capabilities as a language model are degraded.

Please note that this is a condensed version of the tutorial. The full version is available at <https://github.com/springer-llms-deep-dive/llms-deep-dive-tutorials>.

6.5.2 Experimental Design

In this exercise, we will demonstrate the use of the `bias-bench` library to reduce the appearance of gender bias in a Roberta model. We will then use the CrowS metric to demonstrate the improvement and compare the debiased model’s capabilities to those of the original model on a sentiment analysis task.

The dataset used for the CrowS benchmark consists of pairs of sentences. In each pair, one sentence represents a stereotype while the other replaces the relevant words to contradict the stereotype. For example, “black” may be replaced with “white” if it is a racial stereotype, “woman” may be replaced with “man” if it is a gender stereotype, and so forth. The sentence pairs are otherwise identical apart from these tokens. These data are used to measure the bias of a given LLM and the relative effects of potential bias mitigation techniques.

The algorithm chosen for this experiment is called Sent-Debias. The motivation behind this algorithm is that if a model is utterly neutral about an attribute such as gender, its embeddings of “He was a slow runner” and “She was a slow runner” would generally be very close, if not identical. Variations in these embeddings can be primarily attributed to bias. Sent-Debias captures these variations across many examples and maps them to a lower-dimensional subspace using primary component analysis, resulting in a set of vectors representing the direction of the bias. Once this subspace is learned, it is inserted into the forward pass so that any text representation’s bias projection is subtracted before the final output is returned.

Sent-Debias requires a large and diverse dataset to generate the sentences used in the procedure described above. It has a predefined set of biased words to augment the data, such as “boy” and “girl,” for instance. We use a sample of text from Wikipedia to learn a representation of model biases as reflected in the difference between sentence embeddings with potentially biased tokens substituted.

After applying bias mitigation to a model and evaluating whether gender bias has been reduced from the original version, we then assess its comparative ability to be fine-tuned on a downstream task. SST, a standard sentiment analysis dataset that is part of the GLUE benchmark, is used for this purpose (Socher et al., 2013; Wang et al., 2019).

6.5.3 Results and Analysis

Before we begin the debiasing experiments, we assess the current performance of the `roberta-base` model on CrowS. This metric indicates how likely the model is

Table 6.4: Comparison of model variants on the CrowS and SST benchmarks, highlighting the impact of debiasing.

Model Variant	CrowS	SST
Base RoBERTa	60.15	0.922
Sent-Debias RoBERTa	52.11	0.930

to choose a stereotype when asked to fill in masked tokens in a potentially biased sentence, therefore a lower number is better. It is important to note that an inferior language model could achieve nearly perfect results on this metric since it has not learned the biases in the data well enough to select tokens that reflect stereotypes. It is often the case that weaker LLMs tend to appear less biased than more capable LLMs based on this metric.

```
# Output
-----
Total examples: 261
Metric score: 60.15
```

Listing 6.1: CrowS Metrics for RoBERTa

Next, we use Sent-Debias to compute a gender bias subspace for RoBERTa. Once the bias direction is computed, we recheck the CrowS benchmark to determine whether gender bias has decreased. It is now much closer to 50.0, meaning that the model does not prefer stereotypical tokens as frequently as before.

```
# Output
-----
Total examples: 261
Metric score: 52.11
```

Listing 6.2: Re-evaluating CrowS metrics with debiasing.

As mentioned, our model’s improvement on the CrowS metric may be linked to a decreased overall ability to predict tokens accurately. To make sure that we still have a similarly performing LLM after removing gender bias, we compare the results of fine-tuning the model for sentiment analysis both with and without Sent-Debias. There is slight variation between runs of the training loop, but the accuracy on the SST test data appears to be roughly the same regardless of whether Sent-Debias is applied.

While these results are undoubtedly positive, it is not clear whether we can declare success or whether the debiased LLM recovered some degree of gender bias during the fine-tuning process. It seems likely that the sentiment training data may have been biased, and the effects would not be readily captured by the CrowS metric we employed. We would need to analyze this task more closely to ascertain whether our attempt to mitigate bias succeeded.

6.5.4 Conclusion

In this tutorial we have shown a promising approach to address bias in LLMs, but current techniques still fall short of fully solving this issue. A crucial finding of Meade et al. (2022) was that despite numerous proposed debiasing strategies, none perform consistently well across various models and bias types. In addition, they also found that benchmarks such as CrowsS, StereoSet, and SEAT can be unstable in terms of their performance across multiple runs of the same algorithm. This leaves the question of whether the metrics are robust enough to form a complete bias assessment. Further work in both measuring and mitigating bias will be highly important.

References

- Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS’16. ACM, October 2016. doi: 10.1145/2976749.2978318. URL <http://dx.doi.org/10.1145/2976749.2978318>.
- Abubakar Abid, Maheen Farooqi, and James Zou. Persistent anti-muslim bias in large language models. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, pages 298–306, 2021.
- Jaimeen Ahn and Alice Oh. Mitigating language-dependent ethnic bias in bert. *arXiv preprint arXiv:2109.05704*, 2021.
- Giuseppe Attanasio, Debora Nozza, Dirk Hovy, and Elena Baralis. Entropy-based attention regularization frees unintended bias mitigation from lists. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1105–1119, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.88. URL <https://aclanthology.org/2022.findings-acl.88>.
- Emily M Bender and Alexander Koller. Climbing towards nlu: On meaning, form, and understanding in the age of data. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 5185–5198, 2020.
- Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*, pages 610–623, 2021.
- Camiel J Beukeboom and Christian Burgers. How stereotypes are shared through language: a review and introduction of the aocial categories and stereotypes communication (scsc) framework. *Review of Communication Research*, 7:1–37, 2019.
- Su Lin Blodgett and Brendan O’Connor. Racial disparity in natural language processing: A case study of social media african-american english. *arXiv preprint arXiv:1707.00061*, 2017.

- Rishi Bommasani, Percy Liang, and Tony Lee. Holistic evaluation of language models. *Annals of the New York Academy of Sciences*, 2023.
- Conrad Borchers, Dalia Sara Gala, Benjamin Gilbert, Eduard Oravkin, Wilfried Bounsi, Yuki M Asano, and Hannah Rose Kirk. Looking for a handsome carpenter! debiasing gpt-3 job advertisements. *arXiv preprint arXiv:2205.11374*, 2022.
- Shikha Bordia and Samuel R Bowman. Identifying and reducing gender bias in word-level language models. *arXiv preprint arXiv:1904.03035*, 2019.
- Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- Deng Cai, Yan Wang, Huayang Li, Wai Lam, and Lemao Liu. Neural machine translation with monolingual translation memory. *arXiv preprint arXiv:2105.11269*, 2021.
- Meng Cao, Yue Dong, Jiapeng Wu, and Jackie Chi Kit Cheung. Factual error correction for abstractive summarization models. *arXiv preprint arXiv:2010.08712*, 2020.
- Nicholas Carlini et al. Extracting training data from large language models, 2021.
- Kai-Wei Chang, Vinodkumar Prabhakaran, and Vicente Ordonez. Bias and fairness in natural language processing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP): Tutorial Abstracts*, 2019.
- I Chern, Steffi Chern, Shiqi Chen, Weizhe Yuan, Kehua Feng, Chunting Zhou, Junxian He, Graham Neubig, Pengfei Liu, et al. Factool: Factuality detection in generative ai—a tool augmented framework for multi-task and multi-domain scenarios. *arXiv preprint arXiv:2307.13528*, 2023.
- Aakanksha Chowdhery et al. Palm: Scaling language modeling with pathways, 2022.
- Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. Plug and play language models: A simple approach to controlled text generation. *arXiv preprint arXiv:1912.02164*, 2019.
- Ameet Deshpande, Vishvak Murahari, Tanmay Rajpurohit, Ashwin Kalyan, and Karthik Narasimhan. Toxicity in chatgpt: Analyzing persona-assigned language models. *arXiv preprint arXiv:2304.05335*, 2023.
- Harnoor Dhingra, Preetiha Jayashanker, Sayali Moghe, and Emma Strubell. Queer people are people first: Deconstructing sexual identity stereotypes in large language models. *arXiv preprint arXiv:2307.00101*, 2023.
- Emily Dinan, Angela Fan, Adina Williams, Jack Urbanek, Douwe Kiela, and Jason Weston. Queens are powerful too: Mitigating gender bias in dialogue generation. *arXiv preprint arXiv:1911.03842*, 2019.
- Lucas Dixon, John Li, Jeffrey Sorensen, Nithum Thain, and Lucy Vasserman. Measuring and mitigating unintended bias in text classification. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, pages 67–73, 2018.

- Nouha Dziri, Sivan Milton, Mo Yu, Osmar Zaiane, and Siva Reddy. On the origin of hallucinations in conversational models: Is it the datasets or the models? *arXiv preprint arXiv:2204.07931*, 2022.
- Zahra Fatemi, Chen Xing, Wenhao Liu, and Caiming Xiong. Improving gender fairness of pre-trained language models without catastrophic forgetting. *arXiv preprint arXiv:2110.05367*, 2021.
- Chao Feng, Xinyu Zhang, and Zichu Fei. Knowledge solver: Teaching llms to search for domain knowledge from knowledge graphs. *arXiv preprint arXiv:2309.03118*, 2023.
- Emilio Ferrara. Should chatgpt be biased? challenges and risks of bias in large language models. *arXiv preprint arXiv:2304.03738*, 2023.
- Jessica Ficler and Yoav Goldberg. Controlling linguistic style aspects in neural language generation. *arXiv preprint arXiv:1707.02633*, 2017.
- Isabel O Gallegos, Ryan A Rossi, Joe Barrow, Md Mehrab Tanjim, Sungchul Kim, Franck Dernoncourt, Tong Yu, Ruiyi Zhang, and Nesreen K Ahmed. Bias and fairness in large language models: A survey. *arXiv preprint arXiv:2309.00770*, 2023.
- Leo Gao et al. The pile: An 800gb dataset of diverse text for language modeling, 2020.
- Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. Creating training corpora for nlg micro-planning. In *55th annual meeting of the Association for Computational Linguistics (ACL)*, 2017.
- Aparna Garimella, Rada Mihalcea, and Akhash Amarnath. Demographic-aware language model fine-tuning as a bias mitigation technique. In *Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing*, pages 311–319, 2022.
- Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A Smith. Realtoxicyprompts: Evaluating neural toxic degeneration in language models. *arXiv preprint arXiv:2009.11462*, 2020.
- Sayan Ghosh, Mathieu Chollet, Eugene Laksana, Louis-Philippe Morency, and Stefan Scherer. Affect-lm: A neural language model for customizable affective text generation. *arXiv preprint arXiv:1704.06851*, 2017.
- Michael Gira, Ruisu Zhang, and Kangwook Lee. Debiasing pre-trained language models via efficient fine-tuning. In *Proceedings of the Second Workshop on Language Technology for Equality, Diversity and Inclusion*, pages 59–69, 2022.
- Seraphina Goldfarb-Tarrant, Rebecca Marchant, Ricardo Muñoz Sánchez, Mugdha Pandya, and Adam Lopez. Intrinsic bias metrics do not correlate with application bias. *arXiv preprint arXiv:2012.15859*, 2020.
- Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2023.
- Anthony G Greenwald, Debbie E McGhee, and Jordan LK Schwartz. Measuring individual differences in implicit cognition: the implicit association test. *Journal of personality and social psychology*, 74(6):1464, 1998.

- Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR, 2017.
- Demi Guo, Alexander M Rush, and Yoon Kim. Parameter-efficient transfer learning with diff pruning. *arXiv preprint arXiv:2012.07463*, 2020.
- Wei Guo and Aylin Caliskan. Detecting emergent intersectional biases: Contextualized word embeddings contain a distribution of human-like biases. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, pages 122–133, 2021.
- Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. Don’t stop pretraining: Adapt language models to domains and tasks, 2020.
- Xudong Han, Timothy Baldwin, and Trevor Cohn. Towards equal opportunity fairness through adversarial learning. *arXiv preprint arXiv:2203.06317*, 2022.
- Lukas Hauzenberger, Shahed Masoudian, Deepak Kumar, Markus Schedl, and Navid Rekabsaz. Modular and on-demand bias mitigation with attribute-removal subnetworks. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 6192–6214, 2023.
- Daniel Hershcovich, Stella Frank, Heather Lent, Miryam de Lhoneux, Mostafa Abdou, Stephanie Brandl, Emanuele Bugliarello, Laura Cabello Piqueras, Ilias Chalkidis, Ruixiang Cui, et al. Challenges and strategies in cross-cultural nlp. *arXiv preprint arXiv:2203.10020*, 2022.
- Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.
- Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification, 2018.
- Huseyin A. Inan, Osman Ramadan, Lukas Wutschitz, Daniel Jones, Victor Rühle, James Withers, and Robert Sim. Training data leakage analysis in language models, 2021.
- Nishtha Jain, Maja Popovic, Declan Groves, and Eva Vanmassenhove. Generating gender augmented data for nlp. *arXiv preprint arXiv:2107.05987*, 2021.
- Przemyslaw Joniak and Akiko Aizawa. Gender biases and where to find them: Exploring gender bias in pre-trained transformer-based language models using movement pruning. *arXiv preprint arXiv:2207.02463*, 2022.
- Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, et al. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*, 2022.
- Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*, 2019.
- Siwon Kim, Sangdoo Yun, Hwaran Lee, Martin Gubri, Sungroh Yoon, and Seong Joon Oh. Propile: Probing privacy leakage in large language models, 2023.

- Ben Krause, Akhilesh Deepak Gotmare, Bryan McCann, Nitish Shirish Keskar, Shafiq Joty, Richard Socher, and Nazneen Fatema Rajani. Gedi: Generative discriminator guided sequence generation. *arXiv preprint arXiv:2009.06367*, 2020.
- Julia Kreutzer, Isaac Caswell, Lisa Wang, Ahsan Wahab, Daan van Esch, Nasanbayar Ulzii-Orshikh, Allahsera Tapo, Nishant Subramani, Artem Sokolov, Claytone Sikasote, et al. Quality at a glance: An audit of web-crawled multilingual datasets. *Transactions of the Association for Computational Linguistics*, 10:50–72, 2022.
- Keita Kurita, Anna Belova, and Antonios Anastasopoulos. Towards robust toxic content classification. *arXiv preprint arXiv:1912.06872*, 2019.
- Anne Lauscher, Tobias Lueken, and Goran Glavaš. Sustainable modular debiasing of language models. *arXiv preprint arXiv:2109.03646*, 2021.
- Nayeon Lee, Wei Ping, Peng Xu, Mostofa Patwary, Pascale N Fung, Mohammad Shoeybi, and Bryan Catanzaro. Factuality enhanced language models for open-ended text generation. *Advances in Neural Information Processing Systems*, 35: 34586–34599, 2022.
- Alyssa Lees, Vinh Q Tran, Yi Tay, Jeffrey Sorensen, Jai Gupta, Donald Metzler, and Lucy Vasserman. A new generation of perspective api: Efficient multilingual character-level transformers. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3197–3207, 2022.
- Junyi Li, Xiaoxue Cheng, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. Haluval: A large-scale hallucination evaluation benchmark for large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6449–6464, 2023a.
- Shaobo Li, Xiaoguang Li, Lifeng Shang, Zhenhua Dong, Chengjie Sun, Bingquan Liu, Zhenzhou Ji, Xin Jiang, and Qun Liu. How pre-trained language models capture factual knowledge? a causal-inspired analysis. *arXiv preprint arXiv:2203.16747*, 2022.
- Xingxuan Li, Ruochen Zhao, Yew Ken Chia, Bosheng Ding, Lidong Bing, Shafiq Joty, and Soujanya Poria. Chain of knowledge: A framework for grounding large language models with structured knowledge bases. *arXiv preprint arXiv:2305.13269*, 2023b.
- Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, Allie Del Giorno, Suriya Gunasekar, and Yin Tat Lee. Textbooks are all you need ii: phi-1.5 technical report. *arXiv preprint arXiv:2309.05463*, 2023c.
- Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods. *arXiv preprint arXiv:2109.07958*, 2021.
- Adam Liska, Tomas Kociský, Elena Gribovskaya, Tayfun Terzi, Eren Sezener, Devang Agrawal, D’Autume Cyprien De Masson, Tim Scholtes, Manzil Zaheer, Sannah Young, et al. Streamingqa: A benchmark for adaptation to new knowledge over time in question answering models. In *International Conference on Machine Learning*, pages 13604–13622. PMLR, 2022.
- Haochen Liu, Da Tang, Ji Yang, Xiangyu Zhao, Hui Liu, Jiliang Tang, and Youlong Cheng. Rating distribution calibration for selection bias mitigation in recommendations. In *Proceedings of the ACM Web Conference 2022, WWW ’22*, page 2048–2057, New York, NY, USA, 2022. Association for Computing Ma-

- achinery. ISBN 9781450390965. doi: 10.1145/3485447.3512078. URL <https://doi.org/10.1145/3485447.3512078>.
- Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35, 2023.
- Kaiji Lu, Piotr Mardziel, Fangjing Wu, Preetam Amancharla, and Anupam Datta. Gender bias in neural natural language processing. *Logic, Language, and Security: Essays Dedicated to Andre Scedrov on the Occasion of His 65th Birthday*, pages 189–202, 2020.
- Ziyang Luo, Can Xu, Pu Zhao, Xiubo Geng, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Augmented large language models with parametric knowledge guiding. *arXiv preprint arXiv:2305.04757*, 2023.
- Potsawee Manakul, Adian Liusie, and Mark JF Gales. Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models. *arXiv preprint arXiv:2303.08896*, 2023.
- Justus Mattern, Zhijing Jin, Mrinmaya Sachan, Rada Mihalcea, and Bernhard Schölkopf. Understanding stereotypes in language models: Towards robust measurement and zero-shot debiasing. *arXiv preprint arXiv:2212.10678*, 2022.
- Rowan Hall Maudslay, Hila Gonen, Ryan Cotterell, and Simone Teufel. It’s all in the name: Mitigating gender bias with name-based counterfactual data substitution. *arXiv preprint arXiv:1909.00871*, 2019.
- Chandler May, Alex Wang, Shikha Bordia, Samuel R Bowman, and Rachel Rudinger. On measuring social biases in sentence encoders. *arXiv preprint arXiv:1903.10561*, 2019.
- Nick McKenna, Tianyi Li, Liang Cheng, Mohammad Javad Hosseini, Mark Johnson, and Mark Steedman. Sources of hallucination by large language models on inference tasks. *arXiv preprint arXiv:2305.14552*, 2023.
- Nicholas Meade, Elinor Poole-Dayan, and Siva Reddy. An empirical survey of the effectiveness of debiasing techniques for pre-trained language models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1878–1898, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.132. URL <https://aclanthology.org/2022.acl-long.132>.
- Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, Edouard Grave, Yann LeCun, and Thomas Scialom. Augmented language models: a survey, 2023.
- Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen-tau Yih, Pang Wei Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. Factscore: Fine-grained atomic evaluation of factual precision in long form text generation. *arXiv preprint arXiv:2305.14251*, 2023.
- Niloofar Mireshghallah, Hyunwoo Kim, Xuhui Zhou, Yulia Tsvetkov, Maarten Sap, Reza Shokri, and Yejin Choi. Can llms keep a secret? testing privacy implications of language models via contextual integrity theory. *arXiv preprint arXiv:2310.17884*, 2023.

- Niels Mündler, Jingxuan He, Slobodan Jenko, and Martin Vechev. Self-contradictory hallucinations of large language models: Evaluation, detection and mitigation. *arXiv preprint arXiv:2305.15852*, 2023.
- Moin Nadeem, Anna Bethke, and Siva Reddy. Stereoset: Measuring stereotypical bias in pretrained language models. *arXiv preprint arXiv:2004.09456*, 2020.
- Nikita Nangia, Clara Vania, Rasika Bhalerao, and Samuel R Bowman. Crows-pairs: A challenge dataset for measuring social biases in masked language models. *arXiv preprint arXiv:2010.00133*, 2020.
- Milad Nasr, Nicholas Carlini, Jonathan Hayase, Matthew Jagielski, A. Feder Cooper, Daphne Ippolito, Christopher A. Choquette-Choo, Eric Wallace, Florian Tramèr, and Katherine Lee. Scalable extraction of training data from (production) language models, 2023.
- Roberto Navigli and Simone Paolo Ponzetto. Babelnet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. *Artificial intelligence*, 193:217–250, 2012.
- Roberto Navigli, Simone Conia, and Björn Ross. Biases in large language models: Origins, inventory and discussion. *ACM Journal of Data and Information Quality*, 2023.
- Debora Nozza, Federico Bianchi, Dirk Hovy, et al. Honest: Measuring hurtful sentence completion in language models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2021.
- Ali Omrani, Alireza Salkhordeh Ziabari, Charles Yu, Preni Golazizian, Brendan Kennedy, Mohammad Atari, Heng Ji, and Morteza Dehghani. Social-group-agnostic bias mitigation via the stereotype content model. In *Proc. The 61st Annual Meeting of the Association for Computational Linguistics (ACL2023)*, 2023.
- Hadas Orgad and Yonatan Belinkov. Blind: Bias removal with no demographics. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8801–8821, 2023.
- SunYoung Park, Kyuri Choi, Haeun Yu, and Youngjoong Ko. Never too late to learn: Regularizing gender bias in coreference resolution. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*, pages 15–23, 2023.
- John Pavlopoulos, Jeffrey Sorensen, Lucas Dixon, Nithum Thain, and Ion Androutsopoulos. Toxicity detection: Does context really matter? *arXiv preprint arXiv:2006.00998*, 2020.
- Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. The refinedweb dataset for falcon llm: outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116*, 2023.
- Ethan Perez, Sam Ringer, Kamilé Lukošiūtė, Karina Nguyen, Edwin Chen, Scott Heiner, Craig Pettit, Catherine Olsson, Sandipan Kundu, Saurav Kadavath, et al. Discovering language model behaviors with model-written evaluations. *arXiv preprint arXiv:2212.09251*, 2022.

- Yusu Qian, Urwa Muaz, Ben Zhang, and Jae Won Hyun. Reducing gender bias in word-level language models with a gender-equalizing loss function. *arXiv preprint arXiv:1905.12801*, 2019.
- Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016.
- Leonardo Ranaldi, Elena Sofia Ruzzetti, Davide Venditti, Dario Onorati, and Fabio Massimo Zanzotto. A trip towards fairness: Bias and de-biasing in large language models. *arXiv preprint arXiv:2305.13862*, 2023.
- Ruiyang Ren, Yuhao Wang, Yingqi Qu, Wayne Xin Zhao, Jing Liu, Hao Tian, Hua Wu, Ji-Rong Wen, and Haifeng Wang. Investigating the factual knowledge boundary of large language models with retrieval augmentation, 2023.
- Stephen Robertson, Hugo Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009.
- Yash Savani, Colin White, and Naveen Sundar Govindarajulu. Intra-processing methods for debiasing neural networks, 2020.
- Timo Schick, Sahana Udupa, and Hinrich Schütze. Self-diagnosis and self-debiasing: A proposal for reducing corpus-based bias in nlp, 2021.
- John Schulman. Reinforcement learning from human feedback: progress and challenges. In *Berkley Electrical Engineering and Computer Sciences*. URL: <https://eecs.berkeley.edu/research/colloquium/230419> [accessed 2023-11-15], 2023.
- Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Rich James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. Replug: Retrieval-augmented black-box language models. *arXiv preprint arXiv:2301.12652*, 2023.
- Eric Michael Smith, Melissa Hall, Melanie Kambadur, Eleonora Presani, and Adina Williams. “i’m sorry to hear that”: Finding new biases in language models with a holistic descriptor dataset. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 9180–9211, 2022.
- Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL <https://www.aclweb.org/anthology/D13-1170>.
- Irene Solaiman and Christy Dennison. Process for adapting language models to society (palms) with values-targeted datasets. *Advances in Neural Information Processing Systems*, 34:5861–5873, 2021.
- Robin Staab, Mark Vero, Mislav Balunović, and Martin Vechev. Beyond memorization: Violating privacy via inference with large language models, 2023.
- Tianxiang Sun, Xiaotian Zhang, Zhengfu He, Peng Li, Qinyuan Cheng, Hang Yan, Xiangyang Liu, Yunfan Shao, Qiong Tang, Xingjian Zhao, et al. Moss: Training conversational language models from synthetic data. *arXiv preprint arXiv:2307.15020*, 7, 2023.

- Tony Sun, Kellie Webster, Apu Shah, William Yang Wang, and Melvin Johnson. They, them, theirs: Rewriting with gender-neutral english, 2021.
- Himanshu Thakur, Atishay Jain, Praneetha Vaddamanu, Paul Pu Liang, and Louis-Philippe Morency. Language models get a gender makeover: Mitigating gender bias with few-shot data interventions. *arXiv preprint arXiv:2306.04597*, 2023.
- Ewoenam Kwaku Tokpo and Toon Calders. Text style transfer for bias mitigation using masked language modeling. *arXiv preprint arXiv:2201.08643*, 2022.
- Hugo Touvron et al. Llama 2: Open foundation and fine-tuned chat models, 2023.
- Prasetya Ajie Utama, Nafise Sadat Moosavi, and Iryna Gurevych. Towards debiasing nlu models from unknown biases. *arXiv preprint arXiv:2009.12303*, 2020.
- Pranav Narayanan Venkit, Sanjana Gautam, Ruchi Panchanadikar, Shomir Wilson, et al. Nationality bias in text generation. *arXiv preprint arXiv:2302.02463*, 2023.
- Eric Wallace, Tony Z Zhao, Shi Feng, and Sameer Singh. Concealed data poisoning attacks on nlp models. *arXiv preprint arXiv:2010.12563*, 2020.
- Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding, 2019.
- Xinyi Wang, Sebastian Ruder, and Graham Neubig. Expanding pretrained models to thousands more languages via lexicon-based adaptation. *arXiv preprint arXiv:2203.09435*, 2022.
- Kellie Webster, Xuezhi Wang, Ian Tenney, Alex Beutel, Emily Pitler, Ellie Pavlick, Jilin Chen, Ed Chi, and Slav Petrov. Measuring and reducing gendered correlations in pre-trained models. *arXiv preprint arXiv:2010.06032*, 2020.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.
- Wikipedia Contributors. Who writes wikipedia? https://en.wikipedia.org/wiki/Wikipedia:Who_writes_Wikipedia%3F, 2023. Accessed: 2023-04-01.
- Weiqi Wu, Chengyue Jiang, Yong Jiang, Pengjun Xie, and Kewei Tu. Do plms know and understand ontological knowledge? *arXiv preprint arXiv:2309.05936*, 2023.
- Miao Xiong, Zhiyuan Hu, Xinyang Lu, Yifei Li, Jie Fu, Junxian He, and Bryan Hooi. Can llms express their uncertainty? an empirical evaluation of confidence elicitation in llms. *arXiv preprint arXiv:2306.13063*, 2023.
- Ke Yang, Charles Yu, Yi R Fung, Manling Li, and Heng Ji. Adept: A debiasing prompt framework. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 10780–10788, 2023.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.
- Zhangyue Yin, Qiushi Sun, Qipeng Guo, Jiawen Wu, Xipeng Qiu, and Xuanjing Huang. Do large language models know what they don't know? *arXiv preprint arXiv:2305.18153*, 2023.

- Charles Yu, Sullam Jeoung, Anish Kasi, Pengfei Yu, and Heng Ji. Unlearning bias in language models by partitioning gradients. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 6032–6048, 2023.
- Sina Zarrieß, Henrik Voigt, and Simeon Schüz. Decoding methods in neural language generation: a survey. *Information*, 12(9):355, 2021.
- Abdelrahman Zayed, Goncalo Mordido, Samira Shabanian, and Sarath Chandar. Should we attend more or less? modulating attention for fairness. *arXiv preprint arXiv:2305.13088*, 2023.
- Yuheng Zha, Yichi Yang, Ruichen Li, and Zhiting Hu. Alignscore: Evaluating factual consistency with a unified alignment function. *arXiv preprint arXiv:2305.16739*, 2023.
- Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.
- Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuhui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.
- Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. Siren’s song in the ai ocean: A survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219*, 2023.
- Ruochen Zhao, Xingxuan Li, Shafiq Joty, Chengwei Qin, and Lidong Bing. Verify-and-edit: A knowledge-enhanced chain-of-thought framework. *arXiv preprint arXiv:2305.03268*, 2023.
- Wayne Xin Zhao, Jing Liu, Ruiyang Ren, and Ji-Rong Wen. Dense text retrieval based on pretrained language models: A survey. *arXiv preprint arXiv:2211.14876*, 2022.
- Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. Lima: Less is more for alignment. *arXiv preprint arXiv:2305.11206*, 2023.



Chapter 7

Retrieval-Augmented Generation

Abstract Retrieval-augmented generation (RAG) is a prominent application of conversational LLMs. RAG systems accept a user query and return a response similar to chatbots but source the factual details of their response from static knowledge bases, including documents, structured data tables, and more. In RAG, a small language model is used to embed the user query and compare it against a similarly embedded document corpus to find semantically similar, and thus contextually relevant, text segments. The original query and the retrieved documents are then passed to an instruction-tuned chatbot, which uses the documents to answer the query. In this chapter, we discuss the basic underpinnings of RAG and describe the details that an engineer must consider when building a RAG system. We then do deep dives on many modular enhancements that can be incorporated into a RAG workflow to expand capabilities and safeguard against weaknesses. Next, we discuss several test metrics commonly used for evaluating RAG performance, probing both the accuracy of dense retrieval and the success of chatbots in answering queries. Finally, we present a tutorial for building, enhancing, and evaluating a RAG system using the LlamaIndex codebase.

7.1 Introduction

One of the most appealing applications of auto-generative LLMs is as a general knowledge base that can be queried with natural language and replies with factual responses. As we discussed in Sect. 3.2.3, LLMs memorize large volumes of information within their billions of parameters, and with a well-written prompt (and a little bit of luck), users can sometimes elicit accurate responses. However, the accuracy of a given response may be compromised by inaccurate, incomplete, or absent information in the training corpus or by LLM hallucinations (see Sect. 6.1). As such, the real-world actionability of such information is strongly dependent on the toler-

ance level for inaccuracies. Factual errors in LLM responses may not be tolerable in a given application, such as an educational chatbot, medical diagnoses, or automated customer service agents.

Retrieval-Augmented Generation (RAG) has been developed to mitigate these problems of inaccurate or hallucinatory recall. At the most basic level, the RAG approach uses LLMs to create embedding representations of the text within a database of reliable information, rapidly searches for and locates passages responsive to a given query, and return the information in a form useful to the user. In essence, a RAG system is a QA chatbot that sources information from a fixed database instead of relying on pre-training to memorize factual details. This makes it both more reliable in its returned information and extensible to documents that were not part of the LLM pre-training dataset.

RAG was originally introduced in [Lewis et al. \(2020\)](#). However, since the popularization of ChatGPT and similar high-performing chatbots and the realization of their superior ability to reason in-context, research and innovation in RAG techniques have exploded as researchers and developers have worked to solve and optimize the various functional components of the framework. In this chapter, we summarize the essential points of RAG, discuss a number of improvements developed in the recent literature for extending functionality and improving the performance of RAG systems, and overview approaches for evaluating the performance of a RAG system. We will close with a tutorial where we build a RAG system using the popular Lla-maIndex package ([Liu, 2022](#)) and experiment with a few augmentations.

7.2 Basics of RAG

At its core, a basic RAG system executes the four steps represented graphically in Fig. 7.1:

1. **Indexing:** A series of documents are chunked into text segments, and each segment is transformed into a text embedding with a chosen LLM. These embeddings are placed in a vector index where they can be rapidly compared for semantic similarity against additional vectors.
2. **Querying:** A user enters a query that is answerable based on the content of the documents, and this query is embedded using the same embedding model as was used to build the vector index of documents.
3. **Retrieval:** The transformed query is compared against each embedded segment in the vector index, typically using cosine distance, and the segments are ranked by their similarity to the query. The few top-scoring segments are then extracted in their original text representation. Ideally, the most similar chunks will contain information pertinent to the query.
4. **Generation:** These top segments are packaged in a prompting template as context, along with the original query, and the template is passed to an

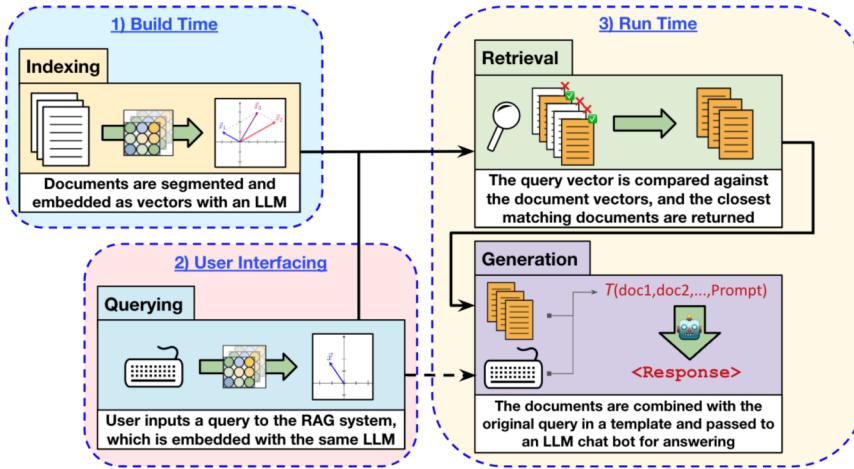


Fig. 7.1: The basic conceptual workflow for a RAG system, including initial document vectorization and indexing, user querying, retrieval, generation, and output. The system locates useful documents within its corpus and passes these documents along with the original query to the generator to create a knowledge-based response to the query.

LLM-based QA agent. The agent then answers the question based on the retrieved context, and the user is given the output.

Fig. 7.2 illustrates a concrete example of the basic RAG cycle. We want a response to the following question:

Who owns the content created by OpenAI programs?

If we ask ChatGPT this question, it responds that:

OpenAI typically retains ownership of the content...

However, this is not true. Instead, let us take the OpenAI terms of service, segment the documents, and create a vector index. When we then query this vector index with the embedded question, we find two chunks specifically detailing ownership rights over outputs from OpenAI services. These documents are placed into a fixed template that includes them as context prior to asking the question. When this templated version is passed to ChatGPT, it now responds:

As per the provided context information, users own any output they rightfully receive from OpenAI services.

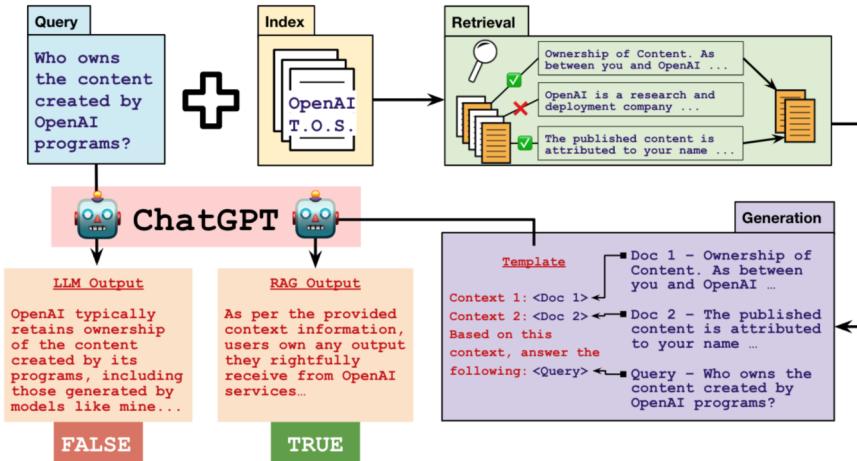


Fig. 7.2: Practical illustration of the RAG workflow, using a question about the ownership of OpenAI output as an example. If we ask ChatGPT the question, we get the wrong answer. Instead, a RAG call with a vector index of the OpenAI terms of service gives the correct answer.

This is the correct answer, which ChatGPT has now been able to report due to our RAG system. (See Sect. 7.6 for full tutorial)

We accomplish a few things by constructing a system that bases its response on a source of information external to the training weights. First, we can use LLM-style semantic reasoning on data that was never part of the original training data. This is critical because LLMs always have cutoff dates for their training set, which prevents them from answering questions about recent events. This is shown in the above example, where the generator bot cannot answer the question about OpenAI's terms of service, which had been updated since the ChatGPT training epoch. Second, because we are passing the relevant context directly to the generator and asking it to answer the query based only on this information, we can increase the accuracy and precision of the response compared to simply trusting our LLMs as knowledge repositories. Many SOTA LLM chatbots have fairly opaque training datasets, and it is not always clear what information they know or how reliably it can be accessed, and they are prone to hallucinate and confidently report things they do not know. In-context reasoning provides more reliable answers than does relying on the correct expression of pre-trained information.

! Practical Tips

As promising as this sounds, several challenges make RAG systems difficult to perfect. Many parameters and approaches control each of the steps listed above, and the model will not work optimally without prudent choices of these configurations. Crucially, you must ensure that the vector search correctly identifies relevant chunks of

text and that you know how to query the generator to extract that information appropriately. Without these optimizations, RAG systems may be as likely to hallucinate as normal LLM calls; to this end, RAG systems must also be taught to admit when they do not know the answer. The cost of failure can be high, as seen in a recent episode in which Air Canada was forced to honor a nonexistent (i.e., hallucinated) policy described to a customer by its AI-based chatbot¹.

7.3 Optimizing RAG

The basic structure of a RAG application is conceptually straightforward, but many details must be worked out. Each of the four steps described in Sect. 7.2 contains a number of preprocessing steps, architectural choices, and hyperparameter settings that must be defined. Prudent choices of these enhancements are essential for optimizing the performance of a RAG system. In this section, we provide an overview of the different procedures implicit in the basic RAG steps and discuss the parameters that should be considered for each step when constructing a RAG system. A tabular summary of these details is given in Table 7.1. First, while building out the vector index, the following considerations apply:

- **Text preprocessing** – This optimization entails the conversion of raw documents into an ideal format. Word documents, HTML pages, PDFs, epubs, and other formats should be converted to plain text. Embedded tables and markdown format should be converted into LLM-friendly input (see below). The text may also be cleaned of extraneous information and normalized to ensure accurate semantic matching. Note that any document normalization must also be performed on the input query at runtime.
- **Text chunking** – This step involves breaking the documents into smaller chunks of similar size for embedding. There are trade-offs to consider when determining the size of each chunk. Smaller chunks allow for more granular information in each vector but may also be too small to capture the relevant context. Smaller chunk sizes also produce more total chunks, thus resulting in slower searches. Chunking can be performed at a fixed character or word length, by sentences, paragraphs, sections, or documents, or by a token length that optimizes the rate of LLM encoding.
- **Metadata** – Chunks can also be augmented with useful metadata, including document titles, authors, subjects, dates, or any other categories that differentiate one document from another. These facts may be used to filter for relevant vectors before the search, or the metadata may be included within each text chunk to add extra context to the vector embeddings.

¹ <https://arstechnica.com/tech-policy/2024/02/air-canada-must-honor-refund-policy-invented-by-airlines-chatbot/>

- **Embedding model** – The choice of the model determines how effectively the RAG system can retrieve chunks responsive to queries. This is a semantic textual similarity NLP task, so models should be chosen appropriately². Larger models will typically produce richer embeddings, while the number of parameters, vector dimensions, and embedding latency determine the expense of computation. The model’s context window size is also relevant as a cap on chunk length. While most of the computational overhead occurs when embedding the documents, this choice also determines what embedding model is used on the query at runtime.
- **Index storage** – Many options exist, with relevant trade-offs including search speed, scalability, static databases vs. expandable databases, open source vs. proprietary, and centralized vs. distributed. Superlinked³ has created and maintained a useful table of vector databases, comparing features and performance.

Each of these steps defines how the documents are handled and stored. Next, we look at how these databases are queried and how the retrieved documents are used for answer generation:

- **Retrieval function** – Similarity between prompt and text chunks is generally determined by cosine distance, but the quantity k of the top documents to return is tunable. A small k provides a shorter context for the generation step, which can improve LLM comprehension but may also leave out relevant information contained in documents with slightly lower scores. A large k passes more information to the generation step but increases the risk of irrelevant information diluting the desired signal.
- **Generation architecture** – Architectural choices for generation include which LLM to use, what prompt template to use for combining query and context, and what system instructions to pass before the query/context portion. Optimal LLMs for the generation step are large, instruction-tuned chat-bots such as Chat-GPT, Claude, or Llama-2. Cost is a significant consideration here, with a trade-off against quality – at the time of writing, GPT-4 API calls cost roughly 50 times GPT-3.5-turbo API calls per token, but provide superior performance in generative tasks.
- **Context formatting** – how to combine potentially disparate top- k documents into a coherent context for the chatbot. Choices include providing all documents in a list, summarizing each with the generator LLM to better fit a context window, or using the generator LLM to consolidate the chunks into a single paragraph of known information.

This overview is not exhaustive but provides a strong starting point for the baseline requirements to consider when creating a RAG system. In the next section, we will detail a number of enhancements that can be added to this picture to increase functionality, improve performance, and broaden the scope

² HuggingFace maintains a leader board benchmarking STS performance against the MTEB dataset (Muenninghoff et al., 2023), useful for RAG applications – <https://huggingface.co/spaces/mteb/leaderboard>

³ <https://superlinked.com/vector-db-comparison>

7.4 Enhancing RAG

In late-2022, ChatGPT and its competitors revolutionized overnight the quality of outputs from the generation step of RAG applications. Perhaps more importantly, their impressive human-like responses to factual questions created broad demand in the market for chatbots that can work with information not included in their training data. These developments opened the door for a flourishing of new RAG applications, and the design of new modules and procedures that can be integrated into the RAG workflow. Gao et al. (2024) have coined the term “modular RAG” to describe systems built with these new innovations. In this section, we discuss a number of these improvements, listing them chronologically in the RAG workflow, from indexing to querying to retrieval and, finally, to generation. We focus particularly on the specific performance issues they are meant to address. Cartoon illustrations of each optimization and enhancement we discuss are shown at their approximate locations in the RAG pipelines in Figs. 7.3-7.6.

7.4.1 Data Sources and Embeddings

The most straightforward improvements to the data indexing stage are standard data sanitation practices, such as input text normalization, stripping extraneous markings like HTML tags, and optimizing segmentation size. However, there are more complex enhancements that can boost model performance and breadth of knowledge. Here, we briefly detail approaches for including structured data tables in a RAG indexing system, and discuss the advantages of fine-tuning the indexing embedding model.

7.4.1.1 Use of Structured Data

Many sources of information that could benefit from RAG-style querying come in formats that are ill-suited for transformation into plain text. These include data tables in documents, SQL databases, knowledge bases, and websites. Data structures are a vital source of factual, numerical, and comparative information that RAG applications must be able to interpret correctly. Here, we review existing approaches for incorporating this information.

One tactic, explored in several works (Hu et al., 2023; Wang et al., 2023d) is to integrate table querying into the retrieval portion of a RAG application. In this approach, a set of documents can be enhanced with, for example, a SQL table containing additional relevant information. Then, a RAG system is equipped with a router (see Sect. 7.4.2.4 below) that determines whether a specific user query would benefit from information in the table. If so, it passes the user query to an LLM trained on SQL code, which generated a fit-to-purpose SQL call. The table is then queried with

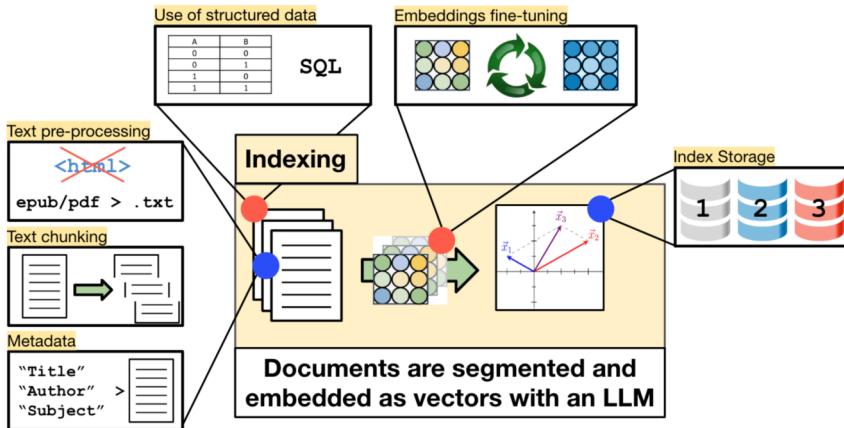


Fig. 7.3: Illustration of the different enhancements discussed for RAG indexing. Pre-processing, chunking, and metadata (Sect. 7.3) operate on the corpus prior to embedding, and can be enhanced with structured data (Sect. 7.4.1.1). Fine-tuning of the embeddings (Sect. 7.4.1.2) and a prudent choice of index storage (Sect. 7.3) can optimize the retrieval accuracy and throughput.

this output. The returned information is then passed along with the query (and any other retrieved documents) to the generator to produce a final response.

As a concrete example, Hu et al. (2023) give the following input/output pair to demonstrate the capabilities of their *ChatDB* system:

```
Question: What was the total revenue for January 2023?

LLM-generated Query:
SELECT SUM(total_price) AS total_revenue
FROM sales
WHERE sale_date >= '2023-01-01' AND sale_date < '2023-02-01';

Database response:
+-----+
| total_revenue |
+-----+
|    707.0     |
+-----+
```

The system converts a plain English request into a precise SQL query designed to return the relevant information, which can serve as the basis for a generated answer.

! Practical Tips

Not all tables come in convenient searchable formats. In particular, when ingesting technical PDFs or similar documents, a RAG system will frequently come across tables containing valuable information. However, it is not obvious how to convert

these tables into a well-suited representation for RAG. Little value can be achieved without a proper structure that retains relationships between table cells and their labels. In response, a number of solutions have been proposed to render PDF tables in a more retriever-friendly format. LlamaParse⁴ is a recent development that uses a proprietary algorithm to parse a diverse array of table shapes to a markdown representation that retains the relationship between table quantities and their row/column labels. These can be integrated with iterative retrieval methods optimized for markdown, which can faithfully extract data relations for generation.

7.4.1.2 Embeddings Fine-tuning

The retrieval accuracy depends on how well the embedding model expresses the critical features of the RAG documents and, thus, how well they can be retrieved. Several open-source embedding models that excel at semantic textual similarity tasks, such as the BGE (Xiao et al., 2023) and VoyageAI (Wang et al., 2023a) series, have been released in recent years; however, given the generality of their training corpora, performance may degrade for subjects with specialized terminology and concepts. This issue can be addressed by fine-tuning the embeddings with domain-specific examples.

A popular approach, implemented in LlamaIndex (Liu, 2022), constructs training examples from the RAG documents themselves. Text chunks from a holdout set are passed to GPT-4, which instructs the creation of individual questions answered by the documents. The embeddings are then fine-tuned so that the retriever selects the correct source document for each generated question. This approach introduces the embedding model to specialized terminology and better adapts the model to bridge the semantic gap between queries and the style of chunking selected for the RAG model. Once the model has been tuned, the documents can be re-embedded, and the RAG application can be constructed. This approach has been shown to improve retrieval accuracy by 5-10%⁵ compared to using base embeddings while improving performance on specific niche topics.

7.4.2 Querying

The central challenge in RAG systems is finding the relevant documents based on a human-written query. However, the wide variation in diction between users and the basic discrepancy between the grammatical and informational content of queries and the documents used to answer them complicate mat-

⁴ <https://www.llamaindex.ai/blog/introducing-llamacloud-and-llamaparse-af8cedf9006b>

⁵ <https://blog.llamaindex.ai/fine-tuning-embeddings-for-rag-with-synthetic-data-e534409a3971>

ters. Querying augmentation generally focuses on transforming human-written queries into a form more likely to match the proper chunks in the vector index. Here, we describe a few approaches for parsing, transforming, and extending to suit the user's needs better.

7.4.2.1 Query Rewriting

As we have discussed elsewhere in this book, LLM responses can be susceptible to the details of the input prompt. Minor variations in diction, grammar, etc., can elicit very different outcomes when passed to an autogenerative model. This is not a desirable outcome for RAG systems, where we want faithful responses to user queries even when they are suboptimally composed. One way to improve these odds is automated *query rewriting*, which transforms human-written queries into a prompt more likely to elicit the desired search results.

We generally do not know *a priori* the most effective form of prompting, so the optimal strategy for query rewriting is to train a rewriter to perform the transformation using reinforcement learning based on RAG pipeline outcomes. In Ma et al.

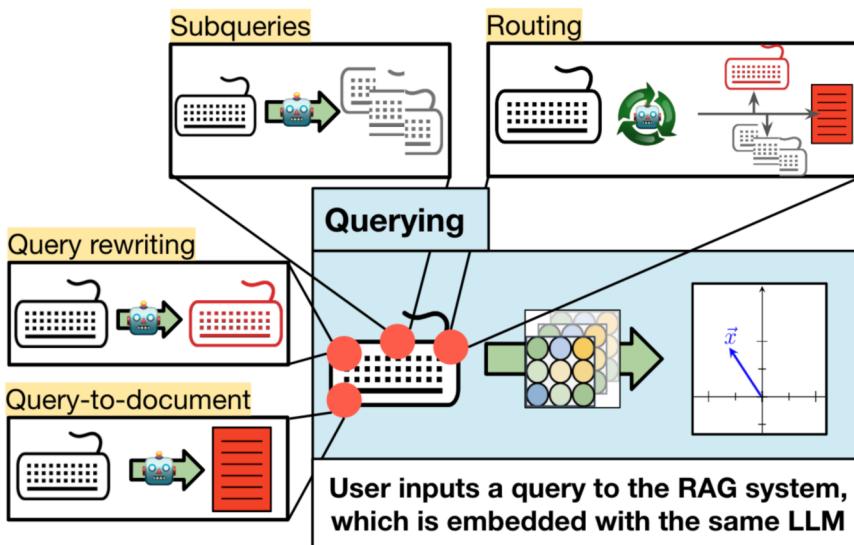


Fig. 7.4: An illustration of the different enhancements discussed for RAG querying. Query rewriting (Sect. 7.4.2.1) and query-to-document expansion (Sect. 7.4.2.2) alter the user prompt using tuned LLMs to increase the likelihood of accurate document retrieval. Subquery generation (Sect. 7.4.2.3) uses an LLM to split complex prompts into component questions that can be queried in the RAG database more easily. Routing (Sect. 7.4.2.4) determines which of these enhancements to apply based on the content of the query.

(2023a), the authors defined a query rewriter using the *T5* model (Raffel et al., 2020), and tuned it using various QA training sets and a reward model based on the accuracy of the generator output. The result is a module sitting between the querying and retrieval stage, which converts the human-written query into an optimized form before embedding. They show improved performance for a trainable rewriter over a static rewriter (i.e., one that was defined but not fine-tuned) and no rewriter at all, demonstrating the value of this approach.

An alternative method for query rewriting was proposed by Raudashch⁶, who developed *RAG-Fusion*. In this approach, an initial query on a database is sent to Chat-GPT, which then rewrites the prompt into several variants. The database is queried with each individual variant, and the output documents for each are merged into a single ranking through reciprocal rank fusion (RRF). In RRF, each document returned by a given search query is assigned a score given by:

$$\text{Score}_{\text{RRF}} = 1/(r + 60), \quad (7.1)$$

where r is that document's place in the search rankings. The scored results are then merged into a single list by summing the scores of any documents present in multiple searches. Thus, the highest scoring document in the RRF merge tends to be highly ranked in multiple search variants. This process is intended to smooth out search variation resulting from word choice and return documents that more robustly address the query in multiple formulations.

7.4.2.2 Query-to-Document Expansion

Basic RAG uses an embedded query to scan a series of vectorized text chunks for the most cosine-similar results in the hope that they contain the specific information that can address the query. One confounding issue in this approach is that, typically, queries are grammatically dissimilar from segments of the chunked documentation. The hope is that if the chunk's subject matter is similar enough to the content of the query, it will produce a good match, but the disparate textual structure can degrade the performance.

Query-to-document expansion seeks to address this issue. In this approach, the user query is passed to an autogenerative model, and the model is asked to create a hypothetical chunk of text within which the answer to the query is found. This chunk is then vectorized with the embedding model and used to search the vector index for semantic similarity. This process is amusingly called generation-augmented retrieval, or GAR (Mao et al., 2021). These generated text chunks frequently contain misinformation as the LLM hallucinates the answer to the query. Nevertheless, it creates a block of text on the queried topic that should be closer in format to the documents we are searching. This generative model can be fine-tuned so that its output more closely resembles the RAG document chunks, or it can use few-shot in-context learning by packaging the query with sample chunks to pick up the salient

⁶ <https://towardsdatascience.com/forget-rag-the-future-is-rag-fusion-1147298d8ad1>

properties of the target documents. Variants of this method have been proposed by [Gao et al. \(2023\)](#) as *HyDE* (Hypothetical Document Encoding), and by [Wang et al. \(2023b\)](#) as *query2doc*. The latter found up to 15% improvement in performance on various dense retrieval tasks when applying their method.

7.4.2.3 Subquery Generation

One weakness of basic RAG is its restriction to semantic similarity matching on a single query. This approach is frequently insufficient for locating all the necessary information to answer more complex queries that require synthesizing multiple pieces of information that could be located in different portions of the RAG corpus. For example, consider a query on a series of reports on the United States Consumer Price Index (CPI), asking about the key drivers of inflation in March from 2020-2023. These values are collectively present in the documents but not in a single chunk or its surrounding context. The values must be extracted from each month's CPI reports. The semantic matching capabilities of a basic RAG system will not be sufficient to retrieve all the necessary information and successfully synthesize it in the generation stage.

Subquery generation was designed to address this shortcoming. This approach uses a sophisticated chatbot (e.g., ChatGPT) to break the original query into a series of prompts that each target a single piece of needed information. Each of these queries follows the retrieval > generation pipeline, and the responses to each are aggregated as context for the original query, which is then passed to a final generation stage. With this paradigm, we can imagine how a subquery engine would ingest and process the CPI request given above:

Input: What were the key drivers of inflation in the month of March from 2020-2023? Subqueries: 1) What were the key drivers of inflation in March 2020? 2) What were the key drivers of inflation in March 2021? 3) What were the key drivers of inflation in March 2022? 4) What were the key drivers of inflation in March 2023?
--

These subqueries will be much more effective at targeting the needed information from the CPI documents and should provide a sampling of the most important drivers of inflation in each of the four months. These four responses can then be synthesized as context for the original query.

7.4.2.4 Routing

We have detailed several pathways that an RAG system might traverse when going from a user query to a retrieval action, including query rewriting, subquery gener-

ation, and the use of knowledge bases separate from our vector index. To take advantage of these capabilities, a RAG system can be designed with multiple options to choose between depending on the content of the query. To handle this decision-making, we can introduce a *routing* system that intakes a query, decides which actions are best suited to seed a quality response, and activates the correct modules. Typically, this decision making is done by a sophisticated autogenerative model with a carefully designed prompt template that instructs the model to consider the query and choose between enumerated options. Conceptual questions to address include whether a query is sufficiently confusing and should be rewritten, whether multiple subprompts are required to retrieval all of the necessary information, or whether the query is about information in associated databases.

7.4.3 Retrieval and Generation

The primary goal of retrieval is to provide the generator with the context necessary to answer the query. However, this goal includes a significant assumption: the text chunk most semantically similar to the query (according to our embedding model) contains the needed information. If this assumption is not met, the RAG call will fail. Retrieval augmentations are concerned with improving the odds that the chosen documents are properly responsive to the user query and rank among the top few most effective additions to a RAG system.

7.4.3.1 Reranking

A common issue with basic RAG is that the text chunks most responsive to a given query often do not appear at the top of the semantic similarity ranking. This retrieval imprecision is partly a result of the relatively small size of typical RAG embedding models. Performance could be improved by using larger and richer embedding models to embed the corpus, but this would be very costly due to the large size of many RAG corpora. A related issue, sometimes called the *lost in the middle* problem (Liu et al., 2023), is that LLMs are more likely to accurately digest in-context information located at the beginning or ends of prompts while being more likely to “lose” information in the middle of the prompt. Without this complication, you could improve performance simply by increasing the quantity of returned documents and hoping to capture the relevant information somewhere in your ranking – *lost in the middle* suggests that this approach will suffer from performance loss.

Reranking was developed as a compromise between these considerations. In reranking, a smaller embedding model is used for initial retrieval, and a large number of the top documents are returned – perhaps 20-30 documents – instead of just a few for basic RAG. These returned documents and the original query are then embedded again with a much larger and more semantically rich model, and the top-k chunks

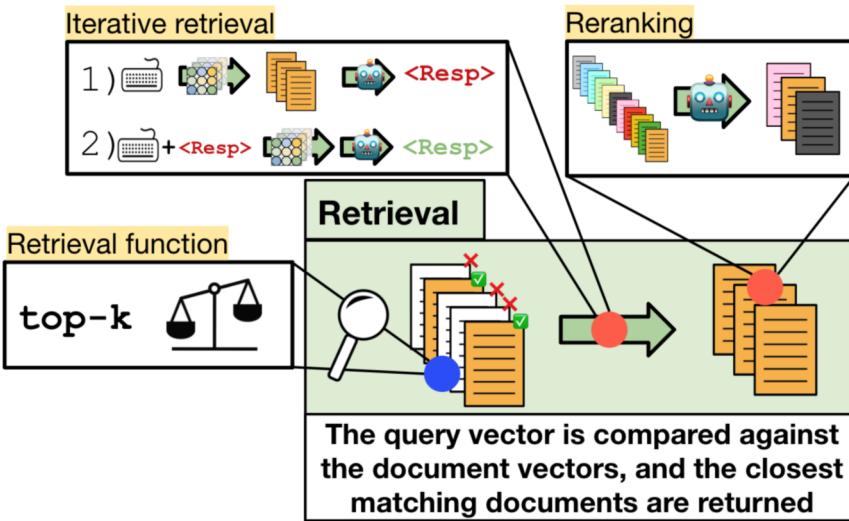


Fig. 7.5: An illustration of the different enhancements discussed for RAG retrieval. The retrieval function (Sect. 7.3) determines how many top documents to collect. Reranking (Sect. 7.4.3.1) uses a second, larger embedding model to rerank the retrieved documents in order to surface the most pertinent information. Iterative retrieval (Sect. 7.4.3.2) uses successive queries and the documents returned from each to answer multi-hop questions.

are reranked according to the new vectors. This allows you to cast a wide net with an inexpensive model and then perform a fine-grained ranking of the results with a superior model, resulting in a far more accurate choice of the top few documents. This will both ensure the generator uses the most relevant documents and pushes the most relevant to the very front of the list to avoid the *lost in the middle* problem. By only using the more expensive model on the returned documents, the higher cost is significantly mitigated while ensuring that relevant documents appear at the top of the ranking. Although the use of embeddings to rerank results is not new, this specific approach in the context of dense retrieval has been advocated by authors such as Ma et al. (2023b) and widely implemented in RAG development software, including LlamaIndex and LangChain (see the tutorial in Sect. 7.6).

7.4.3.2 Iterative Retrieval

One stumbling block that can create failures in RAG querying is questions that require the synthesis of multiple pieces of information. This frequently occurs when a query asks for factual information related to a subject that is not explicitly mentioned but is only implied through a second relationship. An example, given in Shao et al. (2023):

```
Can you get Raclette in YMCA headquarters city?
```

A relevant information database, such as Wikipedia, could tell you that the YMCA is headquartered in Paris and that Raclette is a Swiss dish that is also popular in France. Still, you would have to get lucky to find a single chunk of text explicitly linking the YMCA headquarters to Raclette's availability. A primary RAG generator could answer this question if it was handed a document relating the YMCA to France and another relating France to Raclette. Nevertheless, since the prompt doesn't explicitly mention France, the second piece of information will be missed during retrieval. This style of question is referred to as a "multi-hop question", alluding to the fact that multiple reasoning steps are required for a correct response.

Iterative Retrieval is a process developed to answer multi-hop reasoning queries using dense retrieval. This process follows these steps:

1. A retrieval action is run for a query, and the top-scoring document is returned.
2. The query and document are passed to the generator for a response to the question
3. This response is combined with the original query, embedded with the indexing LLM, and used for another retrieval call against the vector index (generation-augmented retrieval). The top-scoring document is returned.
4. The documents returned in both steps 1) and 3) are combined with the original query and passed to the generator.
5. The generator LLM is asked if the documents and response satisfy the original query. If not, additional alternations of retrieval and generation are done.

! Practical Tips

In the second call to the vector index, the context likely contains hallucinated inaccuracies, but because it is a statement directly responsive to the query, the hope is that it will be semantically similar to the relevant documents in the RAG corpus, which contain the correct information. By allowing multiple calls to the vector index, *iterative retrieval* can answer the different portions of the question step by step. Methodologies along these lines were proposed by both [Shao et al. \(2023\)](#) and [Feng et al. \(2023\)](#).

Using the above example, [Shao et al. \(2023\)](#) shows the following results from their *iterative retrieval* system:

```
Question: Can you get Raclette in YMCA headquarters city?
```

```

*** Iteration 1 ***
-- Retrieved Knowledge --
(1) Title: YMCA Building (Racine, Wisconsin) Context: ... S-
ince the '50s, headquarters of the publishing house and the
bookstore are located in 11 rue de la Montagne Sainte-Genev-
ière 75005 Paris ...

-- Generation --
The YMCA headquarters is located in Paris, France. Raclette
is a traditional Swiss dish, made with melted cheese served
with potatoes, pickles, and other accompaniments. It is not
a traditional French dish, so it is unlikely to be found
in Paris. So the answer is No

```

Note that the generator attempted to answer whether Raclette is available in Paris but hallucinated the wrong conclusion. This generation and the original query are then embedded and used for another round of retrieval:

```

*** Iteration 2 ***
-- Retrieved Knowledge --
(1) Title: Raclette Context: Raclette is a Swiss dish, also
popular in France, based on heating cheese and scraping off
the melted part ...
(2) Title: YMCA Building (Racine, Wisconsin) Context: ... S-
ince the '50s, headquarters of the publishing house and the
bookstore are located in 11 rue de la Montagne Sainte-Genev-
ière 75005 Paris ...

-- Generation --
YMCA headquarters is located in Paris, France. Raclette is
a dish native to parts of Switzerland, but it is also popu-
lar in France. So it is likely that Raclette can be found
in Paris. So the answer is Yes

```

In the first iteration, the RAG call returns information only about the location of the YMCA, and then to answer the query hallucinates an answer about whether Raclette is popular in France. In the second iteration, the generation from iteration 1 is embedded and passed with the query, and because it contains a discussion of whether the dish is available in France, the returned top document also relates to the popularity of Raclette in the region. The final generation uses the retrieved information from both steps, and the correct answer is gleaned from the context.

7.4.3.3 Context Consolidation

Once the documents have been selected, they must be added to a template to pass to the generator. The simplest approach is to concatenate each text chunk together along with the prompt and let the LLM sort out the details. However, this approach has downsides: it will fail if the combined text chunks are longer than the LLM context window size; it may miss crucial information if it is not optimally located (i.e., the *lost in the middle problem* discussed above); and a list of disparate and disconnected

text chunks might be missing the connective tissue that relates their information to one another.

A number of approaches have been suggested for how to better synthesize the information contained in the top returned documents – this process is called *context consolidation*. A common technique is to use LLM calls to summarize the key facts in each text chunk, leading to a shorter context length for the generator (e.g. Chen et al., 2023b). LLMs can also be prompted to build a global summary of the whole corpus of returned documents by looking one-by-one at each chunk and iteratively updating a single summary (e.g. Xu et al., 2023), or by using a tree summarization approach such as the one implemented in LlamaIndex⁷ (e.g. Liu, 2022). Processing the retrieved context from a disconnected series of text snippets into a more coherent and self-consistent document can improve outcomes: across a range of NLP tasks, Xu et al. (2023) showed that prompt compression via summarization both reduced average perplexity (i.e. improved response accuracy) and greatly reduced the length of the input context (reducing the length to as low as 6% in some cases) compared to simply concatenating returned documents in the prompt context.

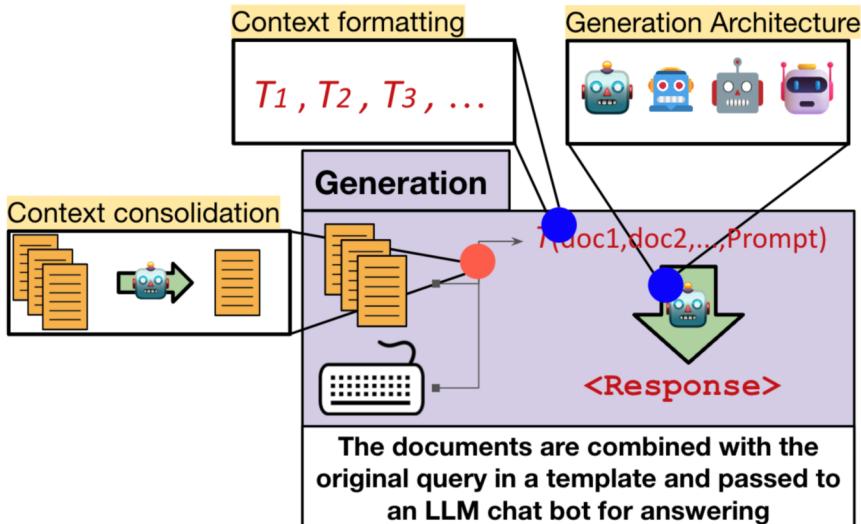


Fig. 7.6: An illustration of the different enhancements discussed for RAG generation. Context consolidation (Sect. 7.4.3.3) comprises methods for distilling the information from multiple retrieved documents into a single document before the generation call. Context formatting (Sect. 7.3) involves choosing an appropriate generation prompt template to suit the needs of the RAG system. Finally, the architecture of the LLM chatbot chosen for generation can be optimized through model selection or even fine-tuning (Sect. 7.3).

⁷ https://docs.llamaindex.ai/en/latest/examples/response_synthesizers/tree_summarize.html

Table 7.1: Summary of retrieval-augmented generation features

Stage	Description	Optimizations	Confounders	Enhancements
Indexing	Converting documents into segments of text, embedding the segments, and storing them in a vector-index.	<ul style="list-style-type: none"> Text normalization Table parsing Chunking strategy Metadata labels Choice of embedding LLM Choice of vector index 	<ul style="list-style-type: none"> Data contained in PDF tables Data contained in structured databases Documents contain technical language unfamiliar to the embedding model 	<ul style="list-style-type: none"> ⇒ Pre-process with LlamaParse ⇒ Automated generation of database search queries ⇒ Fine-tune the embedding model
Querying	User inputs a question that can be answered with information in the documents, and the question is transformed by the embedding model.	Enhanced querying strategies	<ul style="list-style-type: none"> Small variations in query wording can lead to diverse outcomes Low relevance of returned documents Complex queries require multiple pieces of information to answer Different queries require different querying strategies 	<ul style="list-style-type: none"> ⇒ Query rewriting ⇒ Query-to-document ⇒ Subquery generation ⇒ Routing
Retrieval	The vector-index is searched with the embedded query, and the most semantically-similar documents (as determined by cosine-similarity) are returned.	<ul style="list-style-type: none"> Quantity of top documents to return Re-ranking top documents 	<ul style="list-style-type: none"> Most relevant documents not ranking near top Multi-hop questions cannot be answered with one document 	<ul style="list-style-type: none"> ⇒ Reranking ⇒ Iterative retrieval
Generation	The query and the retrieved documents are passed to a chat bot which produces an answer based on the information returned by the search.	<ul style="list-style-type: none"> Choice of generator LLM Generation template and system prompts How to consolidate documents 	<ul style="list-style-type: none"> Generator output not faithful to returned documents Information in certain documents getting "lost in the middle" 	<ul style="list-style-type: none"> ⇒ Change prompting template ⇒ Context consolidation

7.4.4 Summary

We have described many necessary considerations when building a RAG system (7.3), and enumerated useful enhancements (or modules) that can boost RAG performance in key areas. Table 7.1 summarizes these optimizations and enhancements. Considering the enormous variety of RAG components, there is no one-size-fits-all approach to constructing a RAG system. Each enhancement targets a specific RAG weakness for improvement, but not every RAG application will be well suited for each possible enhancement.

The utility of individual RAG modules needs to be considered in the context of a given RAG system’s use case and balanced against the downsides of adding more complexity to your model. These downsides may include greater implementation and testing effort, the addition of more potential points of failure, decreased application latency, and increased computational and financial cost of more calls to LLM agents.

The surest approach is a prudent selection of potential candidate enhancements and a careful trial-and-error process that tests the impact of individual modules on RAG performance. In the next section, we will discuss approaches to performance testing and detail a number of practical considerations required for ensuring your RAG system is performant.

7.5 Evaluating RAG Applications

A key aspect of building a successful RAG solution is an effective evaluation strategy. With the potential for so many modules in more complex RAG applications, there are often multiple evaluation targets to focus on. Similarly, evaluation methods are typically selected according to the application-specific requirements.

In general, the evaluation of RAG leverages preexisting quality and capability measurement metrics applied to either the retriever, generator, or end-to-end output. With multiple evaluation targets, evaluating and leveraging suitable metrics and criteria according to the evaluation target(s) and when evaluation and optimization occur in the application development life-cycle is important. For instance, evaluating the generator component on its own early in the development life-cycle may be a case of premature optimization since generator performance is heavily influenced by the quality of the context provided by the retriever, which may not have been optimized yet. Therefore, sequencing evaluation steps are as critical as selecting suitable metrics and capabilities for measurement (Hoshi et al., 2023).

Indeed, practical guidance provided within the LlamaIndex documentation suggests that an appropriate way to handle this complexity is to begin with an end-to-end evaluation workflow, and as insights into limitations, edge cases, and fail-states are

identified, pivoting to the evaluation and optimization of causal components systematically is a good strategy (Liu, 2022). Assuming an effective evaluation workflow, we will explore the most common evaluation aspects that have emerged for RAG applications.

There are, in essence, seven key aspects commonly leveraged for evaluating RAG applications (Gao et al., 2024). Three can be considered *quality metrics*, and four *system capabilities*:

- Quality metrics
 1. Context relevance
 2. Answer faithfulness
 3. Answer relevance
- System capabilities
 1. Noise robustness
 2. Negative rejection
 3. Information integration
 4. Counterfactual robustness

In the next two sections, we define these aspects, with insights into how and where they are evaluated within a typical RAG framework. Available software tooling and frameworks that enable specific evaluations will also be highlighted where possible.

7.5.1 RAG Quality Metrics

This section describes the context relevance, answer faithfulness, and answer relevance RAG metrics, with a summary illustration shown in Fig. 7.7.

7.5.1.1 Context Relevance

Context relevance measures the effectiveness of the RAG retriever in returning relevant context while passing over irrelevant context. This is typically measured based on a number of preexisting metrics. Some metrics simply look at all retrieved contexts independent of their relevance ranking and are referred to as *rank-agnostic metrics*, while others take context relevance ranking into account and are referred to as *rank-aware metrics*.

Common metrics for measuring context relevance

Recall is a rank-agnostic metric that focuses simply on the number of relevant contexts retrieved and the total number of relevant contexts present within the retrieval

corpus. The recall value is calculated as the proportion or percentage of relevant contexts retrieved relative to the total number of relevant contexts within the retrieval corpus. Since the maximum number of relevant contexts returned is often set within a retrieval setting, a common modification of the recall calculation is $\text{recall}@K$, where K is the fixed number of contexts retrieved.

! Practical Tips

Recall is a good context relevance metric when the rank of returned context is of little impact, such as when short contexts are being used in the generation step or a reranker is being employed downstream (Sect. 7.4.3.1). However, retrieved-context recall may be misleading in this setting when the length of the generator prompt context is susceptible to the *lost in the middle* problem (Liu et al., 2023). Measuring recall in context relevance requires labeled data, typically in the form of query -> relevant document(s) pairs. However, innovations in using highly-capable LLMs to semi-automate recall calculations have been proposed in practical settings. For example, a prompt in the form of ``is all of the required information to answer {query} available in {retrieved_context}'' will allow the LLM to reason over the context conditioned on the query itself.

Precision is another rank-agnostic metric, that measures the proportion or percentage of retrieved documents that are relevant to the query. For example, if the retriever returns 100 contexts, but only 60 of these are relevant to the query, then the precision

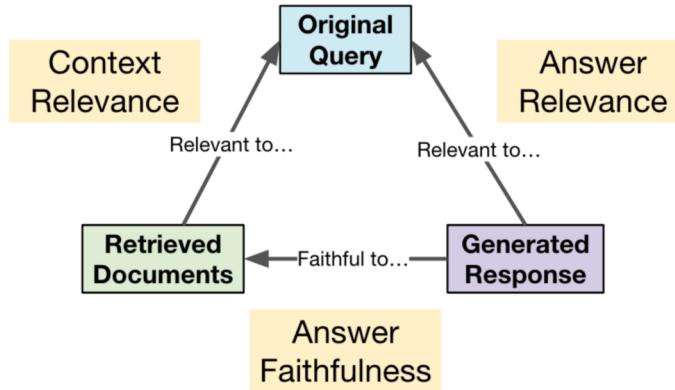


Fig. 7.7: Graphical illustration of the relationships between the three highlighted RAG quality metrics. Context relevance measures how relevant retrieved documents are to the original query, answer relevance measures how relevant the generated response is to the original query, and answer faithfulness measures how faithful the generated response is to the retrieved documents.

will be 0.6 (or 60%). This metric provides insight into how noisy the retriever is, allowing developers to focus on increasing retrieval precision or implementing controls downstream to handle the extra noise in contexts. Such downstream controls include postretrieval reranking (Sect. 7.4.3.1), context consolidation (Fig. 7.6), and simple filter rules. Similar to recall, precision can also be measured as a function of a fixed value for K to give $precision@K$.

! Practical Tips

Traditionally, labeled query/relevant document pairs are used to evaluate retrieved-context precision. However, as in the case of recall, highly capable LLMs are increasingly used for this task, with a prompt of the form:

```
``how many of the search results below are relevant to the
{query}? {retrieved_context}''.
```

As above, the LLM is expected to reason around specific contexts and their relevance to the query, so the extent to which the *lost in the middle* problem impacts this metric calculation should also be carefully evaluated.

Mean Reciprocal Rank (MRR) is a measure of where the most relevant context is within a rank-ordered set of retrieved contexts, on average, across a set of queries. Interpretation of this metric follows the logic that if $MRR = 1$, then for the set of queries evaluated, the most relevant context is always returned by the retriever in the first position for relevance, while $MRR = 0$ indicates that either the retriever is returning no relevant context, or that the most relevant context for each query is always returned in the last position in relevance rank.

! Practical Tips

In practice, MRR typically falls somewhere between these extremes. MRR is particularly useful in measuring retrieval effectiveness in RAG applications where $K = 1$ with respect to the number of retrieved contexts passed to the generator since it is a direct measure of how effectively the retriever is at retrieving the most relevant context in the first position. When used in conjunction with *hit rate* (see below), some of the ambiguity around whether a low MRR is because limited relevant context is being retrieved vs relevant context being retrieved, but with low relevance rank, can be resolved. As an example, for an evaluation with a higher *hit rate* value than *MRR* value is indicative of poor relevance ranking in the retriever, allowing for practical remediation, such as the introduction of a reranker prior to generation.

Hit Rate is a metric that measures the proportion of queries for which the most relevant contexts are retrieved. Practically, this metric is usually limited to measurement

on an application-appropriate value for K retrieved contexts assessed. The simplicity of this metric allows for straightforward interpretation and can enable powerful insights into application inefficiencies when combined with other metrics such as MRR.

Normalized Discounted Cumulative Gain (nDCG) is a rank-aware metric that measures the relevance of retrieved contexts normalized by their retrieval order. Conceptually, nDCG measures a retrieved-context's usefulness (gain) based on its retrieved order. Practically, nDCG is calculated by accumulating gain top to bottom within the retrieved contexts, importantly applying a discount (normalized) to the cumulative gain for highly relevant contexts with low rank order in the list of retrieved contexts. nDCG makes several assumptions, which should be assessed when using the metric and interpreting its meaning with respect to RAG outcomes. In particular, the “discount” portion of the metric calculation assumes that more relevant documents should occur higher in the retrieved context order, thus highly relevant contexts occurring lower in the retrieved context order are penalized through the normalization step. This may not always be an appropriate measure if, for example, all retrieved contexts are used in the generation stage since those high-relevance, low-order contexts will still be provided to the generator.

! Practical Tips

nDCG can be interpreted as a score of how closely the retrieved contexts align to a perfectly ordered list of relevant contexts, where the most pertinent contexts are at the top of the ranked list and relevance declines top to bottom. Thus, nDCG provides clear insights into how well-ranked retrieved contexts are. A low cumulative gain score can indicate the need for better ranking of contexts or the need for better recall in the retriever if few relevant contexts and many irrelevant contexts cause the low score. nDCG is also a helpful metric when evaluating the generator in RAG, where the relevance ranking is simply evaluated on a set of possible responses to a given query rather than retrieved contexts.

7.5.1.2 Answer Faithfulness

As discussed in Sect. 6.1, hallucination is a common weakness of LLMs. This property of even the most advanced LLMs significantly increases the complexity of leveraging them in production settings where honesty is critically important (Sect. 5.1.2). Indeed, RAG is largely an innovation on top of LLMs that attempts to ameliorate this challenge. However, as with all systems leveraging probabilistic mechanisms for generating outputs, how well such innovations ameliorate the hallucination issue needs to be evaluated during development and on an ongoing basis as the application evolves through its development life cycle.

Answer faithfulness, also sometimes called *groundedness*, is a quality measure used to ensure that the responses from the generator are as factually accurate as possible, given the retrieved context. Essentially, this measures how well the generator can ground its responses in the knowledge the retriever provides. Answer faithfulness can be evaluated using several metrics that fall into two categories, *lexical-based* and *model-based*.

Lexical-based metrics

Lexical-based metrics for answer faithfulness aim to measure the extent to which the tokens in the response overlap with or are grounded in the retrieved context or knowledge provided to the generator. In practice, these approaches have been superseded by the model-based metrics discussed later, but we include them here for completeness. Some of the most commonly used lexical-based metrics are as follows:

- **Knowledge-F1** or K-F1 measures the F1 overlap in tokens within the generated answer and the retrieved context provided to the generator. This metric has limitations where the span length from the generator differs significantly from the retrieved context. For example, consider a generator that returns short, concise answers to user queries, while the retrieved context is typically longer than the answer span. In such instances, the response may be penalized on recall, even though the precision of the response tokens is high and the response is otherwise correct with respect to the user’s needs (Adlakha et al., 2023).
- **Knowledge-precision** or K-precision, measures the proportion of tokens found in the generator’s response that are also present in the retrieved context. This metric provides a valuable measure of answer faithfulness without the potential bias that the asymmetry of the measurement introduces in K-F1. This asymmetry arises because the factually relevant tokens within a generator response can only be evaluated as equal to or a subset of the facts present within the retrieved context.

Model-based metrics

As mentioned, lexical-based evaluation metrics for answer faithfulness have been largely superseded by model-based approaches in practice. This is due to the difficulty in generating labeled contexts through annotation and the low correlation that some of these metrics have with human-level judgment (Adlakha et al., 2023), but perhaps more significantly, the ever-improving competency of LLMs for such tasks. While not yet a panacea (e.g., Wang et al. (2023c)), the most capable LLMs have been shown to provide excellent correlation in the evaluation of answer faithfulness with human-based judgment approaches to the same evaluation task (Adlakha et al., 2023). This correlation lends promise to using highly capable LLMs to improve the efficiency of evaluating answer faithfulness in RAG.

One of the earliest model-based approaches for evaluating answer faithfulness was Q^2 (Honovich et al., 2021).

Calculation of this metric begins first with extracting informative spans in the answer. This is typically done using some form of Named Entity Recognition

(NER). As an example, consider the following hypothetical answer response from a RAG system:

"Red wine is acidic"

Using an NER system to extract informative spans mapping to named entities and noun phrases, the informative span Red wine is derived. These informative spans and the responses from which they were identified are then passed to a large generative model to generate relevant question(s) conditioned on the informative span and the original response. In the example above, given Red wine as the informative span and the response above, a generated question could be:

"What is acidic?"

These generated questions are then answered using a fine-tuned QA model.

The answer faithfulness of the original generative system is similar to the informative spans extracted from the original response and the answers to the generated questions. If informative spans are extracted from the original response and the answers to the generated questions are a perfect match, a Q^2 score of 1 is given. If there is no perfect match, then similarity in the informative span from the response and the answer to the generated question is determined using natural language inference (NLI). In this NLI step, entailment receives a score of 1, while contradictions receive score of 0. QA responses with no answer take on a token-level F1 score. The overall system-level Q^2 score is then the average across all answer pairs (Honovich et al., 2021).

More recently, however, model-based approaches have changed to capitalize on the evermore sophisticated generative LLMs available to provide more consolidated measures of answer faithfulness (i.e., the need to have distinct models for question generation, NER, and question answering as in Honovich et al. (2021) is significantly decreased when using only GPT-4, for example). The general approach is very similar to that described for Q^2 , if much less modular since GPT-4 is more capable of leveraging its natural language understanding to complete the task more comprehensively.

Introduced in Adlakha et al. (2023), LLMCritic leverages a simple prompting approach to enable GPT-4 to evaluate whether the response answer from a RAG system contains only information/fact-claims that are either present within or can be inferred from the retrieved context. An example prompt template for this task given by these authors is shown below:

System prompt: You are CompareGPT, a machine to verify the groundedness of predictions. Answer with only yes/no.

You are given a question, the corresponding evidence and a prediction from a model. Compare the "Prediction" and the "

```

Evidence" to determine whether all the information of the prediction is present in the evidence or can be inferred from the evidence. You must answer "no" if there are any specific details in the prediction that are not mentioned in the evidence or cannot be inferred from the evidence.

Question: {Question}
Prediction: {Model response}
Evidence: {Reference passage}

CompareGPT response:

```

Listing 7.1: "Example prompt for assessing Answer Faithfulness"

Here, GPT-4 is prompted to verify whether all of the information within the RAG answer is present or can be inferred from the evidence (retrieved context). In general, when using LLMs to evaluate answer faithfulness, the formula for calculating the metric is:

$$\text{Faithfulness} = \frac{|\# \text{ of facts in answer that can be inferred from retrieved context}|}{|\text{Total } \# \text{ of facts in answer}|} \quad (7.2)$$

As the capabilities of LLMs have increased, their use for the calculation of these kinds of RAG evaluation metrics has increased. Indeed, RAG evaluation frameworks/tools such as LlamaIndex (Liu, 2022), TruLens (Reini et al., 2024) and Ragas (Es et al., 2023) have various methods and approaches like this available for use, some of which we will see in action in the tutorial section.

7.5.1.3 Answer Relevance

Answer relevance is another important metric for evaluating the quality of a RAG system. It answers the question “how relevant is the answer generated to the user query and the retrieved context?”. The most common approaches to calculating this metric also leverage highly capable LLMs. In this instance, the LLM is prompted with the generator’s answer, the context used in generating that answer and instructions to generate N synthetic questions based on this information. These questions are then semantically compared to the original user-query, which results in the reference answer used to generate the synthetic questions. Answer relevancy is measured as the mean “semantic similarity” between the original user query and N synthetic questions. As such, RAG answers that prompt the generation of questions that are most semantically aligned to the original user query will result in higher answer relevancy scores, and *vice versa*. More specifically, the Ragas framework calculates this metric as:

$$\text{Answer Relevance} = \frac{1}{N} \sum_{i=1}^N \text{sim}(E_{g_i}, E_o) \quad (7.3)$$

where N is the number of synthetic questions generated by the evaluation LLM, E_g is the embedding of the i^{th} synthetic/generated question, E_o is the embedding of the original query, and sim is an appropriate measure of similarity between the two (e.g., cosine similarity).

While there are various approaches for evaluating answer relevance, including comparisons to ground-truth answers, etc., the LLM evaluator approaches are becoming dominant because of their ability to overcome the often costly and complex task of defining ground truth for such expressive applications.

7.5.2 Evaluation of RAG System Capabilities

In addition to deriving direct quantitative metrics to evaluate the performance of RAG systems in terms of retrieval quality, answer faithfulness, and relevance, several additional capabilities must be evaluated. The capabilities are evaluated to assess the general performance, versatility, and reliability of the LLM generator used within the RAG system (Chen et al., 2023a). This evaluation approach is necessary to understand how the retrieved context and the prompt augmentation impact the generation process within the LLM. As an example, it is essential to understand the extent to which the LLM generator integrates nonparametric knowledge – the contextual information in the prompt – over its parametric knowledge – the information embedded in the generator LLM’s parameters during pre-training. If the generator overrides its parametric knowledge with the nonparametric retrieved context and said context is erroneous, then as a developer, we understand that we may need to modify our prompt or improve our retrieval precision to ensure that the generated responses are as faithful to ground truth as possible.

In the following sections, we will detail the four most popular and commonly used RAG capability evaluations, with illustrated examples taken from Chen et al. (2023a). A graphical summary of the metrics is shown in Fig. 7.12.

7.5.2.1 Noise Robustness

In simple terms, *noise robustness* measures the LLM generator’s ability to leverage only the useful information within noisy retrieved-context documents. Fig. 7.8 illustrates this property of a RAG system. Effectively, the aim is to understand how well the LLM generator can navigate irrelevant context and still respond with the correct answer to the user’s query.

! Practical Tips

Assessing the RAG LLM’s ability to handle noisy contexts relies on ground-truth knowledge of positive and negative contexts relative to a set of generated question-answer pairs. The typical approach is to pair a relevant document with a random

Noise Robustness

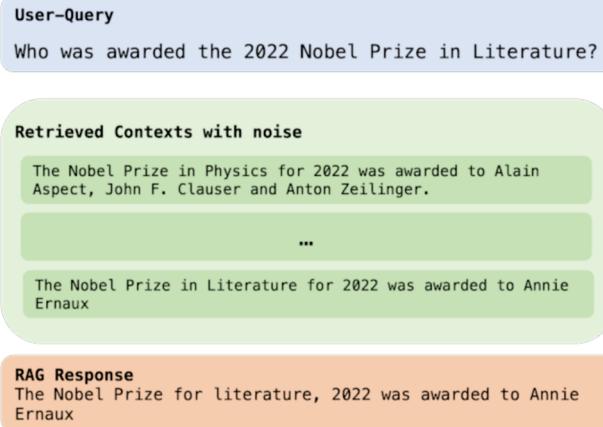


Fig. 7.8: An example of *noise robustness* in a RAG response. Here we can see that even though the retrieved contexts contain noise (e.g., information about the Nobel Prize in Physics rather than literature), the generator can still respond with the correct answer to the user query.

sampling of negative contexts at an appropriate ratio. These “retrieved” contexts can then be passed to the LLM generator, and its answer can be compared to the original answer generated as part of a question-supporting information-answer triplet dataset. An accuracy-based metric such as exact match (EM) calculates the LLM’s ability in this aspect (Chen et al., 2023a).

7.5.2.2 Negative Rejection

In *negative rejection*, the RAG application refuses to answer a given user query in the instance where none of the retrieved contexts contain the relevant information necessary to do so. In Fig. 7.9, we can see that none of the contexts shown contain the relevant facts to answer the question Who was awarded the 2022 Nobel Prize in Literature?. Only contexts relevant to the Nobel Prize in Physics were retrieved. Evaluation of this capability in RAG enables developers to optimize application behavior in the event that the available knowledge sources do not allow faithful or factual responses, such as implementing more stringent system instructions for such settings.

! Practical Tips

Negative Rejection

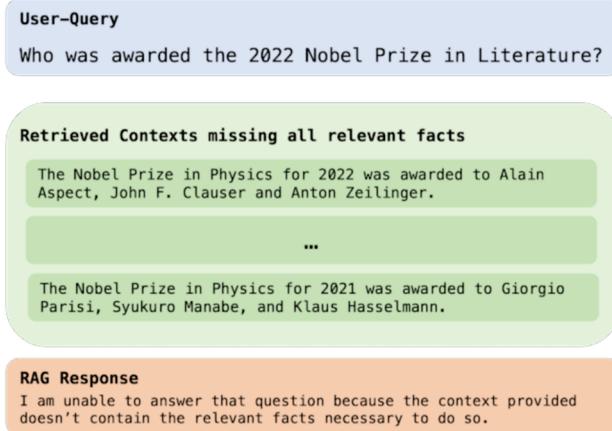


Fig. 7.9: A RAG generator is considered capable of negative rejection in instances where it does not provide an answer to the user query when the necessary information is not provided in any retrieved context. As illustrated, none of the displayed contexts contain information on who the recipient of the 2021 Nobel Prize for Literature was. Therefore, the generator does not answer this question.

To assess negative rejection in practice, again, a set of question-answer pairs is generated, along with supporting information for arriving at the answer. By only sampling contexts from negative documents (i.e., those not generated along with the generated question-answer pairs), the RAG LLM’s negative rejection ability can be measured as a function of the number of times the model correctly answers with the rejection-specific content, which is generally specified through instructions to the LLM. The rejection rate metric can be calculated to understand what proportion of correct vs incorrect negative rejections occur when querying the RAG system.

7.5.2.3 Information Integration

Information integration in RAG refers to the LLM generator’s ability to integrate information across multiple context documents to synthesize a correct answer to a complex question. Suppose the user query contains more than one subquestion or multiple pieces of information distributed across contexts. In that case, it is important that the RAG system systematically integrate the relevant context for each constituent information unit to be provided in the response. As we see in Fig. 7.10, the user query asks for both the 2021 and 2022 winners of the Nobel Prize in Physics. The external

information required to answer this question is also distributed across two separate context documents. The RAG response correctly integrates these contexts to provide a correct response.

Information Integration

User-Query
Who was awarded the 2022 Nobel Prize in Literature and the 2021 Nobel Prize in Physics?

Retrieved Contexts contain all relevant facts

The Nobel Prize in Literature for 2022 was awarded to Annie Ernaux

...

The Nobel Prize in Physics for 2021 was awarded to Giorgio Parisi, Syukuro Manabe, and Klaus Hasselmann.

RAG Response

Annie Ernaux won the 2022 Nobel prize in literature, and Giorgio Parisi, Syukuro Manabe, and Klaus Hasselmann won the 2022 Nobel Prize in Physics.

Fig. 7.10: A RAG generator is said to successfully integrate information, demonstrating the ability to leverage information from multiple contexts/documents to answer complex questions. In this example, the user query asks for both the 2021 and 2022 winners of the Nobel Prize in Physics and successfully integrates the relevant context to provide the correct answer.

! Practical Tips

Again, the evaluation of information integration in practice relies on the generation of question-supporting information-answer triplets. However, an additional step in the test data generation is carried out to create additional aspects to the question's answer, such as combining two questions, their answers, and supporting information, such that the supporting information required to answer the more complex question is distributed across more than one context document. Successful information integration is also determined using an accuracy metric such as EM, where the RAG-generated response is directly compared to the originally generated answer to the question(s) (Chen et al., 2023a).

7.5.2.4 Counterfactual Robustness

Factual errors are common in external knowledge bases commonly used in RAG applications. As such, it is important to evaluate the ability of the RAG generator to identify these falsehoods in retrieved contexts – this is called *counterfactual robustness*. Since identifying errors in the retrieved context relies entirely on the LLM generator’s parametric knowledge, this aspect of the RAG application can be challenging to evaluate where knowledge within the application domain is either not represented or underrepresented in the chosen LLM. While domain adapting or fine-tuning LLMs is always an option, it is expensive and ultimately undercuts some of the advantages of RAG. However, many domain-fine-tuned LLMs have emerged in the open-source space, and as such, generating domain-relevant test data for this purpose is becoming increasingly viable.

Counterfactual Robustness

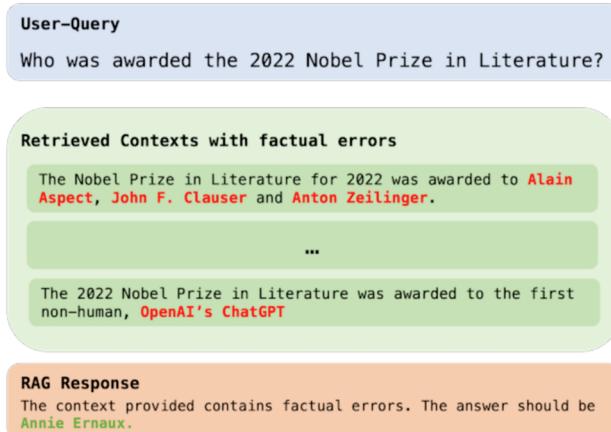


Fig. 7.11: Counterfactual robustness is the generator’s ability to detect and highlight in its response that the context provided contains factual errors. This ability is grounded in the generator LLM’s parametric knowledge, which can mean that it is challenging to assess when using an LLM without domain-relevant knowledge for a given application, or when the application relies on knowledge that arose after the LLM’s knowledge cutoff.

! Practical Tips

To test this capability, the generator LLM is prompted to generate questions and answers solely on its parametric knowledge. This means that the LLM is prompted to generate questions to which it already knows the answers, independent of the in-

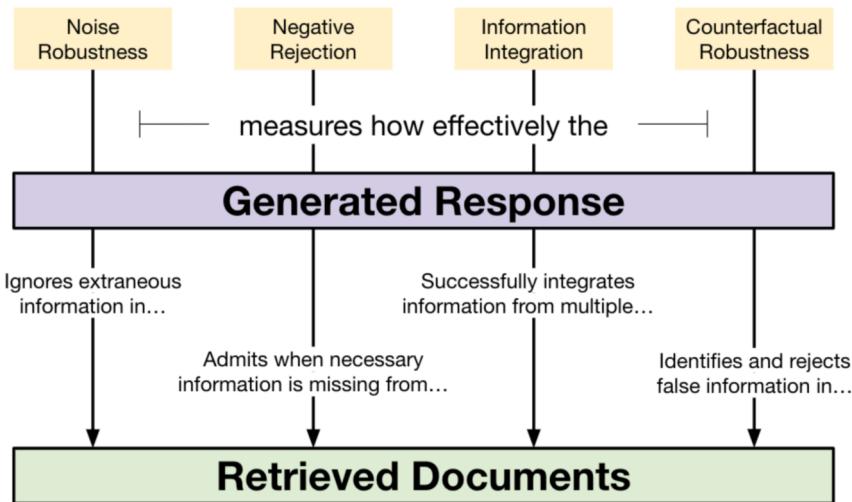


Fig. 7.12: Graphical illustration of the properties measured by each RAG system capability metric. Each of the four metrics determines how well the generated response understands and correctly responds to the properties (positive or negative) of the retrieved documents.

formation in the external knowledge base. To establish the counterfactual supporting information, the generated answers are first verified to be correct, and subsequently, supporting information is retrieved from external knowledge. Supporting contexts are then manually modified to replace correct supporting facts with incorrect supporting facts. This dataset assesses two key metrics: the *error detection rate* and the *error correction rate*.

The error detection rate relies on the LLM generator responding with specified content in the event that supporting contexts contain factual errors. This metric indicates how well the LLM can evaluate the factuality of the retrieved contexts against its parametric knowledge. Similarly, the error correction rate measures how frequently the LLM generator can provide the correct answer despite the supporting information containing errors.

7.5.3 Summarizing RAG Evaluation

The evaluation of RAG applications is complex, thanks to the multiple functional components with fuzzy abstractions. Ensuring that the vector database's indexing is optimal, the chunking used for external knowledge documents, the retrieval relevance

and ordering, and the final generation step are all concerns of the RAG application developer. Thankfully, much research and innovation has occurred to simplify and streamline this complex process. From conceptually useful frameworks, such as the RAG Triad from the TruLens team, to practically efficient implementation frameworks such as LlamaIndex, much of this complexity is simplified for users of these tools and frameworks to enable rapid prototyping and robust production-grade development.

In Chapter 8, we will explore the operational concepts, frameworks, tools, and challenges of using LLMs in production, much of which will apply to RAG application development. However, before we explore these issues in depth, we present a tutorial of RAG development and evaluation.

7.6 Tutorial: Building Your Own Retrieval-Augmented Generation System

7.6.1 Overview

Retrieval-augmented generation is a promising avenue for combining the semantic understanding capabilities of LLMs with the factual accuracy of direct source materials. In this chapter, we have discussed many aspects of these systems, from basic approaches to optimization to enhancements to the core RAG functionality and methods for evaluating RAG performance. This section will present a practical, hands-on example of building and augmenting a RAG system using a popular open-source RAG application.

In recent years, a number of such open-source libraries have been developed and released. These RAG libraries present high-level functions for implementing many different RAG approaches and allow great customization for constructing one's own system. For this tutorial, we will use LlamaIndex ([Liu, 2022](#)) to build a RAG system, experiment with a few of the many tunable parameters, and evaluate the system's performance.

Goals:

- Demonstrate how to set up a basic RAG application with low effort using LlamaIndex.
- Explore the wide range of possibilities for customizing and improving a RAG application.
- Evaluate context relevance, answer relevance, and groundedness for a RAG application.

Please note that this is a condensed version of the tutorial. The full version is available at <https://github.com/springer-llms-deep-dive/llms-deep-dive-tutorials>.

7.6.2 Experimental Design

This exercise walks through the steps to build an experimental RAG application. For our document corpus, we use the OpenAI terms and policies, taken from <https://openai.com/policies>, as they appeared in late January 2024. The tools we incorporate in our application are as follows:

- **LlamaIndex:** This framework handles document parsing, indexing, searching, and generation using an extensive catalog of modules that can be easily incorporated into a single RAG framework. Integrations with Hugging Face allow for great customization in the choice of embedding and generation models. ([Liu, 2022](#))
- **BAAI/bge-small-en-v1.5:** This small English variant of the Beijing Academy of Artificial Intelligence (BAAI) line of text-embedding models is highly performant in text-similarity tasks, yet is small enough (33.4M parameters) to fine-tune easily.
- **OpenAI ChatGPT:** Throughout the tutorial, we use the gpt-3.5-turbo and gpt-4 models from OpenAI as our generators. They will also provide a comparison of the output of our RAG systems.

The first step is to load each document from the OpenAI terms and conditions into LlamaIndex. Next, we choose a chunking strategy and an embedding model to generate our vector index. After this process is finished, LlamaIndex makes it easy to begin querying the RAG database using gpt-3.5-turbo as the generator LLM.

Starting from our initial basic application, we then go on to explore many of the design choices and improvements that can be made. Of these enhancements, the most notable are fine-tuning the embedding model and adding a document reranker. We continually compare results to see how our application responds as we introduce new ideas. Finally, we conduct a thorough evaluation of our end-stage RAG application against an earlier iteration without a fine-tuned embedding model.

7.6.3 Results and Analysis

There are many different approaches to evaluation, but we will consider here only the three quality metrics given in Section [7.5.1](#):

They are:

1. Context Relevance: Is the retrieved context relevant to the query?
2. Answer Relevance: Is the generated answer relevant to the query?

3. Answer Faithfulness/Groundedness: Is the generated answer supported by the retrieved context?

These three metrics ensure that the query finds useful documents, that the generated response is faithful to the documents, and that the generated response answers the original query.

Looking first at context relevance, we construct a test set by generation questions from a large number of corpus chunks. We can then use RAG to retrieve documents from each generated question, and check whether the document it was derived from is selected. We measure this with two metrics: “mrr”, or mean reciprocal rank (= 1 divided by the rank of the correct document), and “hit_rate”, which is equal to 1 if the correct document is among the top 5 returned, or 0 otherwise (see Sect. 7.5.1.1). Looking at one example question:

```
Total MRR = 1.0 / 1
# Hits = 1.0 / 1
Expected ID = ['f712129a-a58d-4e36-b62f-22ebfed56a8']
Retrieved IDs = ['f712129a-a58d-4e36-b62f-22ebfed56a8',
                  '1f95b363-1002-4f2d-bf17-ab4842714072']
```

Listing 7.2: ”Context relevance example”

We can see that the expected document ID was first in the retrieval list, and thus MRR and # hits are both 1/1. Looking now to a sample of 50 validation QA pairs:

```
- Base model:
- Total MRR = 36.5 / 50
- # Hits = 42.0 / 50
- Fine-tuned model:
- Total MRR = 40.0 / 50
- # Hits = 46.0 / 50
```

Listing 7.3: ”Context relevance scores”

We see that the source document was returned in the majority of cases and was frequently (although not always) the top returned document, but the RAG system whose embedding model was previously fine-tuned on the OpenAI terms and conditions corpus does somewhat better.

Turning to answer relevance, we can ask whether the RAG pipeline produces a reasonable answer to our queries. Here, we submit a query, receive a response, and then ask GPT-4 if the query is responsive to the question. In this case, we obtain a simple True or False response. Here is a test case:

```
query = "How can individuals request corrections for factually
        inaccurate information about themselves in ChatGPT output?"
results = run_answer_relevance_eval(index, [query,])

Response:
Individuals can request corrections for factually inaccurate
information about themselves in ChatGPT output by submitting
a correction request through privacy.openai.com or by sending
an email to dsar@openai.com. If the inaccuracy cannot be
```

corrected due to the technical complexity of the models, they can request the removal of their Personal Information from 'ChatGPTs output by filling out a specific form.

Relevant:
True

Listing 7.4: "Answer relevance example"

This response does indeed answer the query. Evaluating 50 samples from the validation set, we find:

- Base model: 47 / 50
- Fine-tuned model: 49 / 50

Listing 7.5: "Answer relevance scores"

Once again, we see a slight improvement from fine-tuning, this time in arguably the most important metric: responsiveness of the query to the question.

The final evaluation metric is answer faithfulness, or "groundedness", where we ensure that the generated responses are grounded in the context. For our models, the transformation from context to response is done by GPT-4 instead of our vector index, so we should expect good performance and little difference between the two models. As expected, both models perform well, with only a minor difference:

- Base model: 48 / 50
- Fine-tuned model: 49 / 50

Listing 7.6: "Groundedness scores"

A summary of our results is given in Table 7.2, along with two additional model configurations – the base and fine-tuned versions combined with reranking (return top 20 > reranked top 2). Reranking significantly boosts context relevance, increasing the number of captured hits to nearly 100% while marginally improving total MRR score. However reranking has actually decreased the metrics for answer relevance and groundedness. Why is unclear, but suggests that care must be taken when incorporating reranking modules – their utility must be validated and not just taken as granted.

Table 7.2: Summary of evaluation results (out of 50) on the TruLens triad of RAG evaluations for four model setups: base, fine-tuned, base + reranking, fine-tuned + reranking.

Model	Context Relevance		Answer Relevance Groundedness	
	MRR	# Hits		
Base	36.5	42.0	47	48
FT	40.0	46.0	49	49
Base RR	37.5	49.0	43	47
FT RR	40.7	49.0	47	48

7.6.4 Conclusion

In this exercise, we discussed the basic steps of setting up a minimally functional RAG application. We then test more advanced methods to improve on the results, and demonstrated how to appropriately evaluate the responses to ensure that the application works as intended. It is clear that tools such as LlamaIndex are extraordinarily powerful in their ability to enrich the knowledge of LLMs without requiring a great deal of effort or model training.

References

- Vaibhav Adlakha, Parishad BehnamGhader, Xing Han Lu, Nicholas Meade, and Siva Reddy. Evaluating correctness and faithfulness of instruction-following models for question answering, 2023.
- Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. Benchmarking large language models in retrieval-augmented generation, 2023a.
- Jifan Chen, Grace Kim, Aniruddh Sriram, Greg Durrett, and Eunsol Choi. Complex claim verification with evidence retrieved in the wild, 2023b.
- Shahul Es, Jithin James, Luis Espinosa-Anke, and Steven Schockaert. Ragas: Automated evaluation of retrieval augmented generation, 2023.
- Zhangyin Feng, Xiaocheng Feng, Dezhi Zhao, Maojin Yang, and Bing Qin. Retrieval-generation synergy augmented large language models, 2023.
- Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Precise zero-shot dense retrieval without relevance labels. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1762–1777, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.99. URL <https://aclanthology.org/2023.acl-long.99>.
- Yunfan Gao et al. Retrieval-augmented generation for large language models: A survey, 2024.
- Or Honovich, Leshem Choshen, Roee Aharoni, Ella Neeman, Idan Szpektor, and Omri Abend. q^2 : Evaluating factual consistency in knowledge-grounded dialogues via question generation and question answering. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7856–7870, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.619. URL <https://aclanthology.org/2021.emnlp-main.619>.
- Yasuto Hoshi, Daisuke Miyashita, Youyang Ng, Kento Tatsuno, Yasuhiro Morioka, Osamu Torii, and Jun Deguchi. Ralle: A framework for developing and evaluating retrieval-augmented large language models, 2023.
- Chenxu Hu, Jie Fu, Chenzhuang Du, Simian Luo, Junbo Zhao, and Hang Zhao. Chatdb: Augmenting llms with databases as their symbolic memory, 2023.

- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.
- Jerry Liu. LlamaIndex, 11 2022. URL https://github.com/jerryjliu/llama_index.
- Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, 2023.
- Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. Query rewriting in retrieval-augmented large language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5303–5315, Singapore, December 2023a. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.322. URL <https://aclanthology.org/2023.emnlp-main.322>.
- Yubo Ma, Yixin Cao, YongChing Hong, and Aixin Sun. Large language model is not a good few-shot information extractor, but a good reranker for hard samples! *arXiv preprint arXiv:2303.08559*, 2023b.
- Yuning Mao, Pengcheng He, Xiaodong Liu, Yelong Shen, Jianfeng Gao, Jiawei Han, and Weizhu Chen. Generation-augmented retrieval for open-domain question answering. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4089–4100, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.316. URL <https://aclanthology.org/2021.acl-long.316>.
- Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. Mteb: Massive text embedding benchmark, 2023.
- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2020.
- Josh Reini et al. truera/trulens: Trulens eval v0.25.1, 2024. URL <https://zenodo.org/doi/10.5281/zenodo.4495856>.
- Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy, 2023.
- Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023a.
- Liang Wang, Nan Yang, and Furu Wei. Query2doc: Query expansion with large language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9414–9423, Singapore, December 2023b. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.585. URL <https://aclanthology.org/2023.emnlp-main.585>.

- Xiao Wang, Guangyao Chen, Guangwu Qian, Pengcheng Gao, Xiao-Yong Wei, Yaowei Wang, Yonghong Tian, and Wen Gao. Large-scale multi-modal pre-trained models: A comprehensive survey. *Machine Intelligence Research*, pages 1–36, 2023c.
- Xintao Wang, Qianwen Yang, Yongting Qiu, Jiaqing Liang, Qianyu He, Zhouhong Gu, Yanghua Xiao, and Wei Wang. Knowledgpt: Enhancing large language models with retrieval and storage access on knowledge bases, 2023d.
- Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. C-pack: Packaged resources to advance general chinese embedding, 2023.
- Fangyuan Xu, Weijia Shi, and Eunsol Choi. Recomp: Improving retrieval-augmented lms with compression and selective augmentation, 2023.



Chapter 8

LLMs in Production

Abstract The promise of LLMs has largely been driven through research efforts, where analytic performance is often prioritized over other practical aspects of their usage. Translating this promise into real-world production-grade applications is rapidly becoming a new research frontier, driven not through academic endeavors but through commercial efforts by firms aiming to differentiate themselves in the marketplace, optimize their operations, or develop unique value from applying LLMs. This chapter aims to bridge the gap from promise to practice by walking the reader through the most important aspects of applying LLMs in practice. From decisions such as which LLM to use to how to optimize LLM latency, the relevant tools and techniques are highlighted to help guide readers in their journey into LLM application development.

8.1 Introduction

In this chapter, we aim to synthesize the various factors developers should consider when building LLM-enabled applications for production. The goal is to arm the reader with the latest set of best-practice guidelines and knowledge to aid in robust, cost-effective, and safe development. As we have discussed elsewhere, LLMs represent immense promise and risk at the same time, so it is important that developers be able to navigate the various steps of the development lifecycle to maximize the realization of that promise while minimizing the risk.

We begin in Sect. 8.2 by exploring common applications for LLMs, in order to give the reader a sense of the types of use cases that the later sections contextualize. We also review the different high-level categories of LLMs available, providing the reader with an additional dimension to assess LLM suitability across different use cases. While there are many lower-level aspects of LLMs and their abilities, such as context length, number of parameters, and architecture, these have been discussed at

length elsewhere (e.g., Chapter 2), so they are not discussed here.

In Sect. 8.3 and 8.4, we introduce common metrics used for evaluating LLM applications, and provide an extensive list of canonical datasets employed for these evaluations across a broad range of use cases.

Sect. 8.5 looks at LLM selection from the perspective of open-source vs. closed-source considerations. Various LLM aspects, such as analytic quality, costs, and data security and licensing, are explored to give the reader a sense of the various trade-offs one might have to make when designing their applications. We also discuss inference latency and LLM customization in this context to help the reader understand the various constraints that the selection of an open-source or closed-source LLM might introduce to their project.

In Sect. 8.6, the aim is to provide the reader with details on the various tools, frameworks, and patterns within the rapidly evolving LLM application development ecosystem. We will discuss various details, such as the available LLM application development frameworks, prompt engineering tooling, vector storage and LLM customization.

Next, we delve into more details around inference in Sect. 8.7. This section discusses important details on model hosting options, performance optimization innovations, and, perhaps most importantly, cost optimization. The inference cost in LLMs is still a core research focus, as Sect. 4.4 in Chapter 4 outlines, so insight into the current state of optimization here is important.

The chapter finishes with an overview of an *LLMops* perspective on LLM application development. Given the complexity of LLMs and their fledgling adoption in applications, rigorous frameworks must underpin these projects. This ensures that as the potential for change in LLMs and how they can be interacted with and customized remains high, these innovations can be sustainably integrated experimentally, evaluated, and deployed with efficiency and minimal disruption to users. In this early phase of LLM adoption, maintaining user confidence and credibility is essential; an *LLMops* perspective is intended to help in this process.

8.2 LLM Applications

Before getting into the technical details about developing production-grade LLM-enabled applications, it is useful to understand some of the problems and use cases that LLMs have been applied to. To do this, we will briefly introduce the various types of generic use cases/applications for which LLMs help to improve outcomes (e.g., conversational chatbots), and then provide an overview of the different categories of LLMs available for these use cases/applications.

This overview of LLM utility will help the reader situate the more technical sections of the chapter so that they are as practically informative as possible from a development life-cycle perspective.

8.2.1 Conversational AI, chatbots and AI assistants

This category of use cases is by far the most common to which LLMs have contributed significant improvements. In chatbots and conversational AI, LLMs and their enhanced language understanding over traditional language models contribute several important new benefits. Perhaps the most significant is their natural language understanding (NLU) abilities (Wei et al., 2022). Within the context of these types of applications, the LLM’s ability comprehend user intent behind a query, and synthesize this input with existing parametric knowledge to create a coherent response, heavily influence the application’s utility.

Similarly, since users of these applications often hold open-ended conversations that may span various knowledge domains or topics, the LLM’s ability to track context is also critical to ensure coherent responses throughout the conversation session. In line with this, in the context of multi-turn dialogues, where the user and the application engage in back-and-forth conversation, LLMs leveraged the need to have the ability to selectively incorporate earlier queries or responses in the conversation to provide useful and coherent responses throughout the dialogue. Recent improvements in input context length have further advanced this specific ability in LLMs (Pawar et al., 2024) by effectively elongating the input range over which the LLM can reason.

Many of the use cases within this category of LLM application have a strong requirement for response factuality, meaning that the inherent tendency for LLMs to hallucinate is a significant challenge to be mitigated during development. The most popular way this risk is mitigated is through external knowledge bases from which relevant context can be extracted and used to condition the LLM response to verified knowledge. Integrating this knowledge base into the application architecture and the knowledge itself into the LLM input to elicit the appropriate response introduces another set of application development challenges to consider.

8.2.2 Content Creation

“Content is king”, as the saying goes in the content marketing and digital media domains. Traditionally, the generation of content, in the form of stories, blog posts, newsletters, social media content, and many more, was performed by skilled humans versed in the art of identifying the types of content that would resonate with their audience, producing that content, and disseminating it through the most efficient channels. Today, however, LLMs have taken over much of the content production step within this domain. Applications exist that allow marketing professionals to curate demographic context, provide relevant content, and provide detailed guidance

for LLM-enabled systems to generate highly engaging content and disseminate that content across channels according to a planned publication schedule.

Similarly, LLMs can be prompted to generate entire essays and stories about factual or fictional topics and events. This content is often indistinguishable from human-generated content by human readers, opening up new avenues for content creators, especially regarding the scale and diversity of content generated. However, these developments are not without their negative consequences, none more so than in educational settings, where students have quickly adopted LLMs such as OpenAI's ChatGPT to complete their assignments, leading to insufficient knowledge mastery (Lo, 2023). Nonetheless, LLMs have greatly improved the efficiency with which educators design, plan, and produce their curricula and serve as handy learning aids for students when leveraged productively.

8.2.3 Search, Information Retrieval, and Recommendation Systems

This category of LLM applications is fundamentally about providing relevant information to users as efficiently as possible. Typically, this information retrieval is performed in the context of the *needle in a haystack*, where not having mechanisms to identify and surface relevant content would mean that the user is perpetually inundated with off-target or irrelevant information. Consider a company with 10 years of product documentation mapping to multiple versions of products and their features and the complexity of navigating such a knowledge base. In the age of LLMs, users can now converse with AI assistants underpinned by innovative techniques for integrating user queries, knowledge bases, and metadata around the knowledge base to make understanding such product functionality more efficient than ever.

Again, owing to their improved NLU, entity extraction, summarization, and semantic embedding capabilities, the precision and recall with which relevant knowledge can be identified, reasoned over, and integrated into highly personalized responses is a revolution in these use cases. Much methodological, architectural, and tooling innovation around LLMs has been critical to these improvements, such as those reviewed in Chapter 7 in the context of retrieval-augmented generation and elsewhere.

8.2.4 Coding

Not surprisingly, LLMs are highly competent at generating computer programming language and natural language. The most popular solution in this space is Github Copilot, which was designed to assist human programmers in developing software using computer code. Since it is the most popular solution in this space, below we will look at its core capabilities as an exemplar of the types of benefits that these types of LLM-enabled applications provide.

- **Code auto-completion**, which can provide functionality as simple as traditional tab completion solutions for function/method and variable name completions and as complex as recommending entire code blocks based on real-time analysis of the existing code in a given script.
- **Multiple programming language support** allows developers to interoperate across coding languages efficiently. This capability is most useful in full-stack or specialist-domain application development, where multiple programming languages are used for different solution components. Imagine a full-stack developer writing data handling routines in JavaScript for the user interface. At the same time, Copilot suggests code blocks in Python for the back-end API that serves the data to the front-end. As of the time of writing, Github Copilot supports all programming languages available within public Github repositories. However, Copilot's competency in these languages is a function of that language's representation in Github public repository code. Interested readers are encouraged to explore GitHut¹ to understand better the relative proportions of different coding languages on Github.
- **Natural language understanding** enables users to specify the functionality or capabilities they would like computer code for through natural language prompting. This can often be achieved by simply writing comments describing what the subsequent code does. If Github Copilot is active for that script, it will recommend code to achieve the descriptions in the comments. These recommendations can provide surprisingly elegant code solutions to many problems and benefit from the wider context of the script being developed, especially if it is well commented/documented. Such functionality has clear benefits from an efficiency perspective. However, as always with LLM generation, users should validate and test recommended code carefully to safeguard against LLM fail-states.
- **Code refactoring and debugging** is another efficiency-improving capability of Github Copilot. Refactoring can be achieved thanks to the scale of code on which the system has been trained. Invariably, during this process, the LLM has learned many variants of code solutions for the same or similar problems, allowing it to provide alternative patterns for users to consider. Similarly, repetitive code, say for defining a class in Python, can be provided by Copilot as a boilerplate so that the developer can focus on only the functional components of the code, saving additional time. From a debugging perspective, Github Copilot can interpret execution errors or descriptions of unexpected outputs from code in the context of the code itself and the developer's description of what the code is intended to do to help identify candidate issues within the code. At a higher level, Copilot can also provide natural language explanations of execution flows and code functionality, further assisting the developers in exploring potential root causes for execution issues.

The aspects listed above contribute to increased efficiency in software development. Using a coding copilot can reduce the effort required to achieve effective and

¹ <https://madnight.github.io/githut/#/>

functional code, which traditionally might involve the use of reference textbooks, many visits to websites such as Stack Overflow or Github Gists, and code reviews by peers. Thanks to coding copilots, developers can achieve similar learning and feedback through a single intuitive interface. This is especially true thanks to some of the efforts to integrate coding copilots into popular Integrated Development Environments, such as Visual Basic Code, Vim, and JetBrains.

8.2.5 Categories of LLMs

There has been an explosion in the development of individual LLMs, especially within the open-source domain (Gao and Gao, 2023). With tweaks to the training data, the fine-tuning approach, the prompts, the scale of computing, learning objectives, and many other influential development aspects used, a proliferation of innovation has resulted in relentless improvements in many abilities. This explosion is impossible to survey within the constraints of a single book chapter. However, many online resources are available to readers to help them identify and understand the performance of specific LLMs on specific tasks and benchmarks, such as HuggingFace’s `open_llm_leaderboard`, which consolidates LLM analytic performance in real-time as new models are developed and published². These tools often allow users to filter comparisons based on their factors of interest, such as only comparing LLMs with a specific architecture type, or with a specific weight precision.

While comparative analysis of individual LLMs is essential for choosing the right model for your use case, understanding higher-level categories of LLMs is also important, as some categories are better aligned regarding outcomes to different use cases (e.g., chat vs. coding vs. retrieval). Below we highlight the most important LLM category types to enable a more informed selection process.

8.2.5.1 General-Purpose LLMs

General-purpose LLMs are typically trained on large corpora of web-sourced content. In some cases that content is curated for quality, but by and large these training sets represent a comprehensive sampling of the various topics and domains represented on the web. As such, they are highly competent in generic language tasks (e.g. NLU, entity extraction, CoT, and question-answering), but may lack competencies in domain-specific tasks (e.g. financial numerical reasoning). This category of LLM is certainly the most versatile and can be a good starting point for many applications. Many also enable task specialization if they are lacking in specific areas; however, this is more true for open-source options than for closed-source models. In summary, these models should be considered for their text generation, NLU, QA,

² https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard

translation, classification, summarization, and conversational abilities, where their out-of-the-box performance may be sufficient for most use cases.

8.2.5.2 Multimodal LLMs

Multimodal LLMs have been trained on data from more than one “modality”. Common modalities include text, audio, video, and image data (Yin et al., 2024). Training models on these different modalities enables a new set of cross-modal use cases and is rapidly becoming the new frontier of generative AI (see Chapter 9 for an extensive overview). In line with some of the use-cases for LLMs discussed in Sect. 8.2, multimodal LLMs extend their use into applications such as image retrieval based on natural language descriptions or audio generation based on natural language instruction. Multimodal content generation, such as storytelling or product specifications, where text and image generation provide a richer and more expressive user experience, is rapidly becoming an area of interest for model developers. In general, these models are very large relative to traditional LLMs, and this scale introduces its own set of challenges for adoption and integration. However, their capabilities are truly impressive, and research is ongoing to improve their analytic and computational performance.

8.2.5.3 Multilingual LLMs

Multilingual LLMs are trained on text data across more than one natural language. These types of models have received significant research attention and are useful for tasks that involve translation, multilingual reasoning, multilingual content generation, etc. Indeed, some multilingual LLMs support a large number of languages, such as the open-source model BLOOM developed by BigScience (Workshop et al., 2023), which is a 196B-parameter model trained on text across 46 natural languages and 13 programming languages. This model category’s promise is clearly aligned with cross-lingual tasks, such as reasoning over text from multiple languages (e.g. Ranaldi et al. (2024)). In terms of applications, multilingual LLMs have been leveraged for customer service and other communication use cases where code-switching, the linguistic practice of alternating between natural language in communication, is commonly exhibited (Yong et al., 2023).

8.2.5.4 Domain-Specific LLMs

In contrast with general-purpose LLMs, *domain-specific LLMs* have been trained on highly selective data from a narrow industry, field, or specialization. The general motivation when training this type of LLM is to adapt its abilities to the idiosyncrasies of the domain. This can be especially important when the domain has much jargon that does not translate in the more general context or when content within

the domain is expected to be skewed relative to the general context. For example, in biomedical science, the domain-specific BioMistral LLM was developed (Labrak et al., 2024). This model was built by adaptively pre-training a Mistral model on PubMed Central, one of the largest repositories of biomedical research literature available on the web. By adapting the General Purpose Mistral 7B-parameter LLM, the domain-specific BioMistral models outperformed the general-purpose model in 9/10 biomedical tasks. Domain-specific LLMs also exist for education, legal, economic, political, scientific, and financial fields, among others. This can be a valuable starting point for many domain-specific LLM-enabled applications.

8.3 LLM Evaluation Metrics

Evaluating LLMs is a critical process involving systematic measurements to assess how effectively these models perform specific tasks. Evaluation metrics for LLMs can be classified across multiple dimensions based on their application and methodological approach. We describe here a number of these dimensions:

1. With References vs. Without References:

- **With References:** Metrics that compare the model's output to predefined correct answers. Common in tasks such as translation and summarization.
- **Without References:** Metrics that assess quality based on the model's internal consistency and linguistic properties.

2. Character-based vs. Word-based vs. Embeddings-based:

- **Character-based:** Focus on character-level accuracy, which is useful in specific text generation tasks.
- **Word-based:** Evaluate the presence, frequency, and order of words.
- **Embeddings-based:** Leverage vector representations to assess semantic similarity beyond exact word matches.

3. Task-Agnostic vs. Task-Specific:

- **Task-Agnostic:** Metrics that can be applied across different types of tasks without modifications.
- **Task-Specific:** Metrics designed for specific applications that are directly related to the quality of task outputs.

4. Human Evaluation vs. LLM Evaluation:

- **Human Evaluation:** Involves human judges assessing the quality or relevance of model outputs. Although inherently subjective, human evaluation is invaluable for gauging natural language fluency, ensuring coherence, and verifying relevance within the specific context of use. This form of assessment provides critical insights that automated metrics might overlook, par-

ticularly in terms of the text's contextual appropriateness and the subtleties of human language understanding.

- **LLM-based Evaluation:** Involves using a second LLM to evaluate LLM outputs, often through automated metrics or model-based judgments.

5. Traditional vs. Non-Traditional

- **Traditional Metrics:** These metrics are concerned with the lexical and syntactic accuracy of the model's output. Common traditional metrics include exact string matching, string edit-distance, BLEU, and ROUGE, which prioritize the order and accuracy of words and phrases.
- **Non-Traditional Metrics:** These metrics exploit the advanced capabilities of language models to assess the quality of generated text more holistically. Examples include embedding-based methods such as BERTScore, which utilizes embeddings to compare semantic similarity, and LLM-assisted methods such as G-Eval, where another powerful LLM is used to assess the quality of the generated text.

Next, we will explore some of these metrics that are commonly employed in the evaluation of LLMs, detailing their methodologies and specific applications.

8.3.1 Perplexity

Perplexity serves as a measure of how uniformly a model predicts the set of tokens in a corpus. A lower perplexity score indicates that a model can predict the sequence more accurately, exhibiting less surprise when encountering actual data. Conversely, a higher perplexity score implies that the sequence is unexpected from the perspective of next-token probabilities generated by the model.

Given a tokenized sequence $X = (x_0, x_1, \dots, x_N)$, where N is the number of tokens, the perplexity of X is calculated as follows:

$$\text{PPL}(X) = \exp \left\{ -\frac{1}{N} \sum_{i=0}^N \log p_\theta(x_i | x_{<i}) \right\} \quad (8.1)$$

Here, $\log p_\theta(x_i | x_{<i})$ represents the log-likelihood of the i -th token, conditioned on all preceding tokens $x_{<i}$, as determined by the model. This value reflects the model's predictive accuracy per token within the sequence.

8.3.2 BLEU

One of the predominant metrics in this category is the *Bilingual Evaluation Understudy* (BLEU) score, which was introduced by Papineni et al. (2002), primarily for evaluating the quality of text translated from one natural language to another.

BLEU assesses the closeness of machine-generated text to one or more reference translations by examining the frequency and presence of consecutive words, known as n-grams, in both texts.

The mathematical representation of BLEU involves several steps:

- **Precision Calculation:** Precision is computed for n-grams of different lengths. For a given n-gram length n , precision p_n is the ratio of the number of n-grams in the generated text that match the reference text to the total number of n-grams in the generated text. This count is clipped by the maximum number of times an n-gram appears in any reference text, which avoids over-counting.

$$p_n = \frac{\text{Number of clipped matching n-grams}}{\text{Total number of n-grams in generated text}} \quad (8.2)$$

- **Geometric Mean of Precision:** After calculating precision for various n-gram lengths, a final BLEU score, referred to as BLEU-N, is determined using the geometric mean of these precision values across all considered n-gram lengths.

$$\text{BLEU-N} = \left(\prod_{n=1}^N p_n \right)^{\frac{1}{N}} \quad (8.3)$$

- **Brevity Penalty:** To address the limitation of precision favoring shorter text (since shorter texts are likely to have a higher precision by virtue of fewer opportunities for error), BLEU incorporates a brevity penalty. This penalty is applied if the length of the generated text is shorter than the reference, thus discouraging overly concise translations.

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases} \quad (8.4)$$

- **Final BLEU Score:** The overall BLEU score combines the geometric mean of the precision scores with the brevity penalty to produce a final score between 0 and 1, where 1 indicates a perfect match with the reference texts.

$$\text{BLEU} = BP \cdot \exp \left(\sum_{n=1}^N \frac{1}{N} \log p_n \right) \quad (8.5)$$

Despite its widespread use, it is important to note BLEU's limitations. It does not account for the semantic accuracy or grammatical correctness of the generated text. Therefore, while BLEU is useful for a preliminary assessment of translation quality, it should be supplemented with other metrics or human evaluations to capture the nuances of language generation more comprehensively.

8.3.3 ROUGE

Another metric, called the *Recall-Oriented Understudy for Gisting Evaluation* (ROUGE), differs from BLEU in that it is recall-oriented. It primarily assesses how many words from the reference texts are also present in the machine-generated output, making it especially useful for evaluating automatic summarization tasks.

ROUGE includes several variants, each with a specific focus:

- **ROUGE-N:** ROUGE encompasses a collection of metrics designed to assess the effectiveness of summaries and translations by contrasting generated text against a set of human-crafted reference summaries.

As it is focused on recall, ROUGE primarily evaluates the extent to which words and phrases from the reference summaries are reproduced in the generated text. This focus makes ROUGE especially valuable in scenarios where capturing as much of the reference content as possible is crucial.

$$\text{ROUGE-N} = \frac{\sum_{S \in \{\text{Reference Summaries}\}} \sum_{\text{gram}_n \in S} \text{Count}_{\text{match}}(\text{gram}_n)}{\sum_{S \in \{\text{Reference Summaries}\}} \sum_{\text{gram}_n \in S} \text{Count}(\text{gram}_n)} \quad (8.6)$$

In this formula, gram_n denotes n-grams of length n , and $\text{Count}_{\text{match}}(\text{gram}_n)$ is the maximum number of times that an n-gram occurs in both a candidate summary and the set of reference summaries.

Examples include:

- ROUGE-1 for unigrams.
- ROUGE-2 for bigrams.

- **ROUGE-L:** Focuses on the longest common subsequence (LCS) between the generated and reference texts. Unlike n-gram overlap, LCS does not require the sequence to be contiguous, thereby capturing more flexible matches.
- **ROUGE-W:** An extension of ROUGE-L, this variant incorporates the length of the texts into its evaluation to counter the length bias.
- **ROUGE-S:** Measures the skip-bigram co-occurrence, which accounts for any pair of words in their sentence order, regardless of gaps. This metric emphasizes the order in which content is mentioned, regardless of intervening content.
- **ROUGE-SU:** Enhances ROUGE-S by including both skip-bigrams and unigrams in the evaluation:

8.3.4 BERTScore

Introduced by [Zhang et al. \(2020\)](#), *BERTScore* represents an advanced methodology for evaluating text quality by utilizing deep contextual embeddings from the BERT model. Unlike traditional metrics such as BLEU and ROUGE, which assess token or

n-gram overlap, BERTScore calculates a similarity score for each token in the candidate text against each token in the reference text using these contextual embeddings.

BERTScore employs greedy matching to ensure that each token from the candidate text is aligned with the most similar token from the reference text, optimizing the overall similarity score. The evaluation includes three key metrics:

- **Recall (R_{BERT}):** This metric is calculated by taking the maximum similarity score for each token in the reference text, summing these scores, and then normalizing by the number of tokens in the reference. It reflects the extent to which the candidate text captures the content of the reference.

$$R_{BERT} = \frac{1}{|x|} \sum_{x_i \in x} \max_{\hat{x}_j \in \hat{x}} \langle x_i, \hat{x}_j \rangle \quad (8.7)$$

- **Precision (P_{BERT}):** Similar to recall, precision sums the maximum similarity scores for each token in the candidate text and normalizes by the number of tokens in the candidate. It measures the extent to which tokens in the candidate text are represented in the reference.

$$P_{BERT} = \frac{1}{|\hat{x}|} \sum_{\hat{x}_j \in \hat{x}} \max_{x_i \in x} \langle \hat{x}_j, x_i \rangle \quad (8.8)$$

- **F1 score (F_{BERT}):** The harmonic mean of precision and recall, providing a balanced measure of both completeness and precision.

$$F_{BERT} = 2 \frac{P_{BERT} \cdot R_{BERT}}{P_{BERT} + R_{BERT}} \quad (8.9)$$

BERTScore, offers semantic awareness and robustness to paraphrasing, making it highly effective for evaluating translations or summaries. However, it demands substantial computational resources and may not always correspond with human judgments, especially in evaluating the structure and coherence of text.

8.3.5 MoverScore

MoverScore evaluates the semantic similarity between a system’s predicted text and a reference text using the concept of *Word Mover’s Distance* (WMD) (Kusner et al., 2015). This metric helps capture semantic distances between words and phrases, making it particularly useful for text evaluation tasks. Unlike BERTScore, which utilizes one-to-one matching (or “hard alignment”) of tokens, MoverScore incorporates many-to-one matching (or “soft alignment”), allowing for more flexible token alignments.

The key components of MoverScore include the following:

- **Transportation Flow Matrix (F):** This matrix represents the amount of flow F_{ij} traveling from the i -th n-gram x_i^n in the predicted sequence \mathbf{x}^n to the j -th n-gram y_j^n in the reference sequence \mathbf{y}^n .
- **Cost Matrix (C):** C is the transportation cost matrix, where each entry C_{ij} represents the distance $d(x_i^n, y_j^n)$ between the i -th n-gram of the prediction and the j -th n-gram of the reference.
- **Element-wise Matrix Operation:** The total transportation cost is calculated by the sum of all entries of the matrix product $C \odot F$, where \odot denotes element-wise multiplication.
- **N-gram Sequences:** Both system predictions x and references y are viewed as sequences of words, and their respective n-grams (e.g., unigrams, bigrams) are utilized in the calculation.
- **Weight Vectors (f_{x^n} and f_{y^n}):** These vectors of weights correspond to n-grams in \mathbf{x}^n and \mathbf{y}^n . They form a distribution over n-grams, typically normalized so that their sum equals one.

The WMD is computed as the minimum value of the transportation flow that satisfies:

$$\text{WMD}(\mathbf{x}^n, \mathbf{y}^n) := \min_{F \in \mathbb{R}^{|\mathbf{x}^n| \times |\mathbf{y}^n|}} \langle C, F \rangle, \quad \text{s.t.} \quad F\mathbf{1} = f_{x^n}, \quad F^T\mathbf{1} = f_{y^n} \quad (8.10)$$

In practice, MoverScore evaluates semantic distances using Euclidean distance between n-gram embeddings:

$$d(x_i^n, y_j^n) = \|E(x_i^n) - E(y_j^n)\|^2 \quad (8.11)$$

where E represents the embedding function that maps an n-gram to its vector. While traditional methods such as word2vec are used, contextualized embeddings such as ELMo and BERT are preferred for their ability to incorporate sentence-level context.

N-gram embeddings are computed as:

$$E(x_i^n) = \sum_{k=i}^{i+n-1} \text{idf}(x_k) \quad (8.12)$$

Here, $\text{idf}(x_k)$ is the inverse document frequency of x_k , and the weight for each n-gram, $f_{x_i^n}$, is determined by:

$$\mathbf{f}_{x_i^n} = \frac{1}{Z} \sum_{k=i}^{i+n-1} \text{idf}(x_k) \quad (8.13)$$

with Z as a normalization constant to ensure $\sum f_{x_i^n} = 1$. When n is greater than the sentence length, resulting in a single n-gram, MoverScore simplifies to Sentence Mover's Distance (SMD):

$$\text{SMD}(\mathbf{x}^n, \mathbf{y}^n) := \|E(\mathbf{x}_1^l) - E(\mathbf{y}_1^l)\| \quad (8.14)$$

where l_x and l_y are the size of sentences.

8.3.6 G-Eval

G-EVAL offers a structured and dynamic method to evaluate generated texts, aiming to provide more detailed and nuanced insights into text quality compared to more traditional methods. It addresses challenges such as variance in scoring and alignment with human judgment by proposing modifications in score calculation and presentation.

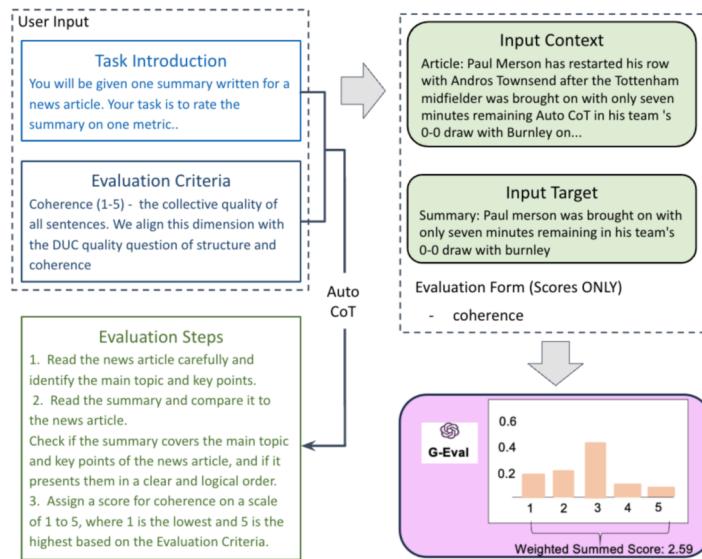


Fig. 8.1: The G-EVAL framework process. Initially, the Task Introduction and Evaluation Criteria are provided to the LLM, which then generates a Chain-of-Thoughts (CoT) outlining detailed evaluation steps. Subsequently, this CoT, along with the initial prompt, is used to assess the NLG outputs using a form-filling approach. The process concludes with a computation of the final score, which is the probability-weighted sum of the individual scores obtained.

G-EVAL is structured around a systematic approach involving three key components as shown in Fig. 8.1:

1. **Prompt:** The prompt outlines the task definition and the specific criteria for evaluation. For example, in text summarization, the prompt would define the task and specify evaluation criteria such as coherence, asking reviewers to rate the summary based on these aspects.

2. **Chain-of-Thoughts:** This is a sequence of intermediate instructions generated by the LLM, providing detailed steps for conducting the evaluation. The CoT aids in guiding the evaluator through the process, enhancing the consistency and depth of the evaluation. For instance, for coherence, the CoT might instruct the evaluator to read the original article and the summary, compare key points, and check the logical order before scoring.
3. **Scoring function:** This component involves the LLM executing the evaluation by combining the prompt, CoT, the original text, and the target text. It then outputs a score based on the probabilities of return tokens that are calculated from the model's response to the evaluation prompt. The scoring is formulated as follows:

$$\text{score} = \sum_{i=1}^n p(s_i) \times s_i \quad (8.15)$$

where s_i represents possible scores predefined in the prompt, and $p(s_i)$ is the probability of each score assigned by the LLM.

8.3.7 Pass@k

The functional correctness of code generated by language models can be effectively assessed using the *pass@k* metric, which was originally introduced by [Kulal et al. \(2019\)](#). This metric evaluates the likelihood of generating at least one correct code sample among multiple attempts for each coding problem, quantified through unit tests. To implement this approach, the following steps are observed:

1. For each problem, generate n code samples, with $n \geq k$. For example, $n = 200$ and $k \leq 100$.
2. Count the number of samples, denoted as c , that successfully pass the unit tests, with $c \leq n$.
3. Compute the unbiased estimator of pass@k using the following formula:

$$\text{pass}@k = 1 - \frac{\binom{n-c}{k}}{\binom{n}{k}} \quad (8.16)$$

This calculation provides the probability that at least one of the k selected samples from n generated samples passes the unit tests, thereby offering a robust metric to gauge the model's ability to solve programming tasks.

8.4 LLM Benchmark Datasets

Benchmarks are essential because they provide a set of standardized metrics that enable fair comparisons among different LLMs. They help identify which models perform best in specific contexts and track the progress and refinement of a single LLM over time or help in comparing and contrasting different LLMs. Benchmark datasets are constructed as either collections of many or unique tasks. Each task within a benchmark dataset comes with its own evaluation metrics, ensuring that models are tested on specific linguistic abilities.

In this section, we discuss a number of key datasets and explore their purpose.

- **Multi-Task or General Abilities**

- **Benchmark:** MMLU [Hendrycks et al. \(2020\)](#), SuperGLUE [Wang et al. \(2019\)](#), BIG-bench [Srivastava et al. \(2022\)](#), GLUE [Wang et al. \(2018\)](#), BBH [\(Srivastava et al., 2022\)](#), Blended Skill Talk ([Smith et al., 2020](#)) and HELM ([Liang et al., 2022](#)).
- **Purpose:** These benchmarks are designed to evaluate the performance of language models across a variety of tasks, providing a comprehensive assessment of a model’s general language understanding, reasoning, and generation abilities, among others.

- **Language Understanding**

- **Benchmark:** CoQA [Reddy et al. \(2019\)](#), WiC [Pilehvar and Camacho-Collados \(2018\)](#), WikiText103 [Merity et al. \(2016\)](#), PG19 [Rae et al. \(2019\)](#), QQP [Le et al. \(2021\)](#), CB [De Marneffe et al. \(2019\)](#), CNSS [Liu et al. \(2018\)](#), CKBQA [Li et al. \(2016\)](#), AQuA [Ling et al. \(2017\)](#), OntoNotes [Weischedel et al. \(2011\)](#), HeadQA [Vilares and Gómez-Rodríguez \(2019\)](#), and Twitter Dataset [Blodgett et al. \(2016\)](#).
- **Purpose:** These benchmarks focus on different aspects of language understanding, including question answering, word-in-context disambiguation, and sentiment analysis.

- **Story Cloze and Sentence Completion**

- **Benchmark:** StoryCloze ([Mostafazadeh et al., 2016](#)), LAMBADA [Paperno et al. \(2016\)](#), AdGen [Shao et al. \(2019\)](#), and E2E [Novikova et al. \(2017\)](#).
- **Purpose:** These benchmarks test a model’s ability to complete stories and sentences, which requires understanding narrative context, commonsense reasoning, and generating coherent text.

- **Physical Knowledge and World Understanding**

- **Benchmark:** PIQA [Bisk et al. \(2020\)](#), TriviaQA [Joshi et al. \(2017\)](#), ARC [Clark et al. \(2018\)](#), ARC-Easy [Clark et al. \(2018\)](#), ARC-Challenge [Clark et al. \(2018\)](#), PROST [Aroca-Ouellette et al. \(2021\)](#), OpenBookQA [Mihaylov et al. \(2018\)](#), and WebNLG [Ferreira et al. \(2020\)](#).

- **Purpose:** These datasets challenge models to demonstrate an understanding of physical concepts and general world knowledge, often in a question-answering format.

- **Contextual Language Understanding**

- **Benchmark:** RACE [Lai et al. \(2017\)](#), RACE-Middle [Lai et al. \(2017\)](#), RACE-High [Lai et al. \(2017\)](#), QuAC [Choi et al. \(2018\)](#), StrategyQA [Geva et al. \(2021\)](#), and Quiz Bowl [Boyd-Graber et al. \(2012\)](#)

- **Purpose:** These benchmarks assess a model’s ability to understand and interpret language in context, which is crucial for applications like chatbots and content analysis.

- **Commonsense Reasoning**

- **Benchmark:** WinoGrande [Sakaguchi et al. \(2021\)](#), HellaSwag [Zellers et al. \(2019\)](#), COPA [Roemmele et al., 2011](#), WSC [Levesque et al. \(2012\)](#), CSQA [Talmor et al. \(2018\)](#), SIQA [Sap et al. \(2019\)](#), ReCoRD [Zhang et al. \(2018\)](#).

- **Purpose:** These benchmarks are designed to evaluate models on their ability to apply commonsense reasoning, causal understanding, and real-world knowledge to complex natural language tasks,

- **Reading Comprehension**

- **Benchmark:** SQuAD [Rajpurkar et al. \(2016\)](#), BoolQ [Clark et al. \(2019\)](#), SQuADv2 [Rajpurkar et al. \(2018\)](#), DROP [Dua et al. \(2019\)](#), RTE [Dagan et al. \(2005\)](#), WebQA [Chang et al. \(2022\)](#), MultiRC [Khashabi et al. \(2018\)](#), Natural Questions [Kwiatkowski et al. \(2019\)](#), SciQ [Welbl et al. \(2017\)](#), and QA4MRE [Peñas et al. \(2013\)](#).

- **Purpose:** Reading comprehension benchmarks test a model’s ability to parse and understand text passages and to answer questions based on that text.

- **Mathematical Reasoning**

- **Benchmark:** MATH [Hendrycks et al. \(2021\)](#), Math23k [Wang et al. \(2017\)](#), GSM8K [Cobbe et al. \(2021\)](#), MathQA [Austin et al. \(2021\)](#), MGSM [Shi et al. \(2022\)](#), MultiArith [Roy and Roth \(2016\)](#), ASDiv [Miao et al. \(2021\)](#), MAWPS [Koncel-Kedziorski et al. \(2016\)](#), SVAMP [Patel et al. \(2021\)](#).

- **Purpose:** These datasets evaluate a model’s ability to solve mathematical problems, ranging from basic arithmetic to more complex questions involving algebra and geometry.

- **Problem Solving**

- **Benchmark:** HumanEval [Chen et al. \(2021\)](#), DS-1000 [Lai et al. \(2023\)](#), MBPP [Austin et al. \(2021\)](#), APPS [Hendrycks et al. \(2021\)](#), and CodeCon-tests [Li et al. \(2022\)](#).
- **Purpose:** Problem-solving benchmarks test a model’s ability to apply logic and reasoning to solve various problems, including coding challenges.

- **Natural Language Inference and Logical Reasoning**

- **Benchmark:** ANLI [Nie et al. \(2019\)](#), MNLI-m [Williams et al. \(2017\)](#), MNLI-mm [Williams et al. \(2017\)](#), QNLI [Rajpurkar et al. \(2016\)](#), WNLI [Levesque et al. \(2012\)](#), ANLI R1 [Nie et al. \(2019\)](#), ANLI R2 [Nie et al. \(2019\)](#), ANLI R3 [Nie et al. \(2019\)](#), HANS [McCoy et al. \(2019\)](#), LogiQA [Liu et al. \(2020\)](#), and StrategyQA [Geva et al. \(2021\)](#).
- **Purpose:** These benchmarks assess a model’s ability to make inferences based on a given text, a key component of understanding and reasoning in natural language.

- **Cross-Lingual Understanding**

- **Benchmark:** MLQA [Lewis et al. \(2019\)](#), XNLI [Conneau et al. \(2018\)](#), PAWS-X [Yang et al. \(2019\)](#), XSum [Narayan et al. \(2018\)](#), XCOPA [Ponti et al. \(2020\)](#), XWinograd [Tikhonov and Ryabinin \(2021\)](#), TyDiQAGoldP [Clark et al. \(2020\)](#), MLSum [Scialom et al. \(2020\)](#).
- **Purpose:** Cross-lingual benchmarks evaluate a model’s ability to understand and process language across different linguistic contexts, which is important for applications in multilingual environments.

- **Language Translation**

- **Benchmark:** WMT [Bojar et al. \(2016\)](#), WMT20 [Loïc et al. \(2020\)](#), and WMT20-enzh [Loïc et al. \(2020\)](#).
- **Purpose:** Translation benchmarks assess a model’s proficiency in translating text between languages, a fundamental task in natural language processing.

- **Dialogue**

- **Benchmark:** Wizard of Wikipedia [Dinan et al. \(2018\)](#), Empathetic Dialogues [Rashkin et al. \(2018\)](#), DPC-generated dialogues [Hoffmann et al. \(2022\)](#), and ConvAI [Dinan et al. \(2020\)](#).
- **Purpose:** Dialogue benchmarks evaluate a model’s ability to engage in coherent and contextually appropriate conversations, which is key for chatbot development.

8.5 LLM Selection

It is fair to say that choosing the most suitable LLMs for your application is the single most important decision. The competency improvements made in language models/-modeling, punctuated by the release of ChatGPT by OpenAI in November 2022, are the main reason for this book, as well as the explosion in innovation stemming from their adoption. However, it is important to realize that LLM competency, or analytic quality, is only one of several attributes one needs to consider when choosing which LLM to leverage for a given application.

Creating your own decision-flow diagram for LLM and customization path selection

The following sections will explore the various LLM attributes and their relevance to the application development context to enable readers to establish their decision flow diagram that maps model attributes to the application domain more appropriately. This process is critical for ensuring that you can solve the problem your application aims for and maintain the solution sustainably in the future, maximizing efficiency and efficacy while minimizing the inherent risks associated with LLM adoption. Think carefully about your application's functional and non-functional requirements, and map them to the various promises and challenges discussed below to establish a rigorous decision-making process.

Many other criteria and model attributes should be considered, as the choice of LLM occurs early in the project and influences many options. As a guiding example of how this LLM selection and development process might proceed within a given domain, consider Fig. 8.2, adapted from Li et al. (2023), which illustrates how one might make decisions between the use of open-source vs closed-source LLMs, and the LLM customization path taken based on criteria such as tooling, data and budget availability. The customization pathways are sequenced from least expensive at the top to most expensive at the bottom, representing a pragmatic, cost-aware sequencing of options.

Another useful framework for selecting LLMs for your project is the *total cost of ownership* (TCO). This approach integrates many different specific costs for the details of your project – model, use-case, etc. – into a total sum for easy comparison between different options. Some of the line-items include:

- **Per Token Costing**, which captures the per-query processing and generation costs.
- **Labor Cost**, which estimates the human resourcing cost associated with building and deploying the LLM service.
- **Total setup costs**, which estimates the total cost of deploying, running and managing the LLM service.

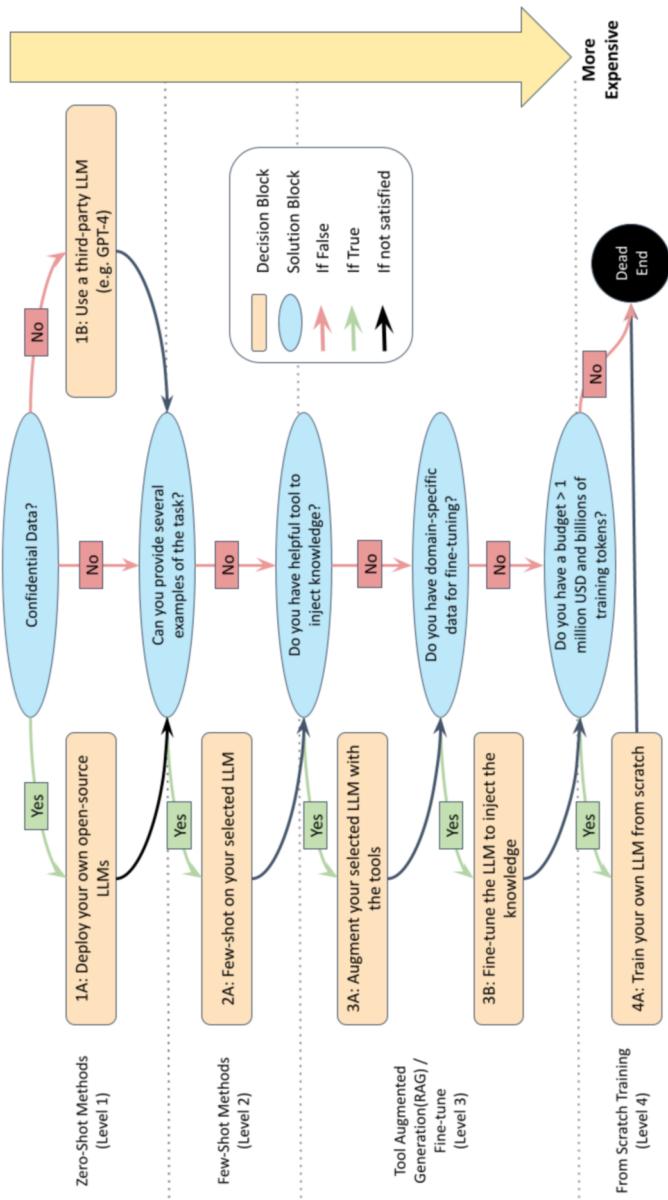


Fig. 8.2: A decision flow diagram for selecting between open-source and closed-source LLM and which customization path to follow within the financial services domain.

A good starting point for developers wishing to understand these factors and the process better is available on HuggingFace³, which includes an interactive TCO calculator⁴. Readers are encouraged to explore this and similar resources to understand better the framework and how it can aid the decision-making process in LLM application development.

Each category of LLMs has its pros and cons, and where each of these matters is highly context dependent. For example, selecting an LLM purporting to have SOTA performance on an entity extraction benchmark for an application leveraging mainly text summarization would not make much sense. This section aims to provide sufficient coverage of the key selection criteria to aid developers in establishing this contextual awareness of LLM attribute relevance, enabling informed decisions in their own development work.

8.5.1 Open Source vs. Closed Source

One of the highest level criteria that developers use to decide which LLMs to use in their applications is *open-source* vs. *closed-source*. In general, the main trade-offs between open-source vs. closed-source LLMs are in the dimensions of usage flexibility, usage convenience, and cost. But there are many additional factors to consider. Tab. 8.1 summarizes a fuller list of relevant criteria. While it may be initially attractive for a development team to adopt an open-source LLM based on low usage-costs or high usage-flexibility, for example, a full evaluation across all of the criteria listed in Tab. 8.1 may reveal that the TCO of an application leveraging open-source models is much greater than that of leveraging a closed-source model. As such, developers must assess their choice of LLM as comprehensively as possible. For each consideration of LLM selection discussed in this section, we highlight relevant trends in open source vs. closed source.

8.5.2 Analytic Quality

The most heavily weighted of all considerations in LLM choice is typically the quality with which a given LLM can execute tasks relevant to the use case you are solving. Larger models tend to have stronger analytic performance, making cost – computational and financial – the primary trade-off that must be considered. A useful reference point for analytic quality are compiled reference leader-boards, wherein LLMs have been evaluated on a broad range of standardized benchmarks, enabling direct comparative selection. Note that these benchmark results are not foolproof and should be interpreted carefully in line with the methodology used. (Alzahrani et al., 2024).

Table 8.1: The various aspects to be considered when deciding between open-source and closed-source large language models for application development.

Aspect	Closed-Source	Open-Source
Accessibility	Often, details on model architecture, training data, learning objectives and source code is not available to the end-user	Details on model architecture, training data, learning objectives and source code are available to the end-user under permissive licenses
Transparency	End-user access is typically through prescriptive APIs, meaning minimal model weights. Developers are responsible for transparency in the attributes from the sible for hosting and serving the model. This has implications for us- age in various settings, including de- bugging, maintenance and support ity, however, developers are responsible the application.	End-user access is typically through prescriptive APIs, meaning minimal model weights. Developers are responsible for transparency in the attributes from the sible for hosting and serving the model. High transparency enables a large range of customizations and use-case flexibility, however, developers are responsible then for support these.
Cost	Usage costs for closed-source models is typically higher than that of open-source models. However, this is in part thanks to higher price-point is often balanced by the end-users ability to customize and optimize their compute consumption etc. However, low usage cost should be carefully weighted against development and support costs, which are both the responsibility of the end-user	Usage costs for open source models can be significantly lower than closed-source models. This is due to the fact that open source models are often funded by the developer or organization that created them. These models are also often released under permissive licenses, which allows for free use and modification. This makes them more accessible and affordable for smaller organizations and individuals.
Customization	Limited to prescriptive approaches. These often come with higher cost than adaptive pre-training, full-parameter open source options	Customization is fully flexible through fine-tuning, parameter-efficient fine-tuning, and prompt-based learning.
Usage	Typically usage is through a limited set of prescriptive mechanisms such as a signed and graphical user interface or an API	Usage is fully flexible and can be developed by the end-user
Collaboration	Limited to whatever integration with collaborative tooling exists	Collaboration is flexible and can be enabled through any collaboration tooling or framework of interest to the end-user
Security	Fully managed by the model owner. This can be valuable for small development teams without security resources. However, no security solution is 100% effective, as vulnerabilities in these models/services can be difficult to identify thanks to their closed-source settings.	Security must be developed and managed by the end-user. While this enables maximum flexibility, it can be a challenge. However, no security solution is 100% effective, as vulnerabilities in these models/services can be difficult to identify thanks to their closed-source settings.
Privacy	Any data sent as input to the model is subject to the end-user agreement. This may mean the provider has permission or preferences to leverage these data for future versions of the model, which introduces IP loss considerations	The end-user determines the level of privacy according to their requirements. This may mean the provider has permission or preferences to leverage these data for future versions of the model, which introduces IP loss considerations

Continued on next page

Table 8.1 – *Continued from previous page*

Aspect	Closed-Source	Open-Source
Support	End-users are provided support through the model owner. The end-user is responsible for providing support around collaborative functionality delivered within their application	The model is often available with no end-user support. However, an active open-source community and wrapper functionality delivered may be available to assist end-users with issues through Github or Huggingface projects.
Updates and Maintenance	The model owner schedules and releases model updates. These happen without transparency and in accordance with the model owner's commercial road-map, which may be directed by all end-users	The end-user is responsible for all model updates and maintenance. While this provides maximum control, it can be a costly responsibility to own for smaller development teams

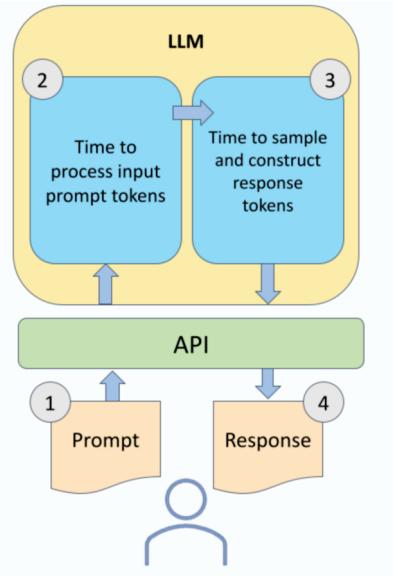
Nonetheless, these leader-boards are a good ballpark view of the relative performance of one LLM over another and provide a useful way to quickly down-select to a more manageable subset of candidate models to be further evaluated for suitability for your project. The best maintained of these is the HuggingFace Open LLM Leader-board referred to in Sect. 8.2.5. However, this approach is limited to open-source LLMs only. Other leader-boards that span both open and closed-source LLMs are available, however the stability of these projects is unknown (e.g. <https://llm-leaderboard.streamlit.app/>), so some web searches may be required to find a good resource when you wish to evaluate across both LLM domains.

Once a developer has down-selected to a manageable subset of candidate LLMs, it is a good idea to evaluate analytic performance further using more targeted tasks. Since LLM task performance is sensitive to the data used, leader-board benchmark results might represent overestimates relative to its performance in a data distribution more aligned to the domain for which you are developing your application. This second, more use-case specific evaluation of your subset of candidate LLMs should further enable down-selection to LLMs that either perform best on the use-case aligned evaluation or show promise if further prompt engineering, pre-training, or fine-tuning is in-scope for the project (Yuan et al., 2023).

8.5.3 Inference Latency

LLM inference latency, which can be considered as the total time it takes for a single request to be processed and a single response to that request to be returned to the user (Fig. 8.3), is a key factor to consider when choosing an LLM. Ignoring the latency introduced from getting the input prompt from the user to the LLM’s API (#1 in Fig. 8.3), and the LLM response back from the LLM’s API to the user (#4 in Fig. 8.3), as these are mostly a matter of network optimization, there are two key inference phases that most influence overall inference latency. Namely, the time it takes to process the

Fig. 8.3 LLM latency is the combination of 1) the time it takes for the user's prompt to reach the LLM's inference API, 2) the time it takes for the LLM to process the user's prompt, 3) the time it takes for the LLM to sample the relevant tokens and compose its response, and 4) the time it takes for the LLM's response to be delivered from the API to the user.



input prompt's tokens through the LLM network (#2 in Fig. 8.3) and the time it takes to sample and compose response tokens (#3 in Fig. 8.3), also known as the *prefill* step and *decode* step, respectively (Agrawal et al., 2024).

Owing to the Transformer architecture, prompt tokens can be processed in parallel within the prefill step, which results in relatively high latency (compared to the decode step) and high compute utilization due to this parallelism. In contrast, the decode step is a sequential process in that the next token to be generated in a sequence of output tokens depends on all previous tokens being generated first. This results in relatively low per-output-token latency, but also low compute utilization due to the sequential nature of the process. This means that the number of input tokens within the prompt should not significantly impact inference latency, while the output length will. For example, Tab. 8.2⁵ shows the impact of varying the input and output token lengths on the response latency for OpenAI's gpt-3.5-turbo model. Increasing the number of input tokens from 51 to 232 while keeping the number of output tokens at 1 results in negligible latency change. However, using a similar input length but increasing the output token length from 1 to 26 results in an almost 3x latency increase, illustrating the imbalanced effect of input and output length on inference latency.

With this imbalance in mind, what attributes of an LLM influence inference latency? The first and most obvious is model size. The simple rule of thumb is that more parameters result in greater latency. LLMs with more model parameters require more computation to process inputs and generate outputs. In addition to model size,

⁵ Reproduced from https://huyenchip.com/2023/04/11/llm-engineering.html#cost_and_latency.

model architecture is another important factor. The number of layers, the complexity of layers, the attention mechanisms used in Transformer blocks, and the number and location of Transformer blocks within the network influence inference latency.

Another important factor influencing inference latency in LLMs is the numeric precision with which model parameters are stored. This aspect is discussed in detail within the quantization sections in Chapter 4. However, in the context of open vs closed-source LLMs, the customization difference between the two categories of models is important. In the closed-source context, where customization is more restrictive, end-user quantization will be limited to whatever the model owner supports. In contrast, in the open-source context, the end-user of the LLM is typically free to test and implement whatever quantization approach works best for their use-case. Since quantization represents a significant opportunity for inference latency decrease and decreases in the memory and storage costs of running/hosting the LLM, any lack of customization in closed-source LLMs should be considered strongly. In use cases where the number of request-response cycles is expected to be low, this might be less of an issue. Nevertheless, when the number of request-response cycles is high, a closed-source LLM might become a problematic bottleneck within an application – for example OpenAI APIs typically have rate-limits that apply to different end-points and models.

8.5.4 Costs

Many aspects of LLMs and their utilization within an application development setting incur costs. Often, cost considerations are limited to the per-token costs of inference, which is certainly one of the most important. However, per-token costs are a moving target, with significant research and commercial investment in relentlessly

Table 8.2: Impact of input length and output length on inference latency. Numbers were calculated for OpenAI’s *gpt-3.5-turbo* model. Some portion of the variation in these results is a result of API latency since how OpenAI schedules and routes user queries is unknown to the user. However, the relationship between input and output length settings remains stable, even if the absolute latency changes. The **p50 latency (s)** indicates that 50% of requests made ($n=20$) received responses at least as fast or faster than the value listed.

# Input Tokens	# Output Tokens	p50 latency (s)
51	1	0.58
232	1	0.53
228	26	1.43

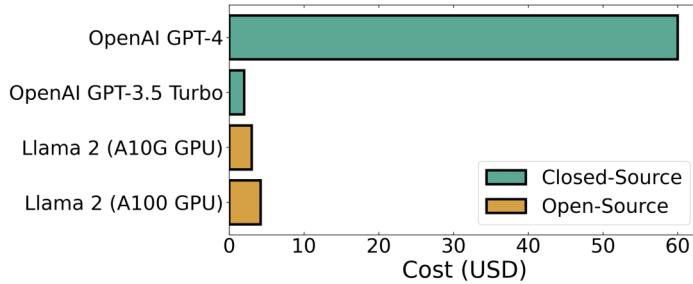


Fig. 8.4: USD cost of generating 1 million tokens. Comparison between two closed-source LLMs, OpenAI’s GPT-4 and GPT-3.5 Turbo models, and one open-source LLM, Llama-2-7B parameter model running on two different GPUs, the A100 and the A10G. Although both Llama-2 generations cost more than 10x less than OpenAI’s most capable LLM, GPT-4, their GPT-3.5 Turbo model costs less than both Llama-2 generations. This trend in closed-source inference costs going down is important to consider when choosing which LLM you will use for your project. Costs were valid at the time of analysis, which was August 2023.

driving them down. For example, consider the trends shown in Fig. 8.4⁶. The consensus view in the debate over open-source vs closed-source LLM adoption has been that closed-source models typically have a significantly higher per-token unit cost than open-source LLMs. However, this is likely true only for the most capable versions of closed-source LLM, as indicated by OpenAI’s pricing strategy, where inference costs for older LLM offerings tend to be a fraction of their latest offerings.

In combination with the per-token generation cost perspective, developers should consider the cost of other aspects of the application development life cycle, in keeping with the TCO framework. For instance, adopting an open-source LLM might have lower inference costs, but might also mean that analytic quality is lower. This analytic quality gap might be solvable with fine-tuning or investment in prompt engineering, but this optimization is not free. Data for fine-tuning or testing are needed, and this collection, annotation, and curation process can be labor-intensive and complex. Moreover, if one fine-tunes an LLM, its performance will need to be maintained on an ongoing basis, meaning that this effort to continuously evaluate and improve the model’s performance (if required) is an additional cost to be tracked. Indeed, customizing and maintaining LLMs is a complex technical task, meaning that a project’s expertise and talent costs will be greater than if a closed-source LLM option is adopted. Model hosting and compute management are other costs to be directly incurred when selecting an open-source LLM, increasing the overall complexity and cost.

⁶ Reproduced from https://medium.com/@ja_adimi/comparison-cost-analysis-should-we-invest-in-open-source-or-closed-source-langs-bfd64ae1f74.

8.5.5 Adaptability and Maintenance

Open-source LLMs have greater adaptability than closed-source LLMs since their weights, training data, and source code are often directly available to the end-user. This enables the adaptation or customization of open-source LLMs using any or all of the techniques presented in Chapters 3 and 4, which can provide important control over the behavior and performance of an application. However, as the saying goes, “*there is no free lunch*”, and this flexibility must be traded-off against a more resource-intensive development life-cycle.

Conversely, the lower adaptability of closed-source LLMs must be considered in light of the much lower resource-intense development life cycle. If a project leverages LLMs to execute common tasks, then it is likely that a proprietary option will provide good capabilities in this task out of the box, thus negating the need for adaptation or customization. Similarly, advanced prompt engineering techniques, such as n-shot in-context learning, can improve outcomes further. Opting for a closed-source or proprietary LLM might be a good option in these circumstances. However, ongoing maintenance is still a factor in this decision. Closed-source maintenance is typically not transparent and occurs in accordance with the LLM owner’s road map or maintenance schedule. Assuming these changes can occur without prior notice to end-users, developers should understand the risks to their application’s performance in the event that a silent upgrade of their chosen LLM occurs. Could the upgraded LLM degrade the user experience? Could it introduce ethical or safety risks if not handled correctly?

To a large extent, many of these types of risks can be mitigated with a suitable application development life-cycle that incorporates ongoing monitoring and evaluation. However, the scale and complexity of LLMs mean that *a priori* anticipating all fail-states is impossible. As such, the use case is the key to deciding which LLM is best for your application. In settings where errorful application behavior carries a high cost (e.g., in regulated industries), then leaving user outcomes to chance, or more appropriately put, to the discretion of a 3rd party such as OpenAI or Anthropic might not be possible. Thus, the only option is to choose an LLM where these risks can be fully owned by you as the application provider.

8.5.6 Data Security and Licensing

Often, applications leverage sensitive data from users or other sources in their delivery of outcomes. When sensitive data are composed into prompts as context and then passed to an LLM to elicit a response, there is a data security or privacy risk since fully safeguarding against the LLM response containing that sensitive context is difficult to achieve. Many solutions to this problem apply a *generate then filter* approach, where sensitive data are scrubbed from LLM responses before being served to the user. Similarly, alignment methods, such as those surveyed in Chapter 5, can

be used to minimize the risk of sensitive or undesired information being served to users.

In the context of closed-source or 3rd party hosted LLMs, additional data security risks arise as data are passed over the network to the LLMs API, and in the case of proprietary LLMs, how that data is retained and used by the model owner. In the early days following ChatGPT’s release to the public, how OpenAI leveraged the sudden influx of user queries and interactions within its service was a hot topic. Many users voiced concern that their prompts, many of which could contain private or sensitive information, would be leveraged by OpenAI to improve ChatGPT further and that this would then expose that private or sensitive data to other users. Given the rapidity with which the popularity of ChatGPT grew, there was a lack of clarity on this issue. The various reports of jailbreaks that aimed to extract “training data” from LLMs created significant skepticism among potential users in industries where data privacy and security are of the utmost importance (e.g., Healthcare and Financial Services) (Yu et al., 2024).

In the context of licensing, there are two key aspects that application developers should be aware of when planning their application design, including maintenance of the application. The first aligns with traditional software and tooling licensing in that the license with which an LLM is released often dictates the scope of their application. For instance, some LLM licenses might only allow for research or experimental usage, while others might make the marketing or promotion of your application more complex than you had anticipated. While a deep-dive of the different types of licenses and their implications for application developers is beyond the scope of this section, interested readers can review valuable resources on the issue⁷.

The second licensing consideration for LLM application developers should be concerned with is that of the data used during pre-training. There is mounting evidence that at least some of the data leveraged to train these models might fall under copyright restrictions that are effectively violated through their inclusion in training LLMs. One of the highest profile instances of this unfolds in the US course between OpenAI and The New York Times. In this case, the Times claims that OpenAI illegally incorporated the media giant’s content into its LLM, violating its rights (S.D.N.Y., 2023). While such cases make their way through the courts, application developers need to anticipate the impact that decisions will have on their application’s maintenance since rulings in favor of the copyright claimant may require the model owner to withdraw any model trained on the copyrighted material and release updated versions. Application developers will be effectively forced to update the LLM within their application, which might involve additional testing to ensure consistent performance.

⁷ <https://github.com/eugeneyan/open-l1ms>

8.6 Tooling for Application Development

Since the explosion of LLM innovation, a commensurate explosion in the tooling ecosystem has occurred. Many of these tools are specialized in specific stages of development, such as fine-tuning LLMs or optimizing prompts. In contrast, others are more feature- and capability-rich, delivering value in many stages of development. Navigating this ecosystem is a daunting task for those unfamiliar with its evolution. The scale innovation is represented in Table 8.3⁸, where a sampling of the most popular tools is listed.

Table 8.3: A non-exhaustive list of tools that form the supporting ecosystem for building and deploying LLM-based applications.

Role in LLM Applications	Popular Tools
Embedding & Indexing	Pinecone, Weaviate, Milvus, Chroma
Data Annotation	Scale, Snorkel, Label Studio
Training, Development & Evaluation	HuggingFace, Lightning AI, HumanLoop
Experiment Tracking	Weights & Biases, MLflow, Comet
Privacy, Safety & Compliance	Lakera, Skyflow, Nomic
Monitoring & Observability	TruLens, WhyLabs, Arize
Hosting & Deployment	Lambda, Together.ai, Groq, Predibase, Anyscale
Prompt Chaining & Integration	LlamaIndex, LangChain, DSPy

In this section, we aim to guide the reader through this ecosystem in a functionality-based way. Initially, we highlight some important tools that aim to be the glue in LLM application development. These tools typically leverage the concept of chains, wherein interactions with an LLM or multiple different LLMs and any other component within the application design are modularized and sequentially linked together in a chain to enable rich workflows and user interactions. Next, we look at tooling for customizing LLMs. We explore libraries for pre-training, adaptive pre-training, and fine-tuning specifically. Highlighting the more popular libraries, as well as those offering unique capabilities. After this, we discuss prompt engineering and the various options during this stage of application development. Then we review some vector database options available to developers, mentioning some tools that integrate these tools conveniently. Finally, we provide some insights into the application evaluation and monitoring aspects of application development.

8.6.1 LLM Application Frameworks

LLM application frameworks provide the glue that ties the often numerous components of LLM applications together. These tools are typically quite prescriptive in their approach to LLM application development, so it is important to choose one that

⁸ Reproduced with modifications from <https://github.com/langgenius/dify>.

matches a pattern that meets your particular application needs. In terms of feature functionality, some frameworks are richer than others. As an example, consider the three frameworks compared across eight features in Tab. 8.4, where only Dify.ai supports enterprise features such as single-sign-on (SSO) integration. If your application has this requirement, choosing this framework might be a better option than building your own SSO on top of an application developed with LangChain.

Table 8.4: Feature comparison across three popular LLM application development frameworks.

Framework Feature	Dify.ai	LangChain	Flowise
Programming Approach	API + App-oriented	Python Code	App-oriented
Supported LLMs	Rich Variety	Rich Variety	Rich Variety
RAG Engine	✓	✓	✓
Agent	✓	✓	✓
Workflow	✓	✗	✗
Observability	✓	✓	✗
Enterprise Features	✓	✗	✗
Local Deployment	✓	✓	✓

One of the earliest and most popular frameworks is *LangChain*⁹, an open-source project focused on helping developers get their ideas to production faster and more reliably. It is centered around the Python programming language, which is also the most mature language for interfacing with frameworks such as HuggingFace, thus representing a good option for applications being developed in this language. A key advantage of LangChain is the extensive control and adaptability that it provides. Typically, frameworks such as these are most valuable when your application has complex interactions with the LLM and user inputs. *LlamaIndex*¹⁰ is another applications development framework that aims to provide developers with sophisticated functionality that they can interact with and leverage at a relatively high level. As shown in Chapter 7, LlamaIndex has extensive RAG application patterns and components, and so is a great option if your application leverages this paradigm. However, it is just as good an option as LangChain for general LLM application development.

Another attractive LLM application development framework is *Flowise*¹¹. This framework, built in JavaScript, enables users to customize their LLM flows visually using drag-and-drop elements within a graphical interface. No-code flow design and development can be particularly attractive for rapid prototyping or for projects with a preference for low/no-code development. Its close alignment with the JavaScript ecosystem enables rapid web application integration, lowering the time-to-value for projects where timely go-to-market is critical.

⁹ <https://github.com/langchain-ai/langchain>

¹⁰ <https://www.llamaindex.ai/>

¹¹ <https://github.com/FlowiseAI/Flowise>

In contrast to these centered around high-level sequential workflows, *DSPy*¹² takes a different approach. *DSPy*, rather than providing pre-built high-level functionality to users, instead offers lower-level modules that one could leverage to achieve similar functionality as that pre-built LangChain modules. One of the most interesting features of *DSPy* is its compiler, which can automatically optimize multi-step pipelines to achieve the highest quality on a pipeline’s tasks. This automatic optimization process can be a beneficial pattern for applications that are expected to change regarding data inputs, control flow or execution sequencing, and even the LLM you leverage. This option, because it does not provide functionality through features like a pre-built prompt template, is a good option for those who require low-level control over their application’s interactions with LLMs.

The final application development framework to highlight is *Dify.Ai*. As shown in Tab. 8.4, this framework is a more full-featured platform with capabilities more aligned to enterprise-grade application development. This enterprise-level focus is reinforced by its alignment with LLMOps, wherein many patterns are explicitly leveraged to ensure rigor, reliability, and reproducibility in development and application behavior. These patterns are essential for enterprise applications, where the costs of application failures or inefficiencies are often much greater, both from a monetary perspective and due to potential societal consequences (e.g., LLMs leveraged as trading agents within financial services).

Now that we have explored platforms that assist in the development process, in the subsequent sections, we will explore the various tools that can be leveraged within the frameworks to achieve more specific application development tasks, such as prompt engineering, LLM customization, and evaluation.

8.6.2 LLM Customization

As discussed previously, the customization options for closed-source and open-source LLMs are very different. The customization of closed-source LLMs typically occurs through prescriptive processes defined by the model owner, whereas open-source LLMs can typically be achieved using any technically feasible method that the user wishes. The technical complexity of each of these options is very different in that closed-source customization can be a great option for application development teams without deep technical expertise in machine learning, while open-source customization is not for the technical faint of heart.

OpenAI enables fine-tuning of its foundational models through APIs, which can be interacted with through their various *Software Development Kit* libraries (SDKs). The fine-tuning process in this context involves steps for dataset preparation, which includes formatting the fine-tuning dataset according to OpenAI’s Chat Completions API standards. Train and test splits are required to enable evaluation-based fine-tuning. These data are then uploaded through the OpenAI API to trigger fine-

¹² <https://github.com/stanfordnlp/dspy>

tuning. Within the fine-tuning process, users can experiment with hyperparameters and data quality/formatting to iteratively improve the fine-tuned LLM outcomes. It is this iterative process that application developers should consider carefully since costs are based on a combination of factors, including the number of training tokens used, the base cost per token for the particular OpenAI model being fine-tuned, and the number of training epochs.

Options for open-source LLM customization are much broader, resulting in much higher complexity. If this complexity is not well understood, the cost of fine-tuning in this context could increase significantly as excess compute costs accrue through experimentation. Cloud services are available that manage LLM computing for developers, such as *AWS Bedrock* or *Google Vertex AI*. While self-managed options are also available, such as *AWS Sagemaker Jumpstart*. In this setting, the fine-tuning or alignment toolkit leveraged is up to the developer for the most part. Tools like `pytorch`, wrapped by higher-level tooling such as HuggingFace `transformers` and HuggingFace `PEFT` are the mainstay of this LLM customization path. However, more and more specialized tooling, centered around fine-tuning complexity, compute, and cost efficiency are emerging, such as `ggml` and `LLMZoo`. For more details on LLM customization, readers are encouraged to revisit the tutorials for Chapters 4 and 5, where their usage is also demonstrated.

8.6.3 Vector Databases

In the early days, when language models were beginning to grow increasingly powerful, semantic similarity quickly emerged as one of their prominent uses. By applying a model to two chunks of text and comparing their embeddings, it can be ascertained whether or not the inputs have similar meanings. Often, this is done using the cosine distance between the two embeddings. However, suppose there is a need to find the most similar text across a large knowledge base. Applying a model and calculating the cosine distance between millions of vector pairs would take too long. The solution is to pre-compute all of the embeddings for each chunk of text and then store them in a *vector database* from which they can be efficiently retrieved.

One of the earliest successes in large-scale vector search was Facebook’s *FAISS* (Johnson et al., 2017), an open-source library of indexing techniques. Numerous vector database solutions have emerged since then, including *Pinecone*, *Milvus*, and *Chroma*, to name a few. Vector databases are designed to optimize both the storage and the retrieval aspects of vector search (Schwaber-Cohen, 2023). Preexisting database technologies such as *Postgres* and *Cassandra* have also begun to enable vector storage capabilities to keep up with the trend.

The use cases for vector databases are wide-ranging. One of the most common is RAG (Chapter 7). Very often, a RAG application’s “retrieval” step must act on a vector database to locate the necessary information to respond to the user. Another important use case is QA. For example, a customer might have a common question about a product but cannot locate the answer anywhere on the company’s website. If

other users have asked similar questions, then there is potentially an answer that can be reused without a human needing to look it up again. More generally, vector search can often be a powerful complement to traditional keyword searches. Keywords provide predictable returns and high recall, while vector based searching expands the range of potential documents that can be retrieved in the search, making a combination of the two techniques in a single search effective.

8.6.4 Prompt Engineering

As we learned in Chapter 3, prompts can range from the most basic cloze and prefix styles that are more suited to masked language models to prompts that have been optimized in continuous space for generative models such as GPT-4 or Llama-2. If your interest is in theory and methodology for prompt engineering, those chapters will be most relevant. This section will highlight some of the most practically valuable tools for developing and maintaining prompts in your application development project.

As mentioned in this chapter, there has been an explosion not only in LLM development, but also the tooling ecosystem surrounding their direct use and integration into applications. This explosion has created a challenge for developers because the quality of these tools is often unknown until the point of usage. Rather than providing a comprehensive survey of all tooling available for the prompt engineering tasks, instead, we recommend that readers explore options from <https://www.promptingguide.ai/tools> and <https://learnprompting.org/docs/tooling/tools>. That said, next, we will highlight some of the more popular prompt engineering tools to provide a sense of the type of functionality one can expect and some of the different approaches available for prompt development.

To help situate the usage of these tools, Fig. 8.5 illustrates a typical process for prompt engineering within the higher-level context of application development for production (Benram, 2023). Typically, prompt engineering and refinement are performed by leveraging several evaluation criteria, such as analytic performance on a benchmark or test dataset, and qualitative alignment to stylistic requirements. Similarly, how prompts are integrated into applications and passed to the LLM itself, how they are stored and maintained, are all functionality within the purview of prompt engineering tooling. Next, we will explore some of these tools.

As discussed in Chapter 3, prompt design can be a straightforward manual process or a complex automated optimization process. Starting simply with a manually designed prompt template is typically a good idea. Tools such as *OpenAI's Playground*¹³ can be extremely useful for such a task. This tool provides several useful features for exploring important aspects of prompting capable LLMs. For instance, users of the OpenAI Playground can easily swap between different OpenAI LLMs to explore how well a given template generalizes across them. Similarly, the interaction between prompt designs and LLM hyperparameters such as `temperature`,

¹³ <https://platform.openai.com/playground>

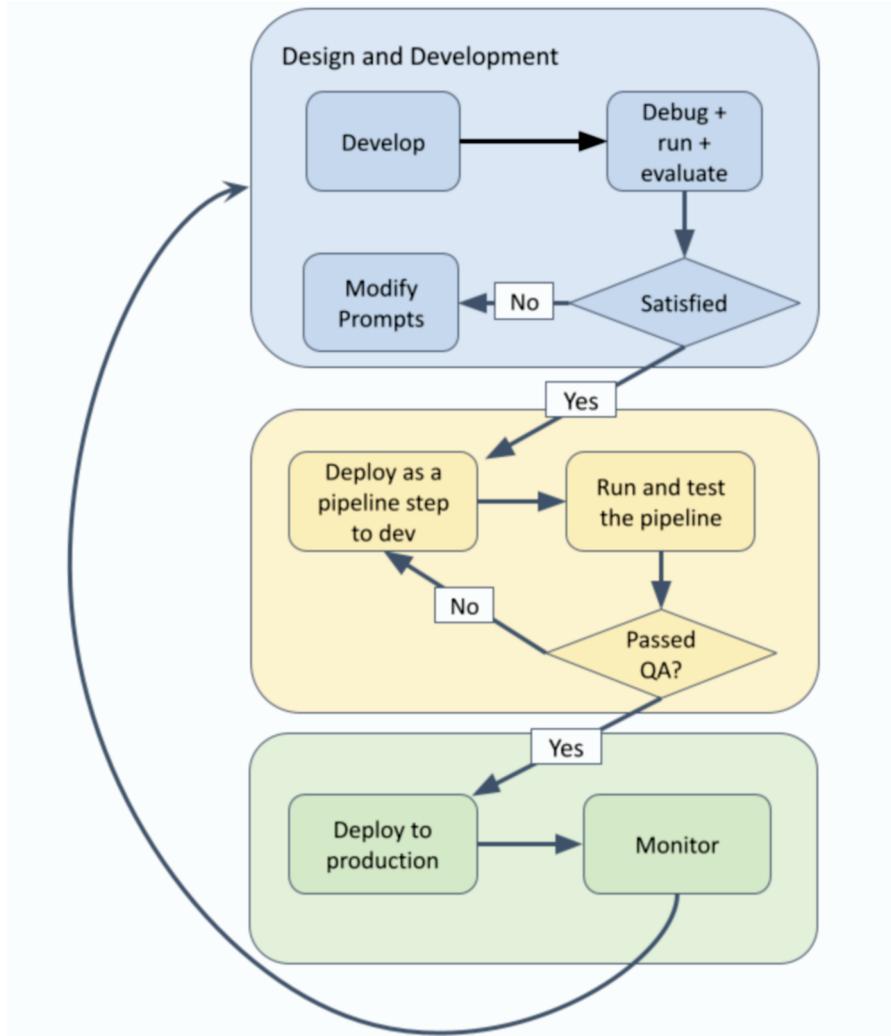


Fig. 8.5: Overview of the prompt engineering workflow.

which acts to select only the most likely tokens during sequence generation when its value is low and introduces increasing randomness into token selection as its value is increased, can be explored allowing users to understand how these LLM settings might influence better or worse responses for a given prompt (Saravia, 2022).

Another prompt design and optimization tool is *Promptmetheus*¹⁴. This tool is a rich-featured prompt engineering IDE with features that enable prompt composition, testing, optimization, and deployment. In addition, unlike OpenAI Playground,

¹⁴ <https://promptmetheus.com/>

which only enables interactions with OpenAI proprietary LLMs, Promptmetheus supports development against over 70 LLMs across proprietary and open-source domains. As Promptmetheus is intended to be more of a development tool, it has superior experimentation tracking features, as well as integration with other developer tooling such as *LangChain*.

```
from langchain_core.prompts import ChatPromptTemplate

chat_template = ChatPromptTemplate.from_messages(
    [
        ("system", "You are a helpful AI bot. Your name is {name}."),
        ("human", "Hello, how are you doing?"),
        ("ai", "I'm doing well, thanks!"),
        ("human", "{user_input}"),
    ]
)

messages = chat_template.format_messages(name="Bob", user_input="What is your name?")
```

Listing 8.1: ChatPromptTemplate construction example. Taken from https://python.langchain.com/docs/modules/model_io/prompts/quick_start/

LangChain is another sophisticated tool with features for prompt optimization and management. As discussed in Sect. 8.6.1, LangChain has many other invaluable features for building multi-component LLM applications. Its prompt development capabilities are popular in and of themselves, providing several convenient methods to users. For example, the ChatPromptTemplate method enables users to programmatically construct prompt messages to be passed to an LLM as shown in Listing 9.8. This code results in a multi-turn chat conversation that can be conveniently passed to an LLM using one of LangChain’s many methods.

A final option to consult when designing prompts are existing prompt libraries. These resources are collections of useful prompts and design/formatting options that can typically be leveraged with minimal modification. An example of a prompt library with comprehensive coverage over many tasks and LLM interaction patterns is *Prompt Hub*¹⁵. This library contains prompts for sentiment classification, SQL query generation, poetry generation and entity extraction, for example. However, While there is no guarantee that a given library will contain exactly the prompt needed, it will almost certainly contain examples that you can use as inspiration and guidance to construct your own. They are valuable for getting started with prompt engineering quickly.

¹⁵ <https://www.promptingguide.ai/prompts>

8.6.5 Evaluation and Testing

In any ML application, it is important to evaluate the performance of a model before deploying it. This is quite straightforward in simple scenarios where a ground truth exists, but there is usually no definitive correct output in generative applications. This leaves the problem of evaluation open to subjective interpretation. A common pattern widely adopted is creating an automated evaluation procedure by having the LLM itself (or another more powerful LLM) judge the output quality. This approach initially requires the creation of test cases that cover a diverse set of prompts. Then, the LLM is applied to generate responses for each test. The outputs can be assessed on a multitude of different criteria, including but not limited to:

- Relevance
- Comprehensiveness
- Groundedness
- Hallucinations
- PII content
- Sentiment
- Toxicity

Tools such as *TruLens* streamline this process by supplying built-in prompts for many common evaluation needs (Reini et al., 2024). This makes it possible, for example, to obtain a hallucination score for a response simply by making a function call in a single line of code. This framework can also be extended in cases where a specific application warrants its own custom evaluation methods.

8.7 Inference

When designing a system to serve an LLM-based application, numerous decisions must be made regarding the approach to model inference. Generally, most of these decisions involve optimization along the key dimensions of cost, speed, and model performance. A 70 billion parameter model will provide very high-quality outputs. Nevertheless, this model will be far slower than a 7 billion parameter model unless one is willing to spend heavily on computing infrastructure. In this section, we explore various approaches for balancing these inherent trade-offs. We will also discuss some important factors that may vary from one use case to another.

8.7.1 Model Hosting

Perhaps one of the most fundamental decisions for an LLM application is the location where the model itself is hosted. The choices can be categorized as follows:

- **Sending inference requests to a public third-party API:** This fast and straightforward approach is common for building demos and prototypes, as well as for quickly getting new concepts into production. No setup or maintenance is involved, and developers can learn to use these APIs without any deep knowledge of how LLMs work. This approach can result in significant savings by reducing the effort and expertise required to deploy an application; however, API usage itself comes at a relatively high cost and may easily negate those savings if there is a large volume of inference requests to serve. There are several other significant limitations to consider as well. First, this approach offers little to no ability to tune or otherwise customize the model to the needs of a specific use case. It does not provide strong guarantees on latency, and as is typical with public APIs, rate limits must also be accounted for. Finally, as discussed in Sect. 6.4, these API calls mean that the data coming through the application are being shared with a third party. For many organizations, this last point is an absolute deal-breaker. While there are many drawbacks, it is also worth noting that OpenAI's latest GPT models are currently available only through their API. For applications where the value to end users is maximized by taking full advantage of the best-in-class capabilities offered by OpenAI, the potential trade-offs may be well worth it.
- **Using a foundation model hosting service:** The three major cloud computing providers offer services, for instance, AWS Bedrock, that makes foundation models readily available within a private and secure environment. For several reasons, this approach scales far better than public APIs. First, while the service providers include many built-in optimizations to the inference process, the owners of an account also have a level of control over the quantity of GPU resources dedicated to any given model. This allows them to find the ideal balance of inference speed and compute cost, both of which are outside the control of API users. Additionally, network issues can be greatly alleviated by assuming that the LLM resides within the same cloud environment as the rest of the application. The inference requests and responses will be less affected by fluctuations in bandwidth, and the environment can be configured to ensure that the model is in the same physical location as the application. In cases where latency is a significant factor, sending requests to an API that might reside halfway around the world can pose problems. There are, however, still some limitations to these model hosting services. Their optimization of computing usage forces them to remain somewhat confined to a fixed set of foundation models and tuning techniques. This optimization also comes at a premium price, which may not be worth it for organizations with the internal expertise to run their own GPU computation.
- **Self-hosting the model on your computing infrastructure:** In cases where a service such as AWS Bedrock is too limiting, the best choice may be a custom-built runtime environment. This provides maximum flexibility to use any desired LLM and optimize it precisely according to the application's needs. However, it is also more complex than the two options presented above. NVIDIA's Triton

inference server¹⁶ is one option that can reduce effort. It provides a significant range of flexibility in model architecture choices while managing many low-level GPU optimizations. For many organizations, employing or contracting a dedicated team of people with deep knowledge of tensor operations is not necessary to build a highly customized inference system. However, this can become cost-efficient if inference demand reaches a massive scale.

8.7.2 Optimizing Performance

Anyone who has ever tried using LLMs in a CPU setting is probably quite aware of how slow they are to respond without adequate GPU computing power. Because computing is costly, several techniques have emerged to process more inference requests faster without adding more hardware to the equation.

Two key concepts related to inference speed are *latency* and *throughput*. Latency refers to the time it takes to process a request and send a response to an application, whereas throughput is the volume of output that can be produced by the LLM in a given timeframe (Agarwal et al., 2023). While these two concepts are closely related, they are not the same. Consider a coding assistant as an example. When users start typing, they expect suggestions to appear almost instantly. This would be an example of an application that would require low latency. Alternatively, imagine a service that filters spam emails. In this case, the user will likely experience any impact whether the spam classification takes half a second, several seconds, or perhaps even longer. However, throughput may still be important in this application. If the service cannot keep up with the influx of new messages, it will fall further behind and fail to deliver the intended benefit.

8.7.2.1 Batching

The optimization of *batching* is a critical factor in maximizing throughput (Fig. 8.6), as combining multiple inputs into a single matrix capitalizes on the performance benefits of vectorization. In general, larger batches are more efficient than smaller batches. This is fairly easy to manage in an application where most inference requests contain large volumes of input data. However, in cases where the user sends single inputs on each request, such as in conversational applications, the only way to take advantage of batching is to combine inputs from multiple users into a single batch. This approach is called *dynamic batching*. This improves throughput but can hurt latency since a user may have to wait for other requests to be accumulated before processing their request.

One solution to the problem described above is *continuous batching*, a method designed specifically for autoregressive LLMs. As generative text models produce

¹⁶ <https://github.com/triton-inference-server>

tokens iteratively, a new user input can be added to a batch of other inputs already in process. When a string of output tokens is completed, meaning that either the maximum length is reached or a stop token is generated, an input slot becomes available in the batch. Then, the next user request in the queue can be inserted into the batch. In this way, the system can begin processing incoming requests as soon as GPU memory becomes available while at the same time, never under-utilizing the GPU by having it process smaller than optimal batches. Furthermore, it naturally accommodates inputs of widely varying lengths without incurring the overhead of excess padding tokens. Since GPUs are highly specialized in large matrix operations, their performance is maximized when the input sizes and shapes are consistently well-matched to the hardware architecture.

8.7.2.2 Key-Value Caching

Key-value caching is another useful inference technique that can be applied to autoregressive LLMs. After each token is generated, it is added to the end of the sequence and fed back into the model to produce the subsequent token. Because all of the previous tokens are the same as before, there is no need to recalculate all of the attention weights in every iteration; instead, they can be cached and re-accessed each time. In

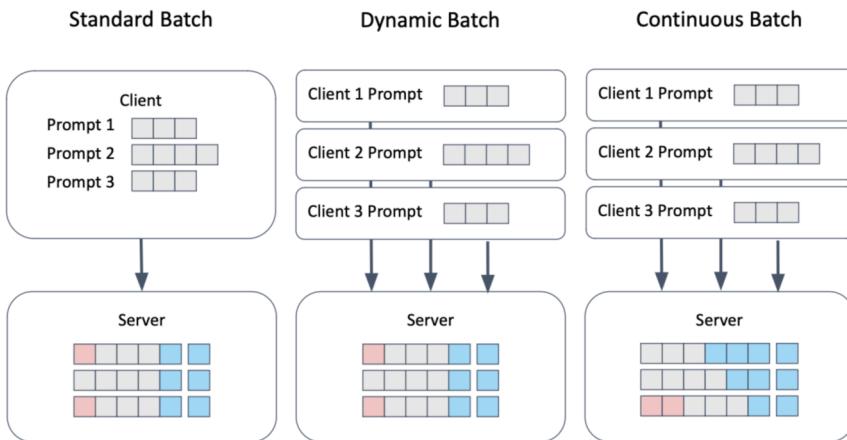


Fig. 8.6: In traditional inference architectures, it is largely up to the client to create batches. Particularly in applications where users send one request at a time, the GPU can be much more effectively utilized by dynamically aggregating multiple inputs on the server. This comes with a latency cost in waiting for more inputs before a complete batch is formed and computation can begin. Continuous batching addresses this problem by putting newly received inputs into existing batches alongside other inputs already in progress.

this way, only the weights relating to new tokens must be computed. The attention mechanism, the Transformer component with the highest order runtime complexity, is often the largest performance bottleneck in the architecture. The ability to scale down these computations can considerably increase the inference speed.

8.7.3 Optimizing Cost

Even when using all available techniques for optimizing inference speed, the largest and most powerful models still require considerably expensive hardware. This is especially true if the application demands low latency and high throughput. It is almost always worth considering whether a smaller model could do the job equally or at least comparably. For some use cases, the customer base may be more limited by what they can spend than by the quality of the results. There will inevitably be a sweet spot along the continuum of minimizing cost and maximizing utility, and this needs to be carefully analyzed for any production application.

However, there is another dimension to the trade-off between model size and model results. Some of the cost savings associated with a smaller model could be applied toward fine-tuning to close the gaps in its capabilities. Part of a larger model's appeal is that it contains enough knowledge to perform well on a wide range of tasks, using only prompt engineering and in-context learning techniques. This is critical because fine-tuning those models is expensive, even with techniques such as LoRA. When a smaller LLM is selected, fine-tuning becomes much more viable. Predibase is one company that has staked itself on this notion. Their philosophy is that the optimal path for most applications is to use small, specialized models with as many adapters as necessary to suit each specific type of inference request.

8.8 LLMOps

With the surging interest in LLMs, it is only natural that *Large Language Model Operations* (LLMOps) has branched off as a logical extension of MLOps. Many challenges that arise when deploying LLM applications in a production environment are similar to the challenges with machine learning models in general. Here, we will focus primarily on these concepts related to LLMs. Nevertheless, much of this material will be familiar to readers with prior experience in operationalizing other models.

As is often the case when new ideas spread rapidly, MLOps and LLMOps are frequently thrown around as buzzwords, leading to disagreement on any precise definition of what they entail. For our purposes, rather than laying out an idealized system, we will offer a general view that encompasses a variety of tools and processes that enable ML capabilities to be deployed in a production environment. This includes the management of data and prompts, iterative training, and workflow orchestration

([Oladele, 2024](#)). Most of these methods aim to maximize efficiency, minimize risk, or perform both in tandem. This is crucial to deriving high value from new ML capabilities. Many people have fallen into the trap of wasting precious time with models that have, at best, only marginal benefit to end users and, at worst, may even have negative impacts.

8.8.1 LLMOps Tools and Methods

As the importance of MLOps, and subsequently LLMOps, has gained wide recognition, the market for solutions has rapidly grown. This has led to the development of many different tools and products. In the sections below, we will survey the landscape of the LLMOps ecosystem, explaining the various pain points that arise when building and deploying LLM-based capabilities and how those issues are commonly addressed. An overview of the types of tools involved and their interplay with a production application is illustrated in Fig. 8.7.

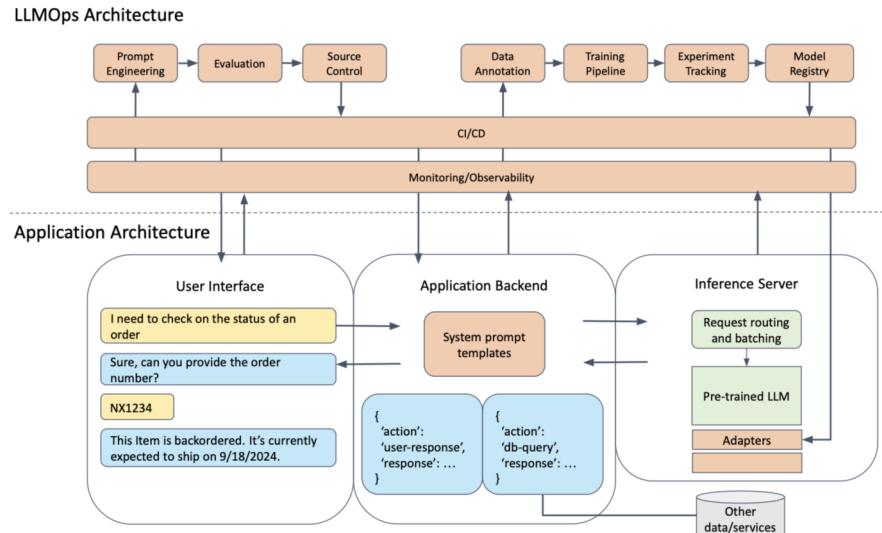


Fig. 8.7: A basic chatbot example with the corresponding LLMOps architecture. The prompt templates are developed through an iterative process and versioned in source control. The LLM in this case also uses adapters that have been trained for the specific needs of the application, thus requiring mechanisms for tracking experiments and promoting trained model components to production. Each of these moves into the deployed application through a CI/CD framework; production metrics are reported back to a monitoring service. The feedback can then be used to further improve the prompts and the training data.

8.8.1.1 Experiment Tracking

Training a model to the quality desired for production deployment is a highly iterative task. A significant amount of trial and error is usually involved, particularly in the early stages of developing a new capability. It can quickly become unwieldy to organize the results of various experiments and track which models perform best ([Jakub Czakon, 2024](#)). Beyond that, it might be important to recall other details later, such as the specific dataset used or the training time needed. A number of tools for *experiment tracking* have been designed to assist with all of these needs. Typically, all that is required is a few simple commands added to the code in the training pipeline, and all relevant information is then automatically captured. These tools are generally equipped with robust user interfaces, including a wide array of metrics and visualizations that enable experiments to be stored and analyzed. This is particularly useful for team collaboration when multiple people are involved in a project and want to see each other's work.

A standard companion to experiment trackers is model registries. A model registry is essentially a repository that stores all the models created through the experiment tracker, although models can also be uploaded directly without experiment tracking. Typically, a model registry allows custom tags to be applied to models. The tags can then be used by downstream processes to automatically identify which models are ready to deploy or to trigger other workflows.

8.8.1.2 Version Control

Using source control for any software project is a widely accepted best practice, and naturally, this extends to the use of LLMs as well. Code repositories such as `git` are generally used for LLM training and evaluation in much the same way that they are used for other types of code bases. However, there are also versioning needs that are not readily addressed with code repositories, including those described below.

- **Model versioning:** In building LLMs and LLM-based applications, many iterations of training and tuning are performed. It is essential to know which version of a model is put into production and to be able to trace back to the exact code and data that went into it. Otherwise, if the model does not perform as expected, debugging and determining what went wrong is challenging. It is worth noting that most of this comes for free when experiment trackers and model registries are employed. However, even when operating at a lower level of maturity without all of the most sophisticated tools available, model versioning in some form is always an absolute must.
- **Data versioning:** Oftentimes, training data are a component of a projects that evolves the most. It is not uncommon to spend a substantial amount of time determining what types of data are most suitable, and far less time working on code. If the data are not versioned, the model cannot be rolled back to a previous state. This effectively erases the history of the work that has taken place.

- **Prompt versioning:** Prompts and prompt templates are another critical part of an LLM system that can change considerably throughout the life of an application. It is quite common for prompts to be stored as part of the application code, but there are reasons why this may not always be the best approach. Prompt templates typically behave like probabilistic models rather than deterministic code; thus, the techniques used to validate them are often quite different from those used to test other code. Managing prompts separately can potentially simplify development, providing the ability to iterate quickly on prompt improvements without having to release and deploy a new version of a larger component each time.

8.8.1.3 Deployment

Many tools commonly used for continuous integration and deployment (CI/CD) in a mature software development lifecycle work equally well for deploying LLM capabilities. These processes aim to automate the construction and testing of new components as they are released. Several of the best practices that these systems enforce are as follows:

- The same battery of tests must run and pass each time a new version is released, thereby reducing the potential for regressions.
- All components are validated in a centralized environment, which typically mimics production, rather than being tested in an individual developer's environment.
- The build and release process is designed to be fully automated and repeatable, intending to eliminate any possibility that a manual misstep could cause the deployed version to differ from the tested version.
- A deployed component can be expeditiously rolled back to a previous version if it does not perform up to standard in production.

This type of system can be of tremendous value in automating model evaluations and reporting metrics. The system then serves as a quality gate to prevent a low-performing model from mistakenly being deployed to production.

8.8.1.4 Monitoring

Many tools offer the ability to monitor the performance of deployed models. Generally, this involves applying real-time evaluation techniques and aggregating relevant metrics. Alerts may be triggered if the model is not behaving as expected initially or has changed over time. For instance, if the generated outputs are trending shorter or longer than previously, it could indicate (among other things) that user behavior has shifted. It is worth investigating whether the model or other components, such as prompt templates, need to be adjusted accordingly. Beyond just monitoring LLM

performance, monitoring tools can safeguard against risks such as model hallucinations and prompt injection attacks ([Ama, 2023](#)).

A closely related concept to monitoring is observability. These two terms overlap and are often used interchangeably, and most LLMOps solutions on the market treat them jointly. The rough distinction is that monitoring aims to identify issues or areas for improvement in a system based on its aggregate performance. In contrast, observability encompasses more mechanisms to allow for deeper investigation. For example, a monitoring tool may increase awareness that LLM response times are longer than normal on a given day. However, without adequate observability, it could prove difficult to determine why this is happening. With observability tools in place, it is possible to isolate individual inputs and trace them through the system step by step to locate where bottlenecks or failures occur.

8.8.2 Accelerating the Iteration Cycle

In previous chapters, we discussed the importance of human feedback for improving model quality. This is not only the case for alignment or instruction tuning of a foundation model; incorporating user feedback is similarly valuable at the application level. The faster a team can capture results from its production system and use them to build and release updates, the more value it can deliver to its customers. Optimizing this workflow also allows for rapid response to unforeseen issues that may arise.

Another consideration that favors short model deployment cycles is that there is often no substitute for user feedback ([Burgess, 2021](#)). Extensive testing by data scientists and developers may help remediate many model flaws. Nevertheless, until it is applied to real user input data, there are no guarantees that the expected results will be achieved. Many teams have learned this the hard way, investing long months or even years of effort into a new technology only to flop when it goes to market. One way to avoid this pitfall is to obtain a minimally viable model out the door as quickly as possible, with the assurance that the necessary infrastructure is in place to quickly update or rollback the system as new data become available. An example of this occurred shortly after the release of ChatGPT, when major competitors such as Google and Microsoft felt pressure to make their chat capabilities available as quickly as possible. These models were immediately attacked by curious users who found amusement in the model's quirky responses, revealing some problematic tendencies ([Orf, 2023](#)). However, those companies moved quickly to overcome their initial issues and ultimately suffered minimal damage to the perception of their products.

8.8.2.1 Automated Retraining

In certain situations, it may be feasible to directly update the model using examples from the production input data. For example, many platforms allow users to flag content they like or dislike. This type of feedback can be directly incorporated into a labeled dataset for the next iteration of training. Assuming that the production model is reasonably mature, acquiring new data, running the training pipeline, validating the results, and deploying the new version could be fully automated. This is worth striving for in applications that adapt quickly to emerging trends; however, it is not easy to achieve. An inadequately trained model could find its way into production if insufficient controls exist. The risks and the effort required to mitigate those risks before committing to fully automated training should be considered immediately. Reaching this level of maturity is likely to involve a significant long-term investment in LLMOps capabilities.

8.8.2.2 Human-in-the-Loop Processes

When automated re-trains are infeasible or unnecessary, other methods exist to drive efficiency. If data need to be reviewed or annotated by human experts, numerous labeling tools can be used. Some also use active learning or semi-supervised techniques to accelerate the labeling process if desired. In previous chapters, we discussed how RLHF led to game-changing increases in LLM performance and proved worthy of the costly manual labor needed. It is difficult to overstate the impact of collecting or generating high-quality training examples that directly target a model's weaknesses and that higher quality generally correlates to greater human effort. For organizations that cannot staff adequate personnel for their annotation needs, there is also the option to outsource the work. Countless companies specialize in this area.

While it is often true that more recent or more robust data will immediately lead to an improved production-ready model, this is not always true. The model architecture may prove to be the limiting factor, requiring further exploration and research to address the weaknesses in the application. Experiment tracking and version control are highly beneficial here, especially if multiple people work on the same problem from different angles. More manual work means greater potential for results to be misplaced, datasets to be altered, or any number of other mistakes. It is also a good practice to establish a representative golden dataset for evaluation purposes and keep it fixed throughout an iteration cycle to allow valid experiment comparisons.

8.8.3 Risk Management

There are many inherent risks in using LLMs, or any ML model for that matter, to assist with tasks that traditionally require human effort. Consider the potential damage to a company if it is found to be using AI to deny people housing or employment

on a discriminatory basis. Deploying LLMs in these types of settings requires extra care in evaluating and minimizing risk.

Risk Assessment and Mitigation

Conducting thorough risk assessments in the context of an LLM-powered application's functionality is an essential step in the path to production. Consider the impact of the following risks:

- **Privacy Violations:** In the course of normal user activity, does the application handle protected data that could end up being unintentionally presented to the user?
- **Security Breaches:** Are there vulnerabilities in the application's design and functionality that could result in data loss? In the world of LLMs, consider new risks such as prompt injection attacks, as well as traditional application security.
- **Bias and Harmfulness:** The risks of generating outputs that are biased or harmful should be well understood. Some domains of application will have explicit laws or regulations around these challenges, so special care is needed where this is the case.

Once a comprehensive risk assessment has been completed, it may be necessary to implement mitigations. These may include the following actions:

- **Human-in-the-loop Oversight:** In many low-volume or highly sensitive/risk use-cases, it may be necessary to target “expert human in the loop” actionability. This effectively means that the user of the application is able to verify the LLM’s outputs before further actions are taken.
- **Content Filtering:** The implementation of content filtering policies within the application may be necessary. These policies can be implemented through alignment tuning or via post-generation filtering, using keyword, pattern-matching or classification-based approaches to ensure that only appropriate outputs are presented to the user.
- **Access Control:** While common in traditional application development, LLMs in applications may introduce further role-based considerations. As an example, prompt template operations might have to be limited to only those developers within the *Prompt Engineer* role rather than a generic *developer* role.
- **Continuous Monitoring:** Ensuring that user behavior and interactions are baselined, and monitored on an ongoing basis is an important capability in all production applications. In the context of LLMs, understanding usage, outputs and performance can help identify issues early.

8.8.3.1 Model Governance

We have seen in previous chapters that several challenges persist with LLMs, with bias being one of many. Understandably, organizations generally exercise an abundance of caution when using ML for any purpose that is subject to legal or regulatory requirements. In particular, this applies to most areas of medicine, finance, and law. For technologists working on these types of use cases, it is important to proactively consider what requirements must be met to convince stakeholders that the benefits of LLMs outweigh the risks. Many organizations have standards to ensure that production models have been adequately validated and documented. Model explainability may also be critical. The effort to meet those standards can be deceptively high, resulting in delays and added costs if not appropriately factored into the project timeline.

One of the most popular patterns used to address model governance is the concept of model cards ([Mitchell et al., 2019](#)). This standard encourages transparency from model developers to reduce the risk of models being used for purposes other than those intended, and the information is presented in a way that makes it easily accessible for anyone using the model. Not all elements are relevant to all models, but ideally, a good model card should highlight characteristics such as recommended usage, known limitations, and potential biases in the training data. Model cards may also contain information on the training process and evaluation metrics on various benchmarks. Nevertheless, they are generally concise and do not include many technical details.

8.8.3.2 Data Governance

When LLMs began to rapidly rise, one of the key drivers was the massive quantity of data scraped from the web. As these datasets explode, it becomes increasingly difficult to curate or filter out specific data types. However, once LLMs entered the mainstream, tension began to emerge as more people realized that their data were being used in ways they had never consented to or even imagined. LLM developers must take these concerns seriously to protect their organizations from legal challenges.

First, checking the terms and conditions when extracting data from sites such as social media channels or message boards is a good idea. Furthermore, the rules governing the use of some data might be ambiguous. Or, there is the possibility that it might be subject to future scrutiny even if it seems acceptable to use at present. For this reason, it is advisable to track data provenance. This means preserving knowledge of each dataset's source and which models were trained on those sources. Then, if the use of any data ever comes into question due to privacy, copyrights, or other concerns, it is possible to perform damage control. The dataset can be purged from storage, and models can be trained without it going forward.

8.9 Tutorial: Preparing Experimental Models for Production Deployment

8.9.1 Overview

In this tutorial, we revisit the experimental models produced in the Chapter 4 tutorial. However, this time, rather than focusing on the training process, we look at some of the steps we might take if we were preparing to deploy one of these models into a production application. Several of the tools and techniques discussed throughout this chapter will be applied and demonstrated. However, we continue to operate entirely within a Colab notebook environment with the understanding that many readers probably prefer to avoid the cost of deploying an actual production-grade inference capability.

Goals:

- Take an open-source evaluation tool and an open-source monitoring tool for a trial run.
- Explore the available capabilities in these tools and how they can be useful.
- Observe whether any new characteristics of our models are revealed through this process which might impact whether they are fit for production deployment.

Please note that this is a condensed version of the tutorial. The full version is available at <https://github.com/springer-llms-deep-dive/llms-deep-dive-tutorials>.

8.9.2 Experimental Design

This exercise will focus on several key factors that merit consideration when endeavoring to take LLM capabilities from concept to production. To set the stage, we assume a scenario in which two candidate models emerged from our work in the Chapter 4 tutorial. We aim to compare their relative strengths and weaknesses to determine which best suits the needs of our hypothetical application while also considering whether any computational bottlenecks can be addressed to control inference costs. We then consider the longer-term implications once our selected model is deployed, demonstrating how we can ensure that it continues to serve its purpose without any unforeseen consequences.

First, we will look at model evaluation, which is important in fully vetting any model's behavior before putting it into operation. In Chapter 4, we evaluated our models by manually prompting GPT-4 with a grading rubric. Here we take a similar

approach but instead using an open-source tool called TruLens (Reini et al., 2024). It offers an extensible evaluation framework along with a dashboard to compare metrics across models. There are a variety of similar solutions on the market, but TruLens has the advantage of being free, whereas many others do not.

Next, we briefly examine the inference speed of our models. In practice, we might want to benchmark performance on different GPU architectures, and consider various optimizations for each before we would have a real understanding of the cost of running a given model. However, for this exercise, we will simply look at how our models are operating on our Colab GPU.

To conclude the tutorial, we construct a scenario in which our model has been deployed in production for some time. We now want to see whether it is still behaving as anticipated or whether anything has changed in our system that may affect the model's performance. To illustrate, we deliberately manipulate some test data to create a trend of increasingly long user prompts. For this final portion of the exercise, we use another free, open-source tool called LangKit (WhyLabs).

8.9.3 Results and Analysis

We begin by demonstrating the `trulens_eval` library using a small portion of the TWEETSUMM test set. TruLens performs evaluation using feedback functions. There are options to use both built-in and custom functions to evaluate models. For this exercise, we choose the coherence and conciseness stock feedback functions. Under the hood, TruLens wraps other APIs such as OpenAI and LangChain, providing developers with several options for which provider they wish to use. Metrics such as conciseness are obtained through the use of prompt templates.

```
f"""{supported_criteria['concreteness']} Respond only as a
number from 0 to 10 where 0 is the least concise and 10 is
the most concise.""""
```

Listing 8.2: An example of a system prompt template provided for TruLens evaluations.

We observe the mean scores below by applying both our DistilGPT2 and Llama-2 LoRA models to the test sample. TruLens uses a scoring system that ranges from 0 to 1 for all metrics. As expected, the larger Llama-2 model performs better across the board. However, we further note that while the coherence and conciseness scores seem fairly reasonable, the summary scores are perhaps slightly low - especially for DistilGPT2. We can recall that these models appeared to perform quite well in our earlier tutorial. It is likely that part of the reason for this is simply that we did not invest much time into the design of the prompt template within the custom evaluation that we wrote for this exercise. The coherence and conciseness evaluations are built on validated prompt templates that are held up against a set of test cases by the developers of TruLens. This example is a good illustration of how difficult evaluation can be, and why it can be so valuable to leverage tried and tested solutions.

Table 8.5: Results of evaluating two candidate models with TruLens. Coherence and Conciseness are built into the tool, while Summary Quality is a custom evaluation that we provide.

Model	Mean Coherence	Mean Conciseness	Mean Summary Quality
DistilGPT2-finetuned	0.66	0.80	0.29
Llama-2 LoRA	0.80	0.83	0.60

There are distinct advantages to having a standard format for evaluation that leverages existing prompts where possible rather than building them all from scratch. First, it can potentially save time when designing the evaluation methodology. However, defining these types of abstraction also enables more seamless automation across various aspects of the LLMOps system. For instance (although we do not simulate this in our example), TruLens offers the ability to plug into an application such that user inputs and model outputs are evaluated in flight for real-time feedback.

We then shift to another freely available LLMOps tool called LangKit. LangKit is part of a software suite from WhyLabs that offers monitoring and observability capabilities. An interesting feature we will explore is the ability to analyze trends in prompts and responses over time. We simulate this by creating two separate data batches, or profiles, and comparing them. We break the data into two small sets consisting of longer inputs and shorter inputs to create variability in the profiles. Then, we link to the WhyLabs dashboard, where we can explore many useful metrics in detail.

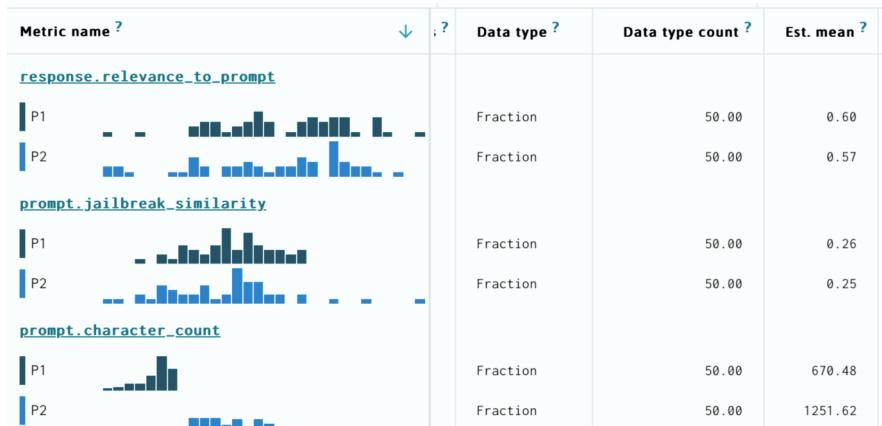


Fig. 8.8: A view of the WhyLabs monitoring dashboard, examining selected metrics to understand how they are impacted by simulated data drift on the prompts.

Having now applied both TruLens and LangKit to our TWEETSUMM models and data, a key observation is that there is in fact some overlap in their capabilities.

However, their implementations are quite different, and each offers certain advantages that the other does not. TruLens is more focused on evaluations, and LangKit is more oriented toward logging and monitoring. Depending on the application, it could make sense to use both, or it could make sense to choose one over the other. These are only two of the many LLM Ops solutions available; however, some research is often required to identify the most suitable approach.

8.9.4 Conclusion

Putting LLM applications into production is a significant undertaking beyond what we can hope to accomplish in a brief exercise such as this one. We were, however, able to demonstrate some of the tools that exist to make this process more manageable. There are a vast number of different considerations that factor into a model’s production readiness, but fortunately, the developers of tools such as TruLens and LangKit have designed repeatable solutions for many of them. By building workflows around these tools, an application can progress to a more mature state in less time.

References

- Megha Agarwal, Asfandyar Qureshi, Nikhil Sardana, Linden Li, Julian Quevedo, and Daya Khudia. Llm inference performance engineering: Best practices, 10 2023. URL <https://www.databricks.com/blog/llm-inference-performance-engineering-best-practices>.
- Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in llm inference with sarathi-serve, 2024.
- Norah Alzahrani et al. When benchmarks are targets: Revealing the sensitivity of large language model leaderboards, 2024.
- Emeka Boris Ama. Llm monitoring: The beginner’s guide, 11 2023. URL <https://www.lakera.ai/blog/llm-monitoring>.
- Stéphane Aroca-Ouellette, Cory Paik, Alessandro Roncone, and Katharina Kann. Prost: Physical reasoning of objects through space and time. *arXiv preprint arXiv:2106.03634*, 2021.
- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.
- Gad Benram. Top tools for prompt engineering?, 2023. URL <https://www.tensorops.ai/post/top-tools-for-prompt-engineering>.

- Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439, 2020.
- Su Lin Blodgett, Lisa Green, and Brendan O’Connor. Demographic dialectal variation in social media: A case study of african-american english. *arXiv preprint arXiv:1608.08868*, 2016.
- Ondrej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, et al. Findings of the 2016 conference on machine translation (wmt16). In *First conference on machine translation*, pages 131–198. Association for Computational Linguistics, 2016.
- Jordan Boyd-Graber, Brianna Satinoff, He He, and Hal Daumé III. Besting the quiz master: Crowdsourcing incremental classification games. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*, pages 1290–1301, 2012.
- Jordan Burgess. What is human-in-the-loop ai?, 11 2021. URL <https://humanloop.com/blog/human-in-the-loop-ai>.
- Yingshan Chang, Mridu Narang, Hisami Suzuki, Guihong Cao, Jianfeng Gao, and Yonatan Bisk. Webqa: Multihop and multimodal qa. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16495–16504, 2022.
- Mark Chen et al. Evaluating large language models trained on code, 2021.
- Eunsol Choi, He He, Mohit Iyyer, Mark Yatskar, Wen-tau Yih, Yejin Choi, Percy Liang, and Luke Zettlemoyer. Quac: Question answering in context. *arXiv preprint arXiv:1808.07036*, 2018.
- Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of NAACL-HLT 2019*, 2019.
- Jonathan H Clark, Eunsol Choi, Michael Collins, Dan Garrette, Tom Kwiatkowski, Vitaly Nikolaev, and Jennimaria Palomaki. Tydi qa: A benchmark for information-seeking question answering in typologically diverse languages. *Transactions of the Association for Computational Linguistics*, 8:454–470, 2020.
- Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.
- Alexis Conneau, Guillaume Lample, Ruty Rinott, Adina Williams, Samuel R Bowman, Holger Schwenk, and Veselin Stoyanov. Xnli: Evaluating cross-lingual sentence representations. *arXiv preprint arXiv:1809.05053*, 2018.
- Ido Dagan, Oren Glickman, and Bernardo Magnini. The pascal recognising textual entailment challenge. In *Machine learning challenges workshop*, pages 177–190. Springer, 2005.

- Marie-Catherine De Marneffe, Mandy Simons, and Judith Tonhauser. The commitmentbank: Investigating projection in naturally occurring discourse. In *proceedings of Sinn und Bedeutung*, volume 23, pages 107–124, 2019.
- Emily Dinan, Stephen Roller, Kurt Shuster, Angela Fan, Michael Auli, and Jason Weston. Wizard of wikipedia: Knowledge-powered conversational agents. *arXiv preprint arXiv:1811.01241*, 2018.
- Emily Dinan, Varvara Logacheva, Valentin Malykh, Alexander Miller, Kurt Shuster, Jack Urbanek, Douwe Kiela, Arthur Szlam, Iulian Serban, Ryan Lowe, et al. The second conversational intelligence challenge (convai2). In *The NeurIPS'18 Competition: From Machine Learning to Intelligent Conversations*, pages 187–208. Springer, 2020.
- Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. *arXiv preprint arXiv:1903.00161*, 2019.
- Thiago Castro Ferreira, Claire Gardent, Nikolai Ilinykh, Chris Van Der Lee, Simon Mille, Diego Moussallem, and Anastasia Shimorina. The 2020 bilingual, bi-directional webnlg+ shared task overview and evaluation results (webnlg+ 2020). In *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*, 2020.
- Sarah Gao and Andrew Kean Gao. On the origin of llms: An evolutionary tree and graph for 15,821 large language models, 2023.
- Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361, 2021.
- Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.
- Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*, 2021.
- Jordan Hoffmann et al. Training compute-optimal large language models, 2022.
- Kilian Kluge Jakub Czakon. Ml experiment tracking: What it is, why it matters, and how to implement it, 4 2024. URL <https://neptune.ai/blog/ml-experiment-tracking>.
- Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus, 2017.
- Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension, 2017.
- Daniel Khashabi, Snigdha Chaturvedi, Michael Roth, Shyam Upadhyay, and Dan Roth. Looking beyond the surface: A challenge set for reading comprehension over multiple sentences. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 252–262, 2018.

- Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. Mawps: A math word problem repository. In *Proceedings of the 2016 conference of the north american chapter of the association for computational linguistics: human language technologies*, pages 1152–1157, 2016.
- Sumith Kulal, Panupong Pasupat, Kartik Chandra, Mina Lee, Oded Padon, Alex Aiken, and Percy S Liang. Spoc: Search-based pseudocode to code. *Advances in Neural Information Processing Systems*, 32, 2019.
- Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From word embeddings to document distances. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 957–966, Lille, France, 07–09 Jul 2015. PMLR. URL <https://proceedings.mlr.press/v37/kusnerb15.html>.
- Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.
- Yanis Labrak, Adrien Bazoge, Emmanuel Morin, Pierre-Antoine Gourraud, Mickael Rouvier, and Richard Dufour. Biomstral: A collection of open-source pretrained large language models for medical domains, 2024.
- Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations, 2017.
- Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. Ds-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pages 18319–18345. PMLR, 2023.
- Huong T Le, Dung T Cao, Trung H Bui, Long T Luong, and Huy Q Nguyen. Improve quora question pair dataset for question similarity task. In *2021 RIVF International Conference on Computing and Communication Technologies (RIVF)*, pages 1–5. IEEE, 2021.
- Hector Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In *Thirteenth international conference on the principles of knowledge representation and reasoning*, 2012.
- Patrick Lewis, Barlas Oğuz, Ruty Rinott, Sebastian Riedel, and Holger Schwenk. Mlqa: Evaluating cross-lingual extractive question answering. *arXiv preprint arXiv:1910.07475*, 2019.
- Peng Li, Wei Li, Zhengyan He, Xuguang Wang, Ying Cao, Jie Zhou, and Wei Xu. Dataset and neural recurrent sequence labeling model for open-domain factoid question answering. *arXiv preprint arXiv:1607.06275*, 2016.
- Yinheng Li, Shaofei Wang, Han Ding, and Hang Chen. Large language models in finance: A survey, 2023.
- Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittweis, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

- Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*, 2022.
- Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *arXiv preprint arXiv:1705.04146*, 2017.
- Bang Liu, Di Niu, Haojie Wei, Jinghong Lin, Yancheng He, Kunfeng Lai, and Yu Xu. Matching article pairs with graphical decomposition and convolutions. *arXiv preprint arXiv:1802.07459*, 2018.
- Jian Liu, Leyang Cui, Hanmeng Liu, Dandan Huang, Yile Wang, and Yue Zhang. Logiqa: A challenge dataset for machine reading comprehension with logical reasoning. *arXiv preprint arXiv:2007.08124*, 2020.
- Chung Kwan Lo. What is the impact of chatgpt on education? a rapid review of the literature. *Education Sciences*, 13(4):410, 2023.
- Barrault Loïc, Biesialska Magdalena, Bojar Ondřej, Federmann Christian, Graham Yvette, Grundkiewicz Roman, Haddow Barry, Huck Matthias, Joanis Eric, Koci Tom, et al. Findings of the 2020 conference on machine translation (wmt20). In *Proceedings of the Fifth Conference on Machine Translation*, pages 1–55. Association for Computational Linguistics,, 2020.
- R Thomas McCoy, Ellie Pavlick, and Tal Linzen. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. *arXiv preprint arXiv:1902.01007*, 2019.
- Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.
- Shen-Yun Miao, Chao-Chun Liang, and Keh-Yih Su. A diverse corpus for evaluating and developing english math word problem solvers. *arXiv preprint arXiv:2106.15772*, 2021.
- Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.
- Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. Model cards for model reporting. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, FAT* ’19. ACM, January 2019. doi: 10.1145/3287560.3287596. URL <http://dx.doi.org/10.1145/3287560.3287596>.
- Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. A corpus and evaluation framework for deeper understanding of commonsense stories. *arXiv preprint arXiv:1604.01696*, 2016.
- Shashi Narayan, Shay B Cohen, and Mirella Lapata. Don’t give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *arXiv preprint arXiv:1808.08745*, 2018.
- Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. Adversarial nli: A new benchmark for natural language understanding. *arXiv preprint arXiv:1910.14599*, 2019.

- Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. The e2e dataset: New challenges for end-to-end generation. *arXiv preprint arXiv:1706.09254*, 2017.
- Stephen Oladele. Llmops: What it is, why it matters, and how to implement it, 3 2024. URL <https://neptune.ai/blog/llmops>.
- Darren Orf. Microsoft has lobotomized the ai that went rogue, 2 2023. URL <https://www.popularmechanics.com/technology/robots/a43017405/microsoft-bing-ai-chatbot-problems/>.
- Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raefaela Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.
- Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL <https://doi.org/10.3115/1073083.1073135>.
- Arkil Patel, Satwik Bhattacharya, and Navin Goyal. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*, 2021.
- Saurav Pawar, S. M Towhidul Islam Tonmoy, S M Mehedi Zaman, Vinija Jain, Aman Chadha, and Amitava Das. The what, why, and how of context length extension techniques in large language models – a detailed survey, 2024.
- Anselmo Peñas, Eduard Hovy, Pamela Forner, Álvaro Rodrigo, Richard Sutcliffe, and Roser Morante. Qa4mre 2011-2013: Overview of question answering for machine reading evaluation. In *Information Access Evaluation. Multilinguality, Multimodality, and Visualization: 4th International Conference of the CLEF Initiative, CLEF 2013, Valencia, Spain, September 23-26, 2013. Proceedings 4*, pages 303–320. Springer, 2013.
- Mohammad Taher Pilehvar and José Camacho-Collados. Wic: 10,000 example pairs for evaluating context-sensitive representations. *arXiv preprint arXiv:1808.09121*, 6:17, 2018.
- Edoardo Maria Ponti, Goran Glavaš, Olga Majewska, Qianchu Liu, Ivan Vulić, and Anna Korhonen. Xcopa: A multilingual dataset for causal commonsense reasoning. *arXiv preprint arXiv:2005.00333*, 2020.
- Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2019.
- Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016.
- Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In Iryna Gurevych and Yusuke Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2124. URL <https://aclanthology.org/P18-2124>.

- Leonardo Ranaldi, Giulia Pucci, Federico Ranaldi, Elena Sofia Ruzzetti, and Fabio Massimo Zanzotto. Empowering multi-step reasoning across languages via tree-of-thoughts, 2024.
- Hannah Rashkin, Eric Michael Smith, Margaret Li, and Y-Lan Boureau. Towards empathetic open-domain conversation models: A new benchmark and dataset. *arXiv preprint arXiv:1811.00207*, 2018.
- Siva Reddy, Danqi Chen, and Christopher D Manning. Coqa: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266, 2019.
- Josh Reini et al. truera/trulens: Trulens eval v0.25.1, 2024. URL <https://zenodo.org/doi/10.5281/zenodo.4495856>.
- Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S Gordon. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *2011 AAAI Spring Symposium Series*, 2011.
- Subhro Roy and Dan Roth. Solving general arithmetic word problems. *arXiv preprint arXiv:1608.01413*, 2016.
- Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Wino-grande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.
- Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. Socialiq: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*, 2019.
- Elvis Saravia. Prompt Engineering Guide. <https://github.com/dair-ai/Prompt-Engineering-Guide>, 12 2022.
- Roie Schwaber-Cohen. What is a vector database how does it work? use cases + examples, 5 2023. URL <https://www.pinecone.io/learn/vector-database/>.
- Thomas Scialom, Paul-Alexis Dray, Sylvain Lamprier, Benjamin Piwowarski, and Jacopo Staiano. Mlsum: The multilingual summarization corpus. *arXiv preprint arXiv:2004.14900*, 2020.
- S.D.N.Y. The new york times company v microsoft corporation, openai, inc., openai lp, openai gp, llc, openai llc, openai opco llc, openai global llc, oai corporation, llc and openai holdings llc., 2023. URL https://nytco-assets.nytimes.com/2023/12/NYT_Complaint_Dec2023.pdf.
- Zhihong Shao, Minlie Huang, Jiangtao Wen, Wenfei Xu, and Xiaoyan Zhu. Long and diverse text generation with planning-based hierarchical variational model. *arXiv preprint arXiv:1908.06605*, 2019.
- Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, et al. Language models are multilingual chain-of-thought reasoners. *arXiv preprint arXiv:2210.03057*, 2022.
- Eric Michael Smith, Mary Williamson, Kurt Shuster, Jason Weston, and Y-Lan Boureau. Can you put it all together: Evaluating conversational agents’ ability to blend skills. *arXiv preprint arXiv:2004.08449*, 2020.

- Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*, 2022.
- Alon Talmor, Jonathan Herzog, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *arXiv preprint arXiv:1811.00937*, 2018.
- Alexey Tikhonov and Max Ryabinin. It's all in the heads: Using attention heads as a baseline for cross-lingual transfer in commonsense reasoning. *arXiv preprint arXiv:2106.12066*, 2021.
- David Vilares and Carlos Gómez-Rodríguez. Head-qa: A healthcare dataset for complex reasoning. *arXiv preprint arXiv:1906.04701*, 2019.
- Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.
- Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32, 2019.
- Yan Wang, Xiaojiang Liu, and Shuming Shi. Deep neural solver for math word problems. In *Proceedings of the 2017 conference on empirical methods in natural language processing*, pages 845–854, 2017.
- Jason Wei et al. Emergent abilities of large language models, 2022.
- Ralph Weischedel, Sameer Pradhan, Lance Ramshaw, Martha Palmer, Nianwen Xue, Mitchell Marcus, Ann Taylor, Craig Greenberg, Eduard Hovy, Robert Belvin, et al. Ontonotes release 4.0. *LDC2011T03, Philadelphia, Penn.: Linguistic Data Consortium*, 17, 2011.
- Johannes Welbl, Nelson F Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. *arXiv preprint arXiv:1707.06209*, 2017.
- WhyLabs. URL <https://github.com/whylabs/langkit>.
- Adina Williams, Nikita Nangia, and Samuel R Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017.
- BigScience Workshop et al. Bloom: A 176b-parameter open-access multilingual language model, 2023.
- Yinfei Yang, Yuan Zhang, Chris Tar, and Jason Baldridge. Paws-x: A cross-lingual adversarial dataset for paraphrase identification. *arXiv preprint arXiv:1908.11828*, 2019.
- Shukang Yin, Chaoyou Fu, Sirui Zhao, Ke Li, Xing Sun, Tong Xu, and Enhong Chen. A survey on multimodal large language models, 2024.
- Zheng-Xin Yong et al. Prompting multilingual large language models to generate code-mixed texts: The case of south east asian languages, 2023.
- Zhiyuan Yu, Xiaogeng Liu, Shunning Liang, Zach Cameron, Chaowei Xiao, and Ning Zhang. Don't listen to me: Understanding and exploring jailbreak prompts of large language models, 2024.

- Lifan Yuan, Yangyi Chen, Ganqu Cui, Hongcheng Gao, Fangyuan Zou, Xingyi Cheng, Heng Ji, Zhiyuan Liu, and Maosong Sun. Revisiting out-of-distribution robustness in nlp: Benchmark, analysis, and llms evaluations, 2023.
- Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.
- Sheng Zhang, Xiaodong Liu, Jingjing Liu, Jianfeng Gao, Kevin Duh, and Benjamin Van Durme. Record: Bridging the gap between human and machine commonsense reading comprehension. *arXiv preprint arXiv:1810.12885*, 2018.
- Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert, 2020.



Chapter 9

Multimodal LLMs

Abstract Multimodal Large Language Models emulate human perception by integrating multiple data types such as text, images, and audio, significantly enhancing AI's understanding and interaction capabilities. The MMLM framework, presented with various components, is discussed both theoretically and practically by mapping each component to state-of-the-art variations. This chapter also presents how various techniques, such as instruction tuning, in-context learning, chain-of-thoughts, and alignment tuning, are adapted from traditional LLMs to multimodal contexts to improve adaptability and reasoning across modalities. Three state-of-the-art MMLLMs—Flamingo, Video-LLaMA, and NExT-GPT—are presented to provide comprehensive coverage and mapping to the generic framework. Having discussed the theoretical underpinnings of MMLLMs in detail, the chapter concludes with a tutorial demonstrating the behavior of a “Text+X-to-Text” model, using images as the modality “X”. This tutorial includes experiments on image labeling and captioning, comparing zero-shot, few-shot, and fine-tuned frameworks to test and improve model performance.

9.1 Introduction

In the real world, humans rarely rely on a single mode of communication. We perceive our environment through various inputs such as sights, sounds, and other sensory inputs, synthesizing this information to understand and react to our surroundings. *Multimodal large language models* (MMLLMs) aim to emulate this multi-faceted approach, enhancing their understanding and response accuracy in real-world applications. Multimodal LLMs represent a significant leap in AI technology, integrating diverse data types (or modalities) such as text, images, audio, and sensory inputs. Unlike traditional models that handle a single data type, multimodal models process and interpret complex, layered data from inputs and outputs that can map to

different modal outputs. This capability mimics human cognitive abilities to understand and interact with the world through multiple senses.

Cross-modal learning encompasses a range of tasks where inputs and outputs span different sensory modalities, such as visual and textual data. Some key examples of these tasks are as follows:

1. **Image-Text Retrieval:** This task involves either using text to retrieve relevant images or using images to retrieve relevant textual descriptions.
2. **Video-Text Retrieval:** This task focuses on either using text to find relevant videos or using videos to generate textual descriptions.
3. **Image and Video Captioning:** The goal is to generate descriptive text for given images or videos. The inputs are visual content (images or videos), and the output is a corresponding textual description.
4. **Visual Question Answering (VQA):** VQA involves providing a system with an image or video (visual input) along with a related question in text form. The task is to output an answer to the question based on the visual content, thus requiring the integration of visual and textual inputs.
5. **Gesture-Based Control with Audio Feedback:** This involves interpreting visual inputs (gestures) and providing corresponding audio feedback. The input is a visual gesture, and the output is an audio response or action the system takes, integrating visual and auditory modalities.

9.2 Brief History

As outlined in [Wu et al. \(2023c\)](#), the multimodal automation field has undergone four distinct evolutionary phases throughout the progression of multimodal research.

The first phase, from 1980 to 2000, focused on single modalities and the use of statistical techniques. During the 1980s, statistical algorithms and image-processing methods were prominently employed in developing facial recognition systems. IBM's research team significantly advanced speech recognition by applying hidden Markov models, enhancing the technology's accuracy and dependability ([Bahl et al., 1986](#)). In the 1990's, Kanade's team pioneered the Eigenfaces approach, employing principal component analysis to identify individuals effectively through statistical analysis of facial imagery ([Satoh and Kanade, 1997](#)). Companies, including Dragon Systems, advanced speech recognition technology and achieved great success in converting spoken words into written text with greater accuracy ([LaRocca et al., 1999](#)).

From 2000 to 2010, the second phase was characterized by the conversion of modalities, strongly emphasizing human-computer interaction. In 2001, the AMI project explored the use of computers for recording and processing meeting data, aiming to enhance information retrieval and collaboration ([Carletta et al., 2005](#)). In 2003, the "Cognitive Assistant that Learns and Organizes" (CALO) project introduced early chatbot technologies, a precursor to systems such as Siri, intending to create a virtual assistant to comprehend and respond to human language ([Tur et al.,](#)

2010). The Social Signal Processing (SSP) project delved into analyzing nonverbal cues, such as facial expressions and voice tones to facilitate more natural human-computer interactions (Vinciarelli et al., 2008).

During the third phase, spanning from 2010 to 2020, the field witnessed the fusion of modalities. This era was marked by the integration of deep learning and neural networks, leading to significant breakthroughs. In 2011, Ngiam et al. (2011) introduced a groundbreaking multimodal deep learning algorithm that facilitated the joint analysis of different modalities, such as images and text, enhancing tasks such as image classification, video analysis, and speech recognition. In 2012, deep Boltzmann machines were utilized to capture relationships between various modalities and for generative power (Hinton and Salakhutdinov, 2012). Furthermore, in 2016, a neural image captioning algorithm with semantic attention emerged, enabling the generation of descriptive captions for images, thereby improving accessibility and supporting applications like automated image tagging (You et al., 2016).

The development of large-scale multimodal models defined the final phase, beginning in 2020 and extending into the future. In 2021, the Contrastive Language-Image Pretraining (CLIP) model disrupted traditional approaches by focusing on the unsupervised processing of image-text pairs rather than relying on fixed category labels (Radford et al., 2021). The following year, DALL-E 2, a model from OpenAI, leveraged a diffusion model based on CLIP image embeddings to generate high-quality images from text prompts. In 2023, Microsoft released KOSMOS-1, a multimodal LLM capable of processing information from various modalities and adapting it through in-context learning (Huang et al., 2024). Additionally, PaLM-E emerged as a benchmark in visual-language performance, combining language and vision models without the need for task-specific fine-tuning and excelling in visual and language tasks, ranging from object detection to code generation (Driess et al., 2023). ImageBind introduced a method to learn a unified embedding for six modalities—images, text, audio, depth, thermal, and IMU data—demonstrating that pairing with images alone suffices for binding these modalities, enabling innovative applications in cross-modal retrieval and generation (Girdhar et al., 2023). NExT-GPT has emerged as a versatile end-to-end multimodal LLM capable of handling any combination of image, video, audio, and text inputs and outputs (Wu et al., 2023c).

9.3 Multimodal LLM Framework

Multimodal LLMs exhibit diverse architectures, depending on various components and choices tailored to specific functionalities and modalities. This section offers an in-depth exploration of the various elements constituting the architecture of MM-LLMs, detailing the specific implementation strategies selected for each component, as depicted in Fig. 9.1. This framework synthesizes insights from diverse research, including the works of Chip (2023); Wang et al. (2023); Wu et al. (2023b); Xu et al. (2023); Yin et al. (2023); Zhang et al. (2024a) on multimodal LLMs.

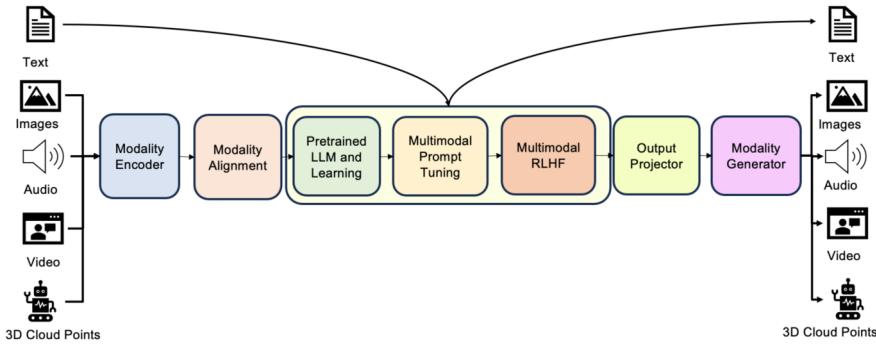


Fig. 9.1: The general framework of MMLLMs with different components providing implementation choices.

9.3.1 Modality Encoder

The *modality encoder* (ME) is typically the initial processing unit for mapping various data modalities. Generally, each data type – images, video, or audio – is processed through a modality-specific encoder. These encoders are designed to convert the unique characteristics of each data type into embeddings, which are vector representations that can be uniformly understood and manipulated by the subsequent layers of the model.

The following formulation captures the operational essence of the ME:

$$\mathbf{F}_X = \text{ME}_X(\mathbf{l}_X), \quad (9.1)$$

where \mathbf{l}_X symbolizes the input data from modalities such as images, videos, audio, or 3D objects, and \mathbf{F}_X represents the extracted features. Next, we will discuss some of the well-known encoders for various streams used in the research.

The Vision Transformer (ViT), proposed by [Dosovitskiy et al. \(2020\)](#), uses the Transformer architecture, traditionally used for natural language processing, for image analysis. By partitioning an image into patches and subjecting them to a linear projection, ViT leverages the power of Transformer blocks to encode visual information. Building on the foundation laid by ViT, CLIP ViT by [Radford et al. \(2021\)](#) introduces a method for learning visual concepts from natural language supervision. By training on a large corpus of text-image pairs, CLIP ViT employs contrastive learning to enhance the alignment between images and their corresponding textual descriptions, significantly improving the model's ability to understand and categorize visual content. Many MMLLMs use the CLIP encoder to encode image data. Eva-CLIP ViT, a further refinement by [Fang et al. \(2023\)](#), addresses some of the challenges associated with training large-scale models like CLIP.

! Practical Tips

By stabilizing the training process, Eva-CLIP ViT provides efficient scaling and enhances the training of multimodal base models in visual recognition tasks, thus providing a good choice for image encoders.

For video content, a common approach involves sampling a fixed number of frames (typically five) and subjecting these frames to the same pre-processing steps as images. This uniform treatment ensures consistency in feature extraction across different visual modalities.

Several encoders, such as C-Former, HuBERT, BEATs, and Whisper, have emerged to transform sound data in the audio domain.

C-Former, by [Chen et al. \(2023\)](#), leverages the continuous integrate-and-fire (CIF) alignment mechanism alongside a Transformer architecture to perform sequence transduction, effectively extracting nuanced audio features from raw sound data. HuBERT, introduced by [Shi et al. \(2022\)](#), adopts a self-supervised learning strategy rooted in BERT's framework. It focuses on predicting masked audio segments, thereby learning robust speech representations that can serve various downstream tasks. BEATs, another contribution of Chen et al., presents an iterative framework for audio pre-training ([Chen et al., 2022](#)).

There has been significant interest in 3D visual understanding, spurred by its expanding utility across several cutting-edge domains, such as augmented and virtual reality (AR and VR), autonomous vehicle navigation, the Metaverse, and various robotics applications. Building upon the foundational achievements of the ULIP framework, ULIP-2, by autogenerating descriptive language for 3D objects, extends the ULIP paradigm, enabling the creation of large-scale tri-modal datasets without the traditional reliance on manual annotations, and is a benchmark in the realm of 3D visual comprehension. 3D point cloud modality encoding is performed using ULIP-2 model encoders ([Xue et al., 2023](#)).

9.3.2 Input Projector

The core of modality alignment, $\text{IN_ALIGN}_{X \rightarrow T}$, involves the process of aligning encoded features from various modalities \mathbf{F}_T with the textual feature space T using the input projector component. This alignment facilitates the generation of prompts \mathbf{P}_X , which, along with textual features \mathbf{F}_T , are input into the LLMs.

In the context of a multimodal-text dataset $\{(\mathbf{l}_X, t)\}$, the primary objective is to minimize the loss associated with text generation conditioned on modality X , expressed as $\mathcal{L}_{\text{txt-gen}}$:

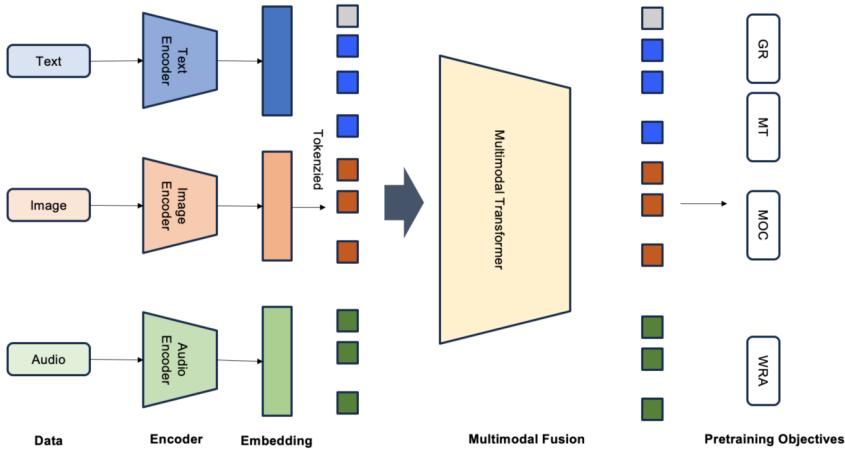


Fig. 9.2: The single-stream architecture

$$\arg \min_{\text{IN_ALIGN}_{X \rightarrow T}} \mathcal{L}_{\text{txt-gen}} (\text{LLM}(\mathbf{P}_X, \mathbf{F}_T), t) \quad (9.2)$$

where the aligned features as prompts \mathbf{P}_X are obtained by applying the Input Projector to the nontextual features:

$$\mathbf{P}_X = \text{IN_ALIGN}_{X \rightarrow T}(\mathbf{F}_X) \quad (9.3)$$

Multimodal pre-trained models use a multilayer Transformer architecture to extract and interact features from various modalities. One way to categorize these architectures is by their approach to multimodal information integration, distinguishing them into single-stream and cross-stream types.

- **Single-Stream Architecture:** Multimodal inputs such as images and text are treated equally and fused in a unified model. This process involves extracting unimodal features from each modality, which are then tokenized and concatenated using separators, as shown in Fig. 9.2. These concatenated features serve as inputs to a multimodal Transformer, which is instrumental in the fusion process. The multi-head self-attention mechanism facilitates the interactive fusion of unimodal features, leading to the generation of multimodal fusion features (Li et al., 2020c). These features are typically derived from the class token of the Transformer, which encapsulates information from various modalities and enhances the model's characterization capabilities.
- **Cross-Stream Architecture:** In this approach, features of different modalities are extracted in parallel by independent models and then aligned using self-supervised contrastive learning (discussed later) as shown in Fig. 9.3. This approach is distinct from single-stream architectures, which focus on aligning uni-

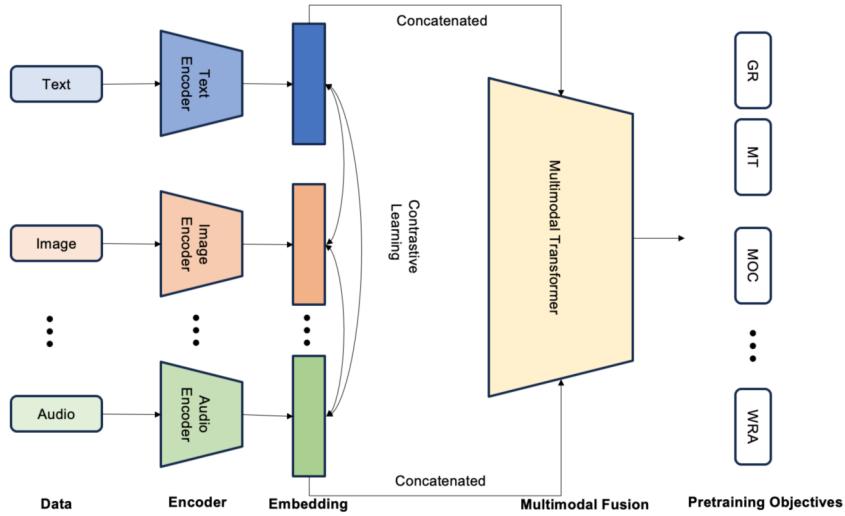


Fig. 9.3: The cross-stream architecture

modal features instead of creating fused multimodal features. Notable examples of large-scale MMLMs employing cross-stream structures include BriVL and CLIP (Huo et al., 2021; Radford et al., 2021). A vital characteristic of these models is their ability to align features from different modalities into a cohesive, high-dimensional feature space. Cross-stream models are recognized for their flexibility; alterations made in one modality’s structure do not impact the others, thus facilitating more accessible applications in real-world settings. These models are primarily designed for embedding-level matching, often leveraging what is termed “weak semantic correlation”. One notable aspect of cross-stream models is their approach to handling the differences and complementarities between multimodal data. Additionally, the structural design of these pre-training models varies based on the specific pre-training objectives. Notably, when generative tasks such as masked image reconstruction or generating images based on text descriptions are involved, these models incorporate a decoder following the encoder. This decoder transforms the multimodal fusion features into the appropriate outputs, completing the pre-training process.

Multimodal Transformers facilitate cross-modal interactions, such as fusion and alignment, through self-attention mechanisms and their variants. The self-attention approaches are modality-agnostic, tokenization-agnostic, and embedding-agnostic, showcasing the versatility of treating any token’s embeddings from any modality. Given inputs X_A and X_B from two distinct modalities, $Z(A)$ and $Z(B)$ denote their respective token embeddings. The following outlines these practices and their mathematical formulations in a two-modality context, although they are adaptable to multiple modalities:

1. **Early Summation:** Token embeddings from multiple modalities are weighted and summed at each token position before processing by Transformer layers:

$$Z \leftarrow Tf(\alpha Z(A) \oplus \beta Z(B)) = MHSA(Q(AB), K(AB), V(AB)),$$

where \oplus indicates element-wise summation. This method offers simplicity and effectiveness without increasing computational complexity ([Gavrilyuk et al., 2020](#)).

2. **Early Concatenation (Co-Transformer):** Token embedding sequences from different modalities are concatenated:

$$Z \leftarrow Tf(C(Z(A), Z(B))).$$

This all-attention or *Co-Transformer* approach allows a unified sequence treatment, enhancing each modality's encoding by contextualizing with other modalities ([Sun et al., 2019](#)).

3. **Hierarchical Attention (Multi-stream to One-stream):** Independent Transformer streams first encode multimodal inputs; their outputs are then concatenated and fused:

$$Z \leftarrow Tf_3(C(Tf_1(Z(A)), Tf_2(Z(B)))).$$

This method represents a form of late interaction or fusion, acting as a particular case of early concatenation ([Li et al., 2021](#)).

4. **Hierarchical Attention (One-stream to Multi-stream):** Concatenated multimodal inputs are encoded by a shared single-stream Transformer, followed by separate streams for each modality:

$$\begin{aligned} C(Z(A), Z(B)) &\leftarrow Tf_1(C(Z(A), Z(B))), \\ Z(A) &\leftarrow Tf_2(Z(A)), \\ Z(B) &\leftarrow Tf_3(Z(B)). \end{aligned}$$

This structure, utilized in InterBERT, captures cross-modal interactions while preserving unimodal representation independence ([Lin et al., 2020](#)).

5. **Cross-Attention:** In two-stream Transformers, exchanging query embeddings across streams enables enhanced cross-modal interactions:

$$\begin{aligned} Z(A) &\leftarrow MHSA(Q_B, K_A, V_A), \\ Z(B) &\leftarrow MHSA(Q_A, K_B, V_B). \end{aligned}$$

First proposed in VilBERT, this method maintains computational efficiency and fosters cross-modal perception ([Lu et al., 2019](#)).

6. **Cross-Attention to Concatenation:** Cross-attention streams are concatenated and further processed to model the global context:

$$\begin{aligned} Z(A) &\leftarrow \text{MHSA}(Q_B, K_A, V_A), \\ Z(B) &\leftarrow \text{MHSA}(Q_A, K_B, V_B), \\ Z &\leftarrow Tf(C(Z(A), Z(B))). \end{aligned}$$

This hierarchical cross-modal interaction approach mitigates the drawbacks of standalone cross-attention (Zhan et al., 2021).

9.3.3 Pre-training: Core LLMs, Datasets and Task-Specific Objectives

At the heart of MMLMs lies the LLM, which generates responses. Given that inputs can include both textual and nontextual data, new techniques are needed for the language model to condition its responses on a range of modalities. The LLM processes representations from various modalities for semantic understanding, reasoning, and decision-making regarding the inputs. It produces two main outputs:

1. Direct textual outputs, denoted as t ,
2. Signal tokens, denoted as \mathbf{S}_X , from other modalities.

These signal tokens act as instructions to guide the generator on whether to produce multimodal content. This can be mathematically represented as:

$$(t, \mathbf{S}_X) = \text{LLM}(\mathbf{P}_X, \mathbf{F}_T), \quad (9.4)$$

where \mathbf{P}_X can be considered as soft-prompt tuning for the LLM.

MMLMs are categorized into encoder-only, decoder-only, and encoder-decoder models. Some common LLMs used for multimodal training are listed below in Table 9.1 with necessary details.

Table 9.1: Base LLMs in Multimodal

LLM Model	Architecture	Notes
Flan-T5 (Chung et al., 2022)	Encoder-Decoder	Explores Instruction Tuning for T5, demonstrating strong zero-shot and Chain-of-Thought (CoT) capabilities.
ChatGLM2 (Zeng et al., 2022)	Autoregressive	A bilingual model for Chinese-English dialog, optimized for question-answering and dialog in Chinese.
UL2 (Tay et al., 2022)	Encoder-Decoder	Trained with denoising objectives, surpassing T5 benchmarks.
Qwen (Bai et al., 2023)	Decoder-Only	Focuses on bilingual capabilities for Chinese and English, using alignment techniques for enhanced dialog model performance.

Continued on next page

Table 9.1 – *Continued from previous page*

LLM Model	Architecture	Notes
Chinchilla (Hoffmann et al., 2022)	Causal Decoder	Advocates for model size scaling with the dataset size, trained on a large corpus of text data.
OPT (Zhang et al., 2022)	Causal-Decoder	An open-source effort to replicate GPT-3's performance.
PaLM (Chowdhery et al., 2022)	Causal Decoder	Features parallel attention and feed-forward layers, improving training speeds with innovations like RoPE embeddings and SwiGLU activation.
Llama (Touvron et al., 2023)	Decoder-Only	Utilizes efficient causal attention for decoder-only architectures.
Llama-2 (Touvron et al., 2023)	Decoder-Only	Enhances Llama with 40% more training data and innovations for conversation generation.
Vicuna (Chiang et al., 2023)	Decoder-Only	Built on Llama. Leverages user dialog data for training, aiming to enhance conversational abilities.

During the pre-training phase, models typically utilize datasets that include a range of modalities, such as image-text, video-text, and audio-text. This phase's primary focus is training two key components: input projectors and output projectors. The objective is to achieve feature alignment across these various modalities. While optimization is generally concentrated on these components, parameter-efficient fine-tuning is occasionally employed within the LLM to further refine the model's capabilities in processing multimodal information further.

Table 9.2 lists datasets commonly utilized in the pre-training process ([Wang et al., 2023](#); [Yin et al., 2023](#)).

Table 9.2: List of Datasets Commonly Used in Pre-training Process

Dataset Name	Modality	Size
ALIGN	Image-Text	1.8B
LTIP	Image-Text	312M
MS-COCO	Image-Text	620K
VisualGenome	Image-Text	4.5M
CC3M	Image-Text	3.3M
CC12M	Image-Text	12.4M
SBU	Image-Text	1M
LAION-400M	Image-Text	400M

Continued on next page

Table 9.2 – *Continued from previous page*

Dataset Name	Modality	Size
Flickr30k	Image-Text	158K
AIChallengerCaptions	Image-Text	1.5M
COYO	Image-Text	747M
Wukong	Image-Text	101M
COCOCaption	Image-Text	1M
WebLI	Image-Text	12B
EpisodicWebLI	Image-Text	400M
CC595k	Image-Text	595K
RefCOCO+	Image-Text	142K
Visual-7W	Image-Text	328K
OCR-VQA	Image-Text	1M
ST-VQA	Image-Text	32K
DocVQA	Image-Text	50K
TextVQA	Image-Text	45.3K
DataComp	Image-Text	1.4B
GQA	Image-Text	22M
VQAv2	Image-Text	1.4M
DVQA	Image-Text	3.5M
OK-VQA	Image-Text	14K
A-OKVQA	Image-Text	24.9K
TextCaptions	Image-Text	145K
M3W (Interleaved)	Image-Text	43.3M Instances
MMC4 (Interleaved)	Image-Text	101.2M Instances
MSRVT	Video-Text	200K
WebVid	Video-Text	10M
VTP	Video-Text	27M
AISHELL-2	Audio	128K
WaveCaps	Audio	403K
VSDial-CN	Image-Audio-Text	1.2M

Designing learning objectives based on tasks and modalities is vital for multimodal pre-training. The following sections outline common learning objectives used in pre-training.

9.3.3.1 Contrastive Learning

Before CLIP, vision-language models mainly used classifier or language model objectives. The classifier approach was limited to predefined classes, restricting the model's response diversity and adaptability to different tasks. The language model objective, while more flexible, faced training challenges due to its focus on generating specific texts for each image.

Contrastive learning, as implemented in CLIP, aims to overcome the limitations of previous models by shifting the focus from predicting the exact text for each image to determining whether a given text is more aptly associated with a specific image than others (Radford et al., 2021). In practice, for a batch of N image-text pairs, CLIP generates N text embeddings and N image embeddings. Let V_1, V_2, \dots, V_N represent the embeddings for the N images, and L_1, L_2, \dots, L_N represent the embeddings for the N texts. CLIP computes the cosine similarity scores for all N^2 possible pairings of V_i, L_j . The training objective is to maximize the similarity scores for the N correct pairings while minimizing the scores for the $N^2 - N$ incorrect pairings.

$$\mathcal{L}_{i2t} = -\frac{1}{N} \sum_i \log \frac{\exp(V_i^T L_i / \sigma)}{\sum_j \exp(V_i^T L_j / \sigma)}, \quad (9.5)$$

$$\mathcal{L}_{t2i} = -\frac{1}{N} \sum_i \log \frac{\exp(L_i^T V_i / \sigma)}{\sum_j \exp(L_i^T V_j / \sigma)}, \quad (9.6)$$

$$\mathcal{L}_{CL} = \mathcal{L}_{i2t} + \mathcal{L}_{t2i}. \quad (9.7)$$

Here, \mathcal{L}_{i2t} and \mathcal{L}_{t2i} are image-to-text and text-to-image classification loss functions, respectively. \mathcal{L}_{CL} is the total contrastive loss. V_i and L_i represent the normalized image and text embeddings, respectively. N is the batch size, and σ is the temperature parameter.

9.3.3.2 Modality Matching Loss

Modality matching loss (MML) plays a critical role in pre-training large multimodal models, mainly due to its ability to capture explicit or implicit alignment relationships between different modalities. This loss function is applied in models such as Unicoder-VL, which employs visual linguistic matching (VLM) for vision-language pre-training (Li et al., 2020a). The VLM approach involves extracting both positive and negative image-sentence pairs and training the model to discern whether these pairs are aligned. The objective is to predict the matching scores of given sample pairs:

$$\mathcal{L}_{MML} = - \sum_{(x,y) \in \text{Pos}} \log p(\text{aligned}|x, y) - \sum_{(x',y') \in \text{Neg}} \log p(\text{unaligned}|x', y') \quad (9.8)$$

Here, (x, y) represents the positive image-sentence pairs, and (x', y') denotes the negative pairs. The model predicts the probability $p(\text{aligned}|x, y)$ that a pair is aligned and $p(\text{unaligned}|x', y')$ that it is not.

InterBERT introduces this variation with image-text matching using hard negatives, termed ITM-hn (Lin et al., 2020). This approach selects negative samples

based on the highest TF-IDF similarities, differing from typical negative sampling strategies:

$$\mathcal{L}_{\text{ITM-hn}} = - \sum_{(x,y) \in \text{Pos}} \log p(\text{aligned}|x, y) - \sum_{(x',y') \in \text{Hard Neg}} \log p(\text{unaligned}|x', y') \quad (9.9)$$

Including hard negatives, identified by high TF-IDF similarity scores, makes learning more challenging and effective, as the model must discern between closely related but unaligned pairs.

9.3.3.3 Masked Language Modeling

Masked language modeling (MLM) is a prevalent objective in pre-training frameworks, where researchers typically mask and fill input words randomly using special tokens. This method leverages the context from surrounding words and associated image regions to predict the masked words. In SIMVLM, as developed by Wang et al. (2021), this approach is combined with prefix language modeling (PrefixLM). PrefixLM applies bidirectional attention to a prefix sequence and autoregressive factorization for the subsequent tokens. In this context, words are denoted as $w = \{x_1, \dots, x_K\}$ and image regions as $v = \{v_1, \dots, v_T\}$. For MLM, a certain percentage $p\%$ of input words, represented as x_m , are masked at randomly generated indices m . The objective is to predict these masked words using the unmasked words $x_{\neg m}$ and all image regions v , by minimizing the negative log-likelihood:

$$\mathcal{L}_{\text{MLM}}(\theta) = -\mathbb{E}_{(x, v)} \log P_\theta(x_m | x_{\neg m}, v), \quad (9.10)$$

where θ are the trainable parameters.

In addition to MLM, PrefixLM in SIMVLM is another strategy for pre-training vision-language representation. This technique focuses on predicting the continuation of a text sequence given a prefix, formalized as:

$$\mathcal{L}_{\text{PrefixLM}}(\theta) = -\mathbb{E}_{x \sim D} \log P_\theta(x_{\geq T_p} | x_{< T_p}), \quad (9.11)$$

where x is the text sequence, D represents the pre-training data, and T_p is the length of the prefix sequence of tokens.

9.3.3.4 Masked Object Classification

This technique involves selectively masking portions of visual images, typically by setting their values to zero and then utilizing the labels predicted by an object detector as ground truth for these masked regions.

The methodology behind MOC is somewhat analogous to the masked language modeling (MLM) approach in NLP. In MOC, specific image regions are masked by

altering their visual features with a certain probability $p\%$. The primary objective is to predict the object category for these masked image regions accurately, denoted as v_i^m . This process entails passing the encoder output of the masked image regions v_i^m through a fully connected (FC) layer, which computes the scores for T object classes (Li et al., 2020a). These scores are then transformed into a normalized distribution $g_\theta(v_i^m)$ via a softmax function. The MOC objective is formally expressed as:

$$\mathcal{L}_{\text{MOC}}(\theta) = -\mathbb{E}_{(w,v)} \left[\sum_{i=1}^M \text{CE}(c(v_m^i), g_\theta(v_m^i)) \right] \quad (9.12)$$

where $c(v_m^i)$ represents the ground-truth label for the masked image region, and CE denotes the cross-entropy loss function. Here, θ signifies the parameters of the model, and the expectation \mathbb{E} is over the distribution of words w and visual features v . The MOC objective, therefore, focuses on enhancing the model's ability to infer and classify objects in partially observed or occluded visual contexts, reinforcing its understanding of visual information.

9.3.3.5 Image-Text Matching (ITM)

The ITM process is integral in developing models that can understand and relate visual content to corresponding textual descriptions. A crucial aspect of ITM involves generating negative training data, typically associating negative sentences with each image and vice versa. The objective is to enhance the model's discriminative capability in distinguishing between correctly matched image-text and mismatched pairs.

In the context of ITM, each image-text pair (v, t) is associated with a ground truth label y , indicating whether the pair is correctly matched (positive) or not (negative). The optimization of ITM is conducted using a binary classification loss function, which assesses the model's ability to predict these alignments accurately. The loss function for ITM, denoted as $\mathcal{L}_{\text{ITM}}(\theta)$, is mathematically formulated as:

$$\mathcal{L}_{\text{ITM}}(\theta) = -\mathbb{E}_{(v,t)} [y \log s_\theta(v, t) + (1 - y) \log(1 - s_\theta(v, t))] \quad (9.13)$$

where $s_\theta(v, t)$ represents the image-text similarity score computed by the model with parameters θ . The expectation $\mathbb{E}_{(v,t)}$ is taken over the distribution of image-text pairs. This loss function effectively measures the model's proficiency in identifying correct and incorrect alignments, thus refining its understanding of the complex relationships between visual and textual modalities.

9.3.3.6 Image-Text Generation

Image-text Generation (ITG) is an essential component of vision-language-related pre-training tasks. It focuses on training a model to generate text based on a given

image, leveraging aligned image-text pairs. For instance, Xu et al. (2021) trained the E2E-VLP model using the ITG objective. The ITG objective is formulated as follows:

$$\mathcal{L}_{ITG} = - \sum_{(x,y) \in (\mathcal{X},\mathcal{Y})} \log \prod_{t=1}^n P(y_t | y_{<t}, x) \quad (9.14)$$

Here, \mathcal{X} represents the visual sequence with context, and \mathcal{Y} is the set of generated text. The variable n indicates the length of tokens in the text y . This objective aims to maximize the probability of correctly generating the sequence of text tokens y_t based on the preceding tokens $y_{<t}$ and the visual input x .

9.3.3.7 Video-Subtitle Matching (VSM)

Video-subtitle matching (VSM) in video-text pre-training, as exemplified in HERO, focuses on two key alignment targets: local and global alignment (Li et al., 2020b). Score functions quantify the alignment between video and subtitle content, with separate scores for local and global alignment. The loss functions, however, are designed to optimize the model by minimizing the difference between these alignment scores for correctly matched video-subtitle pairs (positive pairs) and maximizing it for incorrectly matched pairs (negative pairs).

In HERO’s VSM implementation, two alignment targets are considered: local and global.

Score Functions

- Local Alignment Score Function:

$$S_{\text{local}}(s_q, \mathbf{v}) = \mathbf{V}^{\text{temp}} \mathbf{q} \in \mathbb{R}^{N_v}$$

- Global Alignment Score Function:

$$S_{\text{global}}(s_q, \mathbf{v}) = \max \left(\frac{\mathbf{V}_{\text{temp}}}{\|\mathbf{V}_{\text{temp}}\|} \cdot \frac{\mathbf{q}}{\|\mathbf{q}\|} \right)$$

Loss Functions

- Hinge loss for positive and negative query-video pairs:

$$\mathcal{L}_h(S_{\text{pos}}, S_{\text{neg}}) = \max(0, \delta + S_{\text{pos}} - S_{\text{neg}})$$

- Local alignment loss:

$$\mathcal{L}_{\text{local}} = -\mathbb{E}_D [\log(\mathbf{p}_{\text{st}}[y_{\text{st}}] + \log(\mathbf{p}_{\text{ed}}[y_{\text{ed}}]))]$$

- Global alignment loss:

$$\mathcal{L}_{\text{global}} = -\mathbb{E}_D [(\mathcal{L}_h(S_{\text{global}}(s_q, \mathbf{v}), S_{\text{global}}(\hat{s}_q, \mathbf{v})) + \mathcal{L}_h(S_{\text{global}}(s_q, \mathbf{v}), S_{\text{global}}(s_q, \hat{\mathbf{v}})))]$$

- Combined VSM loss:

$$\mathcal{L}_{\text{VSM}} = \lambda_1 \mathcal{L}_{\text{local}} + \lambda_2 \mathcal{L}_{\text{global}}$$

In this model, s_q represents the sampled query from all subtitle sentences, \mathbf{v} is the entire video clip, and $\mathbf{V}_{\text{temp}} \in \mathbb{R}^{N_v \times d}$ is the final visual frame representation generated by a temporal Transformer. The query vector $\mathbf{q} \in \mathbb{R}^d$, start and end indices $y_{\text{st}}, y_{\text{ed}} \in \{1, \dots, N_v\}$, and the probability vectors $\mathbf{p}_{\text{st}}, \mathbf{p}_{\text{ed}} \in \mathbb{R}^{N_v}$ are derived from the scores. The hinge loss function \mathcal{L}_h is used for both positive and negative query-video pairs, where (s_q, \mathbf{v}) is a positive pair and $(s_q, \hat{\mathbf{v}}), (\hat{s}_q, \mathbf{v})$ are negative pairs. The margin hyper-parameter δ and balancing factors λ_1, λ_2 are key components of this framework.

9.3.3.8 Frame Order Modeling (FOM)

Frame order modeling (FOM) is conceptualized as a classification challenge within the HERO model's context, focusing on accurately predicting the chronological order of a given set of video frames (Li et al., 2020b). The primary goal of FOM is to determine the original sequence of timestamps for a subset of frames extracted from a video, thereby testing the model's understanding of temporal dynamics and narrative flow in video content.

The FOM objective is formulated as a loss function, mathematically expressed as:

$$\mathcal{L}_{\text{FOM}} = -\mathbb{E} \left[\sum_{i=1}^R \log \mathbf{P}[r_i, t_i] \right] \quad (9.15)$$

where:

- R denotes the total number of frames that have been reordered and is subject to classification.
- i represents the index within the reordered set, ranging from 1 to R .
- t_i symbolizes the true timestamp position of the i^{th} frame within the video, which spans from 1 to N_v , where N_v is the total number of frames in the video.
- r_i is the index corresponding to the reordered position of the i^{th} frame.
- \mathbf{P} is a probability matrix of dimensions $N_v \times N_v$, where each element $P[r_i, t_i]$ indicates the model's predicted probability that the frame at reordered position r_i corresponds to timestamp t_i .

9.3.4 MMLLM Tuning and Enhancements

Following the pre-training phase, MMLLMs can be further enhanced to improve their adaptability, reasoning, and task generalization capabilities. This enhancement

is achieved through various methodologies, three of which are presented here: *multimodal instruction tuning* (MM-IT), which refines models to follow instructions for a broad spectrum of tasks; *multimodal in-context learning* (MM-ICL), which enables models to apply preexisting knowledge to new tasks presented within input prompts; and the *multimodal chain-of-thoughts* (MM-COT) approach, which enables more transparent and logical reasoning by the model in solving complex problems.

9.3.4.1 Multimodal Instruction Tuning

Instruction tuning (IT) diverges from the data-heavy demands of traditional supervised fine-tuning and the limited improvements of prompting methods in few-shot scenarios by aiming to generalize task performance beyond initial training data (Sect. 4.2). Building on this, *multimodal instruction tuning* (MM-IT) adapts IT principles to enhance LLMs through fine-tuning multimodal datasets structured around instructional tasks (Liu et al., 2024; Zhao et al., 2023; Zhu et al., 2023). This approach empowers LLMs to handle new tasks by interpreting instructions efficiently, markedly boosting zero-shot learning abilities across various modalities.

```
<BOS> Below is an instruction that describes a task. Write a response that
appropriately completes the request.

###Instruction: <instruction>
###Input: {<image>, <text>}
###Response: <output><EOS>
```

Fig. 9.4: Multimodal instruction tuning template for visual question answering task.

A multimodal instruction sample is represented as a triplet, $(\mathcal{I}, \mathcal{M}, \mathcal{R})$, encapsulating the instruction, multimodal input, and the ground truth response, respectively. The model's task, governed by parameters θ , is to predict the answer based on both the instruction and the multimodal input:

$$\mathcal{A} = f(\mathcal{I}, \mathcal{M}; \theta) \quad (9.16)$$

Here, \mathcal{A} signifies the predicted answer. The training objective often adheres to the original auto-regressive objective, compelling the MMLM to predict the subsequent response token. This objective is mathematically expressed as:

$$\mathcal{L}(\theta) = - \sum_{i=1}^N \log p(\mathcal{R}_i | \mathcal{I}, \mathcal{R}_{<i}; \theta) \quad (9.17)$$

where N denotes the length of the ground-truth response, highlighting the model’s aim to accurately generate the next token in the response sequence based on the preceding context and instruction. Fig. 9.4 presents a sample template for a visual question answering task, and Table 9.3 presents a selection of the most commonly used datasets for multimodal instruction tuning

Table 9.3: Multimodal Instruction Tuning Datasets. In the table, the symbols represent the transition from input to output modalities, where I->O denotes Input to Output, T for Text, I for Image, V for Video, A for Audio, B for Bounding box, and 3D for Point Cloud.

Dataset Name	I->O	Size (#Instances)
MiniGPT-4’sIT	I+T->T	5K
StableLLaVA	I+T->T	126K
LLaVA’sIT	I+T->T	150K
SVIT	I+T->T	3.2M
LLaVAR	I+T->T	174K
ShareGPT4V	I+T->T	-
DRESS’sIT	I+T->T	-
VideoChat’sIT	V+T->T	11K
Video-ChatGPT’sIT	V+T->T	100K
Video-LLaMA’sIT	I/V+T->T	171K
InstructBLIP’sIT	I/V+T->T	~1.6M
X-InstructBLIP’sIT	I/V/A/3D+T->T	~1.8M
MIMIC-IT	I/V+T->T	2.8M
PandaGPT’sIT	I+T->T	160K
MGVLID	I+B+T->T	-
M3IT	I/V/B+T->T	2.4M
LAMM	I+3D+T->T	196K
BuboGPT’sIT	(I+A)/A+T->T	9K
T2M	T->I/V/A+T	14.7K
MosIT	I+V+A+T->I+V+A+T	5K

9.3.4.2 Multimodal In-context Learning

In-context learning (ICL) equips LLMs to understand and perform tasks by learning from a few examples, often with instructions. This method is distinct from traditional supervised learning which requires extensive data. This method enables LLMs to handle novel tasks without additional training. Unlike instruction tuning, which fine-tunes models on instructional datasets, ICL

leverages the model's pre-trained capabilities to adapt to new tasks during inference, bypassing the need for further model updates.

As the concept of ICL extends into the multimodal domain, it evolves into *multimodal in-context learning* (MM-ICL), enriching the learning process with diverse modalities (Gupta and Kembhavi, 2023). MM-ICL incorporates a demonstration set alongside the original sample at the inference stage, enhancing the learning context with multiple in-context examples.

```
<BOS>Below are some examples and an instruction that describes a task. Write
a response that appropriately completes the request.

###Instruction: "Generate captions for the following images."
###Image: <image: A cat sitting on couch>
###Response: "A cat sitting on a couch."
###Image: <image: A group of people hiking on a mountain trail>
###Response: "A group of adventurers trekking on mountain.

###Image: <image: A car parked in front of a house>
###Response: <EOS>
```

Fig. 9.5: Multimodal In-context Learning for Caption Generation task

Fig. 9.5 depicts an example of MM-ICL for caption generation with two examples. The structure of these examples, including their quantity, can be adjusted flexibly, acknowledging that model performance often hinges on the sequence of presented examples. We also list in Table 9.4 a few critical datasets for MM-ICL.

Table 9.4: Multimodal In-context Learning Dataset

Dataset	Modality	Size	Notes
MM-ICL	Image-Text	5.8M	Includes interleaved text-image inputs and multimodal in-context learning inputs constructed manually.
MIMIC-IT	Image-Text	2.8M	Provides multimodal instruction-response pairs to improve VLMs in perception, reasoning, and planning across multiple languages.

9.3.4.3 Multimodal Chain-of-Thought

Multimodal chain-of-thought (MM-CoT) is an extension of the chain-of-thought concept in LLMs, which is recognized for its effectiveness in complex reasoning tasks. CoT involves LLMs generating the final answer and the intermediate reasoning steps, akin to human cognitive processes. MM-CoT adapts this unimodal CoT to a multimodal context, requiring initial modality bridging. This bridging can be achieved by fusing features or translating visual inputs into textual descriptions. Regarding learning paradigms, MM-CoT can be developed through fine-tuning or through training-free few/zero-shot learning, each with varying sample size requirements.

In their research, [Lian et al. \(2023\)](#) use ChatGPT to synthesize clues from multiple descriptions provided by human annotators into a cohesive summary, focusing on key behaviors and expressions, and then use this consolidated insight to deduce the subject's underlying emotional state accurately, as shown in Fig. 9.6.

Step 1. Prompt for Clue Summarization

Multi-paragraph descriptions of a video is given below. Please summarize these descriptions as follows:
 1. Please unify the subject of multiple paragraphs of "Clue Description" into "he".
 2. Please summarize the multiple paragraphs of "Clue Description", delete repeated words, phrases or sentences, and describe the final result in complete sentences.
 3. Check punctuation.
 Input:
 "Clue Description 1": {clue1}
 "Clue Description 2": {clue2}
 ...
 "Clue Description N": {clueN}
 Output

Step 2. Prompt for Emotion Summarization with Example

Please summarize the person's emotional state:
 Input: He looks happy but is actually anxious.
 Output: anxious

 Input: {prediction}
 Output

Fig. 9.6: Multimodal chain-of-thought for emotion detection through video clip annotations as clues from human

The configuration and pattern of reasoning chains in MM-CoT can be broadly classified into two types:

1. **Adaptive Configuration:** In this approach, LLMs autonomously determine the length of the reasoning chain (Wu et al., 2023a). This flexibility allows the model to adapt the reasoning process to the complexity of the task, ensuring a more tailored and potentially more accurate response. It is particularly beneficial in scenarios where the depth of reasoning required can vary significantly from one task to another.
2. **Predefined Configuration:** Contrary to the adaptive approach, the length of the reasoning chain is predetermined here (Himakunthala et al., 2023). This setup provides a consistent and uniform structure for reasoning across different tasks. While this approach might simplify the model's operation, it may limit the depth of reasoning in more complex scenarios.

Beyond the configuration, the generation pattern of the reasoning chain itself is another area in MM-CoT and provides the following choices:

1. **Infilling-Based Pattern:** This pattern involves deducing intermediate steps to logically connect the surrounding context, effectively filling the gaps in the reasoning process (Himakunthala et al., 2023). It requires the model to identify and bridge missing links in a sequence of thoughts, ensuring a coherent and logical flow of ideas. Consider a task where the model is given a sequence of images depicting a story and is asked to narrate the events. The infilling-based pattern would require the LLM to fill in the narrative gaps between the images, ensuring a coherent storyline.
2. **Predicting-Based Pattern:** In contrast, the prediction-based pattern extends the reasoning chain forward based on given conditions such as instructions or the history of previous reasoning steps (Wu et al., 2023a). This approach requires the model to understand the current context and anticipate logical continuations, synthesizing new steps in the reasoning chain. When an LLM is asked to predict the next scene in a visual story, the prediction-based pattern involves extending the narrative based on the given images and textual descriptions. This requires the model to anticipate future events or actions, building upon the existing context.

Some well-known datasets for MM-CoT reasoning are described in Table 9.5.

9.3.5 Multimodal RLHF

MMLMs face more challenges than do LLMs trained on a single modality due to the complexity of integrating and interpreting information across diverse data types. Similar to its application in unimodal LLMs, RLHF can address numerous issues in multimodal LLMs, including incorporating human preferences and choices, integrating human feedback into descriptions, and generating responses that adhere

Table 9.5: Multimodal Chain-of-Thought Dataset

Dataset	Modality	Size	Notes
EMER	Video-Text	100	Focuses on explainable emotion-based reasoning, offering clues and summarization for reasoning tasks.
EgoCOT	Video-Text	3,670 hours	Embodied planning dataset on a large scale for embodied scenario planning.
VIP	Video-Text	3.6M, 1.5K test	Designed for Video Chain-of-Thought evaluation, featuring inference-time challenges with extensive caption data.
ScienceQA	Image-Text	21K Q-A	Multimodal, multi-choice question dataset across science and diverse domains for in-depth analysis.

to safety and ethical standards. We will highlight some of the research in the field that addresses trustworthiness and methods to incorporate human preferences and alignment.

! Practical Tips

[Li et al. \(2023\)](#) focused on using preference distillation to produce helpful and anchored responses in the visual context. The research introduced the VLFeedback dataset, which contains 80,000 multimodal instructions, with responses from 12 LVLMs and preference annotations from GPT-4V. The findings demonstrate that the Silkie model, refined with this dataset, significantly outperforms the base model on various benchmarks. Compared with human-annotated datasets, the dataset effectively boosts the perception and cognitive abilities of LVLMs and shows advantages in terms of scalability and broader performance improvements.

RLHF-V is an RLHF-based approach aimed at improving the trustworthiness of MMLLMs by aligning their behavior with fine-grained human feedback ([Yu et al., 2023](#)). It addresses a critical issue existing MMLLMs face: the tendency to produce hallucinated text not factually grounded in the associated images, which compromises their reliability for real-world applications, especially those with high stakes. The RLHF-V framework collects human preferences through segment-level corrections for hallucinations and applies dense, direct preference optimization based on this feedback. Through extensive experiments across five benchmarks involving both automatic and human evaluations, RLHF-V is shown to significantly enhance the trustworthiness of MMLLM behaviors while demonstrating promising data and computational efficiency.

! Practical Tips

In their study, Sun et al. (2023) presented a new alignment algorithm, “Factually Augmented RLHF”, which enhances the existing reward model by integrating factual content, including image captions and accurate multichoice answers. This strategy aims to address and reduce the occurrence of reward hacking in RLHF, leading to notable improvements in model effectiveness. Additionally, this study enriches the training dataset for vision instruction tuning, which was originally generated by GPT-4, with pre-existing human-authored image-text pairs to bolster the model’s general performance. By applying RLHF to a language multimodal model (LMM) for the first time, the method showed a marked improvement in performance on the LLaVA-Bench dataset, aligning closely with the results of the text-only GPT-4.

9.3.6 Output Projector

The Output Projector, denoted as $\text{OUT_ALIGN}_{T \rightarrow X}$, transforms the signal token representations \mathbf{S}_X , derived from the LLM, into features \mathbf{H}_X that are interpretable by the subsequent Modality Generator MG_X .

Specifically, for a given modality-text dataset $\{(\mathbf{l}_X, t)\}$, the process starts with input t being processed by the LLM to yield \mathbf{S}_X , which is subsequently converted into \mathbf{H}_X .

The primary objective is to ensure that \mathbf{H}_X aligns closely with the modality generator’s understanding, as defined by:

$$\arg \min_{\text{OUT_ALIGN}_{T \rightarrow X}} \mathcal{L}_{\text{mse}}(\mathbf{H}_X, \tau_X(t)), \quad (9.18)$$

where

$$\mathbf{H}_X = \text{OUT_ALIGN}_{T \rightarrow X}(\mathbf{S}_X). \quad (9.19)$$

\mathcal{L}_{mse} represents the mean squared error loss, aiming to minimize the discrepancy between the projected features \mathbf{H}_X , and τ_X is the textual condition encoder in MG_X . This optimization process primarily utilizes processing texts without requiring direct multimodal inputs such as audio or visual inputs X .

! Practical Tips

The Output Projector is usually implemented using a Tiny Transformer or an MLLP, focusing on efficiency and adaptability.

9.3.7 Modality Generator

The Modality Generator MG_X is engineered to generate outputs across various modalities, effectively translating encoded features into multimodal content.

! Practical Tips

This component often employs SOTA latent diffusion models (LDMs) for synthesizing outputs specific to each modality, such as images, videos, and audio (Zhao et al., 2022). Commonly used implementations include Stable Diffusion for image synthesis, Zeroscope for video synthesis, and AudioLDM-2 for audio output generation (Cespense, 2023; Liu et al., 2023; Rombach et al., 2022).

The process leverages \mathbf{H}_X from the output projector as conditional inputs to guide the denoising step, which is essential for generating high-quality multimodal content.

During the training phase, the original content is first encoded into latent features z_0 using a pre-trained variational autoencoder (VAE) (Kingma and Welling, 2013). This latent representation is then perturbed with noise ϵ to produce a noisy latent feature z_t .

A pre-trained Unet (ϵ_X) is normally used for computing the conditional LDM loss \mathcal{L}_{X_gen} (Ronneberger et al., 2015). Given as:

$$\mathcal{L}_{X_gen} := \mathbb{E}_{\epsilon \sim N(0,1), t} \|\epsilon - \epsilon_X(z_t, t, \mathbf{H}_X)\|_2^2 \quad (9.20)$$

$IN_ALIGN_{X \rightarrow T}$ and $OUT_ALIGN_{T \rightarrow X}$ are optimized by minimizing \mathcal{L}_{X_gen} .

9.4 Benchmarks

This section overviews selected benchmark datasets for evaluating multimodal LLMs across various modalities and tasks. Although not exhaustive, this compilation emphasizes benchmarks notable for their task diversity, modality range, and widespread application in the field. For a more detailed or comprehensive list of benchmark datasets, readers are encouraged to refer to the work of Yin et al. (2023).

1. **CMMU** is a comprehensive collection of 12,000 multimodal questions, manually curated from college exams, quizzes, and textbooks across six fundamental disciplines: Art and Design, Business, Science, Health and Medicine, Humanities and Social Science, and Tech and Engineering (Zhang et al., 2024b). This diversity mirrors that of its counterpart, MMMU, which extends across 30 distinct subjects. The dataset is characterized by its variety, featuring 39 different

- types of images—including charts, diagrams, maps, tables, music sheets, and chemical structures—to test a wide range of multimodal understanding capabilities.
- 2. **MMCbench** presents a detailed framework for assessing LMMs, emphasizing their resilience and self-consistency when faced with typical corruption challenges (Zhang et al., 2024c). It focuses on the interplay between text, image, and speech modalities, encompassing key generative tasks such as text-to-image, image-to-text, text-to-speech, and speech-to-text.
 - 3. **MMVP** evaluates the visual capabilities of multimodal LLMs through VQA tasks (Tong et al., 2024). It includes a directory of 300 test images and a CSV file with questions and correct answers.
 - 4. **TimeIT** addresses six timestamp-related video tasks and incorporates 12 datasets from various domains (Ren et al., 2023). It focuses on time-sensitive long video understanding tasks such as dense video captioning, video grounding, video summarization, video highlight detection, step localization, and transcribed speech generation, with a total training data size of 124,861 instances.
 - 5. **ViP-Bench** is a benchmark designed to test multimodal models on visual reasoning capabilities through 303 image-question pairs derived from MM-Vet, MM-Bench, and Visual Genome (Cai et al., 2023). It aims to evaluate models on six key aspects of visual understanding at the region level: recognition, OCR, knowledge, math, object relationship reasoning, and language generation. The benchmark employs GPT-4 for grading multimodal model responses from 0 to 10, offering a quantitative comparison tool.
 - 6. **M3DBench** compiles more than 320K pairs of 3D multimodal instruction-following data, including over 138K instructions with unique multimodal prompts (Cai et al., 2023). It utilizes existing datasets and instructions generated by LLMs for diverse 3D tasks. The dataset spans object detection to question answering, with instructions and responses tailored to each task. High data quality is ensured by filtering out irrelevant responses through pattern matching, making M3DBench a robust dataset for 3D instruction-following evaluations.
 - 7. **Video-Bench** introduces a comprehensive benchmark for assessing Video LLMs (Ning et al., 2023). This benchmark encompasses ten carefully designed tasks that gauge Video-LLMs' proficiency in video-specific understanding, leveraging prior knowledge for question-answering, and skills in comprehension and decision-making and has a size of approximate 15,033.
 - 8. **Bingo** classifies instances of model failures and successes in multimodal understanding, comprising 190 instances of failures contrasted with 131 instances of success (Cui et al., 2023). Each instance, paired with one or two questions, falls into the categories of “Interference” (image-to-image and text-to-image) and “Bias” (region, OCR, and factual). The benchmark aims to dissect the nuanced reasons behind hallucinations in responses, offering a detailed exploration of bias within GPT-4V(ision) across diverse images reflecting cultural, linguistic, and factual diversities.
 - 9. **MMHAL-BENCH** is designed to evaluate hallucinations in multimodal models, focusing on hallucination detection with tailored evaluation metrics (Sun

- [et al., 2023](#)). It features 96 image-question pairs across eight question categories and twelve object topics and was specifically constructed to test LMMs against false claims about image contents. The benchmark leverages images from the Open Images validation and test sets to avoid data leakage. It includes comprehensive object meta-categories such as “accessory,” “animal,” and “vehicle.” Responses are evaluated using GPT-4, which assesses the presence of hallucinations by comparing LMM outputs with human-generated answers and the image content.
10. **Sparkles** leverages GPT-4 to construct a multimodal dialog dataset that simulates realistic conversations around images and text, aiming for dialogs that span a variety of real-life scenarios ([Huang et al., 2023](#)). This process uses a two-turn dialog pattern, starting with a user query about images, followed by an assistant’s detailed response, and then introducing a new image for further discussion. The dataset generation emphasizes dialog demonstrations for in-context learning and candidate image descriptions for selecting relevant images, employing detailed textual descriptions to represent images due to GPT-4’s text-only input capability.
 11. **SciGraphQA** introduces a large-scale, synthetic, multturn question-answer dataset for academic graphs ([Li and Tajbakhsh, 2023](#)). The dataset encompasses 295,000 samples derived from 290,000 Computer Science or Machine Learning papers from ArXiv (2010–2020). Utilizing Palm-2 generates dialogs based on graphs within these papers, incorporating titles, abstracts, relevant paragraphs, and contextual data. Each dialog averages 2.23 question-answer turns. GPT-4’s evaluation of the dataset’s question-answer match quality averages at 8.7/10 across a 3,000-sample test set.
 12. **LAMM** introduces a comprehensive multimodal instruction tuning dataset comprising 186,000 language-image and 10,000 language-3D instruction-response pairs, utilizing images and point clouds from diverse vision tasks ([Yin et al., 2024](#)). This dataset, constructed through GPT-API and self-instruction methods, includes four types of multimodal instruction-response pairs: daily dialogs, factual knowledge dialogs, detailed descriptions of images and 3D scenes, and visual task dialogs to enhance visual task generalization. It incorporates a variety of 2D and 3D vision tasks, such as captioning, scene graph recognition, VQA, classification, detection, counting, and OCR.

9.5 State-of-the-Art MMLLMs

This section provides an overview of several SOTA MMLLMs, showcasing models that integrate various modalities into their framework. A detailed Table 9.6 is presented, which encapsulates a wide range of well-known multimodal LLMs, each mapping distinct components to the generic framework outlined earlier. Next, we delve into the specifics of three multimodal LLMs, each representing a significant leap in the complexity and capability of handling multimodal data. Starting with

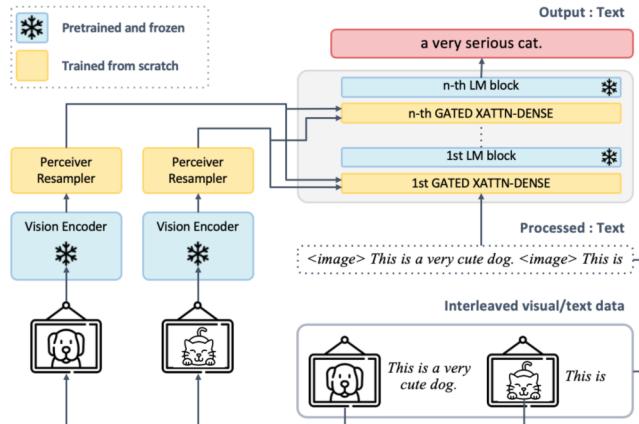


Fig. 9.7: The architecture of the Flamingo model.

Flamingo, which combines vision and language, we then discuss Video-LLaMA, which adds video and audio to text. Finally, we discuss NExT-GPT, which expands to support six different modalities, allowing any conversion between them.

9.5.1 Flamingo (Image-Video-Text)

Flamingo is a pioneering visual language model in MMLMs, heralding advancements in few-shot learning for a broad spectrum of vision and language tasks (Alayrac et al., 2022). It distinguished itself by surpassing the fine-tuned state-of-the-art models in six of sixteen tasks, utilizing considerably less task-specific training data. The subsequent discussion delves into the architectural decisions aligned with components outlined in the preceding framework, highlighting where certain elements, such as multimodal RLHF, the output projector, and the modality generator, are absent. Notably, RLHF techniques were not implemented, and given the textual nature of the outputs, there was no necessity for output projection and generation processes.

9.5.1.1 Modality Encoder

Central to Flamingo is the integration of the Normalizer-Free ResNet (NFNet) F6 as the vision encoder, which employs contrastive learning for vision-text modalities to encode visual inputs efficiently. Flamingo adopts BERT for text encoding, diverging from the conventional use of GPT-2. The model processes embeddings from both vision and text modalities through mean pooling, subsequently projecting them into a joint embedding space to facilitate seamless modality integration.

Table 9.6: This table provides a detailed overview of various multimodal LLMs, highlighting their choices of base models, input-output modalities, modality encoders, input projectors, core LLMs, and modality generators.

Model	I→O	Modality Encoder	Input Projector	LLM	Output Modality Projector
Flamingo	I+V+T→T	I/V: NFNet-F6	Cross-attention	Chinchilla-1.4B/7B/70B	-
BLIP-2	I+T→T	I: CLIP/Eva-CLIP ViT@224	Q-Former w/ Linear Projector	Flan-T5/OPT	-
LLaVA	I+T→T	I: CLIP ViT-L/14	Linear Projector	Vicuna-7B/13B	-
IDEFICS	I+T→T	I: OpenCLIP ViT-H/14	Cross-attention	LLaMA-v1 7B/65B	-
MiniGPT-4	I+T→T	I: Eva-CLIP ViT-G/14	Q-Former w/ Linear Projector	Vicuna-13B	-
X-LLM	I+V+A+T→T	I/V: ViT-G; A: C-Former	Q-Former w/ Linear Projector	ChatGLM-6B	-
VideoChat	V+T→T	V: ViT-G	Q-Former w/ Linear Projector	Vicuna	-
InstructBLIP	I+V+T→T	I/V: ViT-G/14@224	Q-Former w/ Linear Projector	Flan-T5/Vicuna	-
Video-LLaMA	I+V+A+T→T	I/V: EVA-CLIP ViT-G/14; A: ImageBind	Q-Former w/ Linear Projector	Vicuna/LLaMA	-

Continued on next page

Table 9.6 – *Continued from previous page*

Model	I→O	Modality Encoder	Input Projector	LLM	Output Projector	Modality Generator
BuboGPT	I+A+T→T	I:CLIP/Eva-CLIPViT; A:ImageBind	Q-Former w/ Linear Projector	Vicuna (Frozen)	-	-
Qwen-VL- (Chat)	I+T→T	I:ViT@448 initialized from OpenClip's ViT-bigG	Cross-attention	Owen-7B (PT: Frozen; IT: PEFT)	-	-
Palm-E	I+3D+T→T	I:ViT, 3D:OSRT	Affine Transformations	PaLM (PT: Frozen; co-training)	-	-
MACAW- LLM	I+V+A+T→T	I/V: CLIP; A:Whisper	Linear Projector	Llama-7B	-	-
NExT-GPT	I+V+A+T→I+V+A+T	I/V/A:ImageBind	Linear Projector	Vicuna-7B (PEFT)	Tiny Transformer	I:StableDiffusion; V:ZeroScope;
MiniGPT-5	I+T→I+T	I:Eva-CLIP ViT-G14	Q-Former w/ Linear Projector	Vicuna-7B (PEFT)	Tiny Transformer	A:AudioLM I:StableDiffusion- 2
LLaVA-1.5	I+T→T	I:CLIP ViT-L@336	MLP	Vicuna-v1.5- 7B/13B (PT: Frozen; IT: PEFT)	-	-
X- InstructBLIP	I+V+A+3D+T→T	I/V:Eva-CLIPViT-G14; A:BEATS; 3D:ULIP-2	Q-Former w/ Linear Projector	Vicuna-v1.1- 7B/13B (Frozen)	-	-
CoDi-2	I+V+A+T→I+V+A+T	I/V/A:ImageBind	MLP	Llama-2-Chat-7B (PT: Frozen; IT: PEFT)	MLP	I:StableDiffusion- 2.1; V:ZeroScope-v2; A:AudioLM-2

9.5.1.2 Input Projector

Flamingo's ability to handle visual inputs, including images and videos, necessitates addressing the variability in feature outputs. This is achieved through the perceiver resampler component, which standardizes outputs to a consistent 64 visual tokens, as shown in Fig. 9.7. The modality alignment between language and visual modalities is achieved by incorporating cross-attention (GATED XATTN-DENSE) layers among the preexisting frozen language model layers, enhancing the attention mechanism toward visual tokens during text token generation.

9.5.1.3 Pre-training: Core LLMs, Datasets and Task-Specific Objectives

The foundation of Flamingo is built upon the Chinchilla language model by freezing nine of the pre-trained Chinchilla LM layers. The training regimen spans four distinct datasets: M3W (Interleaved image-text), ALIGN (Image-text pairs), LTIP (Image-text pairs), and VTP (Video-text pairs). This approach enables Flamingo to predict subsequent text tokens y by considering both preceding text and visual tokens, quantified as:

$$p(y|x) = \prod_{\ell=1}^L p(y_\ell|y_{<\ell}, x_{\leq \ell}). \quad (9.21)$$

The training loss function is defined as a weighted sum of the expected negative log-likelihoods of the generated text across the datasets, where λ_m signifies the training weight for the m -th dataset:

$$\sum_{m=1}^M \lambda_m \mathbb{E}_{(x,y) \sim \mathcal{D}_m} \left[- \sum_{\ell=1}^L \log p(y_\ell|y_{<\ell}, x_{\leq \ell}) \right], \quad (9.22)$$

where \mathcal{D}_m and λ_m represent the m -th dataset and its associated weighting, respectively.

9.5.1.4 MMLM Tuning and Enhancements

The Flamingo models exhibit exceptional performance in in-context learning, outclassing state-of-the-art models fine-tuned for specific tasks despite relying on a singular set of model weights and a limited number of 32 task-specific examples – a thousand times fewer task-specific training examples than existing state-of-the-art approaches. The analysis presents support examples as pairs of images or videos (visual inputs) with corresponding text (expected responses or task-specific information, such as questions) to predict responses for new visual queries. The default prompts use are “Output: output” for tasks excluding question-answering, and

“Question: question Answer: answer” for question-answering or visual dialog tasks.

9.5.2 Video-LLaMA (Image-Video-Audio-Text)

[Zhang et al. \(2023\)](#) introduced *Video-LLaMA*, a multimodal framework designed to augment LLMs with the ability to comprehend visual and auditory elements in videos. Unlike prior initiatives that have enabled LLMs to process visual or audio signals, *Video-LLaMA* takes a comprehensive approach by incorporating cross-modal training leveraging frozen pre-trained visual and audio encoders alongside frozen LLMs. The framework is distinctive for its focus on video comprehension, addressing two key challenges: capturing the temporal dynamics within visual scenes and effectively merging audio-visual information. The experiments that were conducted reveal *Video-LLaMA*’s remarkable ability to facilitate audio and video-grounded dialogs, underscoring its viability as an advanced prototype for audio-visual AI assistants. The following section explores the architectural choices corresponding to the components presented in the prior framework, identifying the absence of specific elements, again including multimodal RLHF, the output projector, and the modality generator. It is important to note that RLHF methodologies were not applied, and the requirement for output projection and generation was not needed because the output was only the text.

9.5.2.1 Modality Encoder

For the encoding of visual inputs, the branch leverages a frozen visual encoder with ViT G/14 model from EVA-CLIP and a BLIP-2 Q-former to process video frames, as shown in Fig. 9.8. Each frame is transformed into a set of image embedding vectors, resulting in a sequence of frame representations $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_N]$, where $\mathbf{v}_i \in \mathbb{R}^{K_f \times d_f}$ denotes the d_f -dimensional image embeddings for the i -th frame.

The pre-trained Imagebind is used as the audio encoder to address the auditory component of videos ([Girdhar et al., 2023](#)). The videos are uniformly sampled as M segments of 2-second audio clips. Each of these clips is then transformed into spectrograms utilizing 128 Mel spectrogram bins, effectively capturing the audio’s spectral features. The audio encoder processes these spectrograms, converting each into a dense vector representation. As a result, the compiled audio representation for a given video is denoted as $\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_M]$, where each \mathbf{a}_i represents the encoded feature vector of the i -th audio segment.

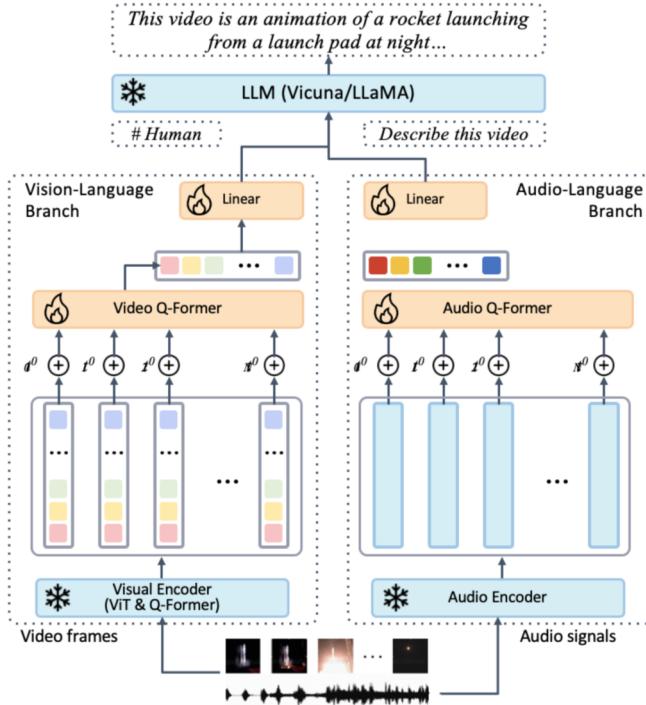


Fig. 9.8: The architecture of Video-LLMA.

9.5.2.2 Input Projector

In both the video and audio branches, the Q-Former combined with a linear projection is used to align the encoded modalities with textual data.

For the vision-language branch, position embeddings are incorporated to imbue these representations with temporal context. This is because the frame representations, \mathbf{v}_i , are derived from the frozen image encoder and thus lack inherent temporal information. Next, the position-encoded frame representations are introduced into the Video Q-former. The purpose is to fuse the frame-level representations into a consolidated video representation, achieving a set of k_V video embedding vectors, each of dimension d_v . Consequently, this yields a comprehensive video representation $\hat{\mathbf{v}} \in \mathbb{R}^{k_V \times d_v}$, effectively capturing both the visual and the temporal dynamics of the video content. A linear layer is introduced to transform the video embedding vectors into video query vectors to align the video representations with the input requirements of the LLMs. These vectors match the dimensionality of the LLM's text embeddings, ensuring video and textual data compatibility. During the forward pass, video query vectors are concatenated with text embeddings, serving as a video soft prompt. This concatenation effectively guides the frozen LLMs to generate text

outputs conditioned on the video content, thereby integrating video information into the multimodal understanding process.

Similar to the vision-language branch, a position embedding layer is applied to incorporate temporal information into these audio segments in the audio-language branch. This addition ensures that temporal dynamics, which are critical for understanding the sequence and evolution of sounds within the video, are captured. Following this temporal encoding, the audio Q-former is used to fuse the features of different audio segments into a unified audio representation. Mirroring the vision-language branch, a linear layer is employed to map the comprehensive audio representation into the embedding space of the LLMs.

9.5.2.3 Pre-training: Core LLMs, Datasets and Task-Specific Objectives

Video-LLaMA leverages Vicuna-7B, as the core LLM for its multimodal understanding and generation capabilities.

Video-LLaMA's pre-training process utilizes the Webvid-2M dataset, a collection of short videos accompanied by textual descriptions from stock footage websites, to train its vision-language branch. This dataset and the CC595k image caption dataset derived from CC3M and refined by Liu et al. (2024) form the basis for a video-to-text generation task during pre-training. The audio-language branch in Video-LLaMA utilizes the ImageBind audio encoder, which is inherently aligned across multiple modalities hence no pre-training is required.

9.5.2.4 MMLM Tuning and Enhancements

Following its pre-training phase for the Video-Language branch, Video-LLaMA demonstrated proficiency in generating content based on video information. However, its ability to adhere to specific instructions showed a need for enhancement, and instruction-based fine-tuning was performed. The datasets employed for this purpose included:

1. A collection of 150K image-based instructions from the LLaVA dataset.
2. A set of 3K image-based instructions sourced from MiniGPT-4.
3. An assembly of 11K video-based instructions from VideoChat.

For the audio tuning process in Video-LLaMA, the approach addresses the challenge posed by the scarcity of audio-text data by incorporating the vision-text datasets mentioned above into the training regimen. This strategy enables these components to learn the alignment between the common embedding space produced by the ImageBind encoder and the embedding space of the LLMs.

9.5.3 NExT-GPT (Any-to-Any)

NExT-GPT is a general-purpose, multimodal LLM that integrates a large language model with multimodal adaptors and diffusion decoders, allowing it to handle and generate text, images, videos, and audio content (Wu et al., 2023c). It is fine-tuned on a small number of parameters, making training cost-effective and expanding to new modalities straightforward. The system also features modality-switching instruction tuning and a high-quality dataset for improved cross-modal understanding and generation, demonstrating the feasibility of creating a unified AI agent for diverse modalities.

9.5.3.1 Modality Encoder

NExT-GPT employs ImageBind as a universal encoder across all modalities, diverging from the traditional approach of modality-specific encoders used in many previous studies. ImageBind demonstrated the capability to forge a joint embedding space encompassing multiple modalities, eliminating the need to train on data representing every possible modality combination and showing state-of-the-art results.

9.5.3.2 Input Projector

NExT-GPT utilizes a linear projection layer (4 million parameters) to transform the outputs through ImageBind into language-like representations, thus aligning all modalities in a format that the LLM can readily understand and process.

9.5.3.3 Pre-training: Core LLMs, Datasets and Task-Specific Objectives

NExT-GPT uses Vicuna2, an open-source text-based LLM widely adopted in existing multimodal LLMs, as its core LLM.

In stage 1 of pre-training NExT-GPT, everything in the pipeline – the encoders that process inputs, the decoders that process outputs, and the LLM – are kept frozen, and only the input alignment through the linear projection layer is adapted through back-propagation. This training strategy aims to align various modalities – images, audio, or videos – with their corresponding textual descriptions (captions) using specific datasets for each modality. The CC3M dataset, comprising 3 million image-caption pairs, is employed for image modality alignment. Video modality alignment utilizes the WebVid-10M dataset, which contains 10.7 million video-caption pairs from diverse content sourced from stock footage websites, totaling 52,000 hours of video. AudioCaps provides a foundation for the audio modality with around 46,000 pairs of audio clips and human-written text, curated through crowdsourcing on the AudioSet dataset. This method trains NExT-GPT to generate captions that match the input modality against a benchmark “gold” caption.

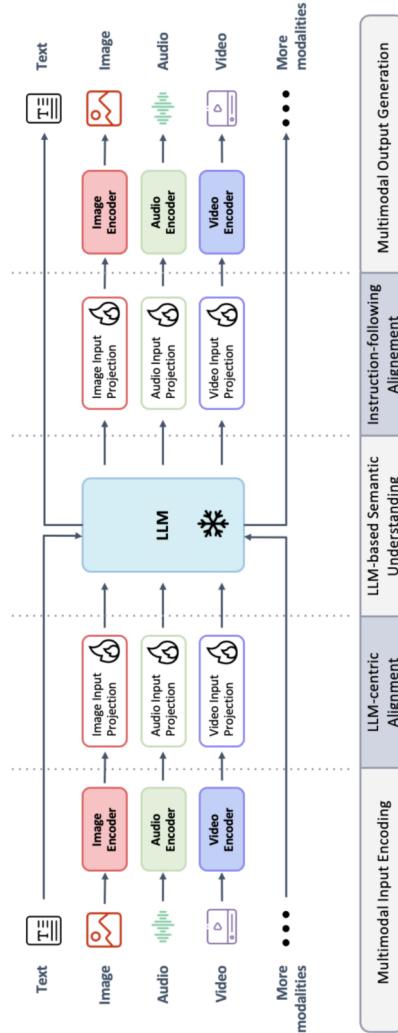


Fig. 9.9: The architecture of NExT-GPT.

During stage 2 of training, the focus is on the output projection layers. The ImageBind, LLM, and input projection layers are kept frozen, and the training employs the same datasets used in the initial stage: the CC3M dataset for images, the WebVid-10M dataset for videos, and the AudioCaps dataset for audio.

9.5.3.4 MMLM Tuning and Enhancements

Stage 3 of NExT-GPT’s training uses multimodal instruction tuning, a process designed to refine the model’s ability to understand and execute complex instructions across different modalities. In this phase, the core LLM (Vicuna2) is fine-tuned using the LoRA technique, and both the input and output projection layers are fine-tuned during this stage, but without altering the encoders or decoders. The following datasets are used:

1. **“Text+X” to “Text” Data** Here, “X” stands for any nontextual modality included in NExT-GPT (i.e., image, video, or audio). The process involves feeding the model inputs that combine textual information with one of these modalities, to generate textual responses that accurately reflect the combined input. The datasets used include LLaVA-Instruct-150K (vision-language), miniGPT-4 image description dataset, and Videochat video instruction dataset.
2. **“Text” to “Text+X” Data** This dataset is used to generate not only textual outputs but also multimodal content, referred to as “Text to Text+X”. A dataset for text-to-multimodal (T2M) data was created, utilizing a collection of “X-caption” pairs from existing corpora and benchmarks such as Conceptual Captions, MS COCO (Microsoft Common Objects in Context), AudioCaps, and more. By employing templates and GPT-4, varied textual instructions that include these captions are produced, forming a dataset that supports the generation of both textual and multimodal outputs from the text prompts.
3. **MosIT Data** In crafting NExT-GPT, a key innovation was the development of a specialized dataset named Modality-switching Instruction Tuning (MosIT) to refine the model’s instruction-following capabilities across different modalities. Recognizing the shortfall in existing datasets, which did not fully capture the complexity of real-life interactions between users and AI across different formats, the creators of NExT-GPT identified a need for a more sophisticated approach. To ensure that the dataset included a rich variety of multimodal content the team sourced materials from external resources, including YouTube for videos, and various AI-generated content (AIGC) tools such as Stable-XL and Midjourney for creating images and audio clips. Each dialog within the MosIT dataset consists of 3-7 turns, with the human-AI exchanges designed to shift modalities between inputs and outputs, resulting in a dataset of 5,000 dialogs.

9.5.3.5 Output Projector

The output projector in NExT-GPT translates tokens generated by the LLM into formats suitable for modality-specific decoders. To accomplish this, NExT-GPT employs TinyTransformer (31 million parameters), which is dedicated to handling the conversion for each specific modality. The training of these output projectors occurs during the second and third stages of the overall training process.

9.5.3.6 Modality Generator

The final step in NExT-GPT involves creating outputs for different modalities with specialized decoders. This begins when the system receives multimodal signals and instructions from the LLM, which are then converted by Transformer-based layers into formats that the decoders can process. For this purpose, NExT-GPT uses leading diffusion models tailored for each modality: Stable Diffusion for images, Zeroscope for videos, and AudioLDM for audio. These models are integrated into the system as conditioned diffusion models, and fed with the transformed signal representations to generate the final content in the specified modality.

9.6 Tutorial: Fine-Tuning Multimodal Image-to-Text LLMs

9.6.1 Overview

Having discussed the theoretical underpinnings of MMLLMs in detail, we can now test the behavior of a “Text+X” to “Text” model. For this demonstration, we choose images to represent the the modality “X”. Image/text-to-image models are useful for detecting specific properties of images, categorizing the events occurring in the images, and generating automated captions, among other tasks. In this tutorial, we test the out-of-the-box capabilities of a SOTA MMLLM on image labeling and captioning and explore ways to improve performance with fine-tuning and few-shot prompting.

We set up two experiments with the same dataset to accomplish this goal. First, we will ask the model to identify which sport is in each image, both in a zero-shot framework and in a fine-tuned framework, and compare the results. Second, we will ask the model to write simple captions of what is occurring within the images, comparing zero-shot, few-shot, and fine-tuning modes.

Goals:

- Successfully set up and prompt the IDEFICS 9-billion parameter model with arbitrary text and images.
- Generate zero-shot predictions for the 100SIC test set and try to improve performance with QLoRA fine-tuning.
- Generate zero-shot captions for the 100SIC test set and compare them to fine-tuned and in-context learning captions.

Please note that this is a condensed version of the tutorial. The full version is available at <https://github.com/springer-llms-deep-dive/llms-deep-dive-tutorials>.

9.6.2 Experimental Design

There are many MMLM to select from, so to narrow our choices we consider models small enough to be QLoRA-tuned in a Google Colab notebook and which are already integrated with Huggingface so that we can easily take advantage of their PEFT and fine-tuning routines. With these considerations, we choose as our model the 9 billion parameter variant of IDEFICS (Image-aware Decoder Enhanced à la Flamingo with Interleaved Cross-attentionS), an open-source text and image-to text LLM modeled on Flamingo (Laurençon et al., 2023). The model takes arbitrarily interleaved text and images as input and outputs a textual response.

The dataset we choose for this experiment is the 100 Sports Image Classification dataset (100SIC) hosted at Kaggle¹. This set includes many small photos labeled by sport for 100 different sports. It consists of approximately 13,000 training images and 500 test and validation images. For caption fine-tuning, we supplement this dataset with a subset of the flickr30k dataset (Young et al., 2014), a 30,000+ item catalog of image and caption pairs. We used the subset extracted by Shin Thant², who identified flickr30k images of sports.

9.6.3 Results and Analysis

9.6.3.1 Predicting the Sport

We start by loading the model. IDEFICS is too large to predict with and tune on a single moderate GPU effectively, so we will use BitsAndBytes to quantize to 4-bit and fine-tune in the QLoRA paradigm. For sport classification, we adopt the following prompt template:

```
<image>
Question: What sport is in this image?
Answer:
```

Listing 9.1: Sport classification prompt

We use this to generate predictions for every image in the test set and compare the output against the label assigned by the compilers of the dataset:

```
- Zero-shot results:
- 212 / 500 correct
```

Listing 9.2: Zero-shot test set predictions

It thus guessed the correct name for the sport on approximately 42% of the images. Note that we have done a simple exact-match evaluation, so if the model guesses

¹ <https://www.kaggle.com/datasets/gpiosenka/sports-classification/data>

² <https://github.com/ShinThant3010/Captioning-on-Sport-Images>

Table 9.7: Three cherry-picked examples demonstrating three themes of relative classification performance in increasing rareness in the zero-shot vs. fine-tuned approach. For bobsled, the zero-shot model correctly identifies the sport in most cases but does not know which name it should use. For chuckwagon racing, the model is unfamiliar with this obscure sport and guesses other types of equestrian competitions. For tug of war, fine-tuning has actually degraded the model’s predictive power – this would likely improve with additional fine-tuning.

Index	bobsled		chuckwagon racing		tug of war	
	ZS	FT	ZS	FT	ZS	FT
1	bobsledding	bobsled	rodeo	chuckwagon racing	hurling	oxen pulling
2	the u	bobsled	horse-drawn carriage	chuckwagon racing	rugby	tug of war
3	bobsleigh	bobsled	calgary stam-pede rodeo	chuckwagon racing	tug of war	log rolling
4	bobsledding	bobsled	horseback riding	chuckwagon racing	tug of war	log rolling
5	bobsled	bobsled	chariot racing	chuckwagon racing	tug of war	axe throwing

another acceptable name for a sport, it will be considered a missed prediction. We can improve our predictions by fine-tuning the model with the training set. The Llama base model is too large for full fine-tuning, so we employ a QLoRA tuning approach similar to that discussed in the tutorial in Sect. 4.6. Selecting 10 training examples per sport as our train set and adopting the same template as in the zero-shot example to create QA pairs for fine-tuning, we fine-tune the model and again predict on the test set:

```
- Fine-tune results:  
- 419 / 500 correct
```

Listing 9.3: Fine-tune test set predictions

This shows major improvement, moving from 42% to 84% correct. We highlight a few interesting examples in Table 9.7 to demonstrate the details of this improvement.

9.6.3.2 Captioning Photos

A second common use of image-to-text models is generating automated captions. In this section, we test the capabilities of IDEFICS for this task. As before, we can use a simple zero-shot prompt template to query the model to generate a caption. For this exercise, we use an image of a Wake Forest quarterback in a black jersey throwing a pass in a game of American football and the following prompt template:

```
<image>
```

```
Question: What is a caption for this photo?
Answer:
```

Listing 9.4: Sport captioning prompt

When using zero-shot prompting, we get the following response:

```
Question: What is a caption for this photo? Answer: Aaron Murray,
Georgia Bulldogs quarterback, throws a pass during the first
half of the Chick-fil-A Bowl NCAA college football game
against the Nebraska Cornhuskers
```

Listing 9.5: Sport captioning zero-shot

While this is a quarterback throwing a pass, every other piece of information in this response is false. It is not Aaron Murray nor a Georgia Bulldog, and this is not the Chick-fil-A bowl nor a game against Nebraska. All of this information was hallucinated, but notably the final two false facts are not even items that could be determined based on the image alone. Ideally we would like our captions to be straightforward descriptions of the image, downplaying specific identifying information that is not plainly visible in the photograph.

An inexpensive way to improve the model captioning is with in-context examples. For this approach, we pass several examples of training images along with hand-written captions before the target image that we are generating for. With this approach, we get the following output.

```
<image1> Question: What is a caption for this photo? Answer: A
man prepares to throw an ax at a target.
<image2> Question: What is a caption for this photo? Answer: A
woman rolls a bowling ball down a bowling alley.
...
<image5> Question: What is a caption for this photo? Answer:
A man in a white jersey throws a football.
```

Listing 9.6: Sport captioning in-context prompt

Under few-shot conditions, the model has generated “A man in a white jersey throws a football.” This is a slight mistake as the jersey color is black, but the model has formatted the caption according to our preferences and not hallucinated extraneous information like the identity of the player or their opponent. This is a promising avenue with some improvements.

A more expensive approach is to use the sports image/caption pair subset of the flickr30k dataset to fine-tune the model. We use the same QLoRA approach described above and fine-tune the base IDEFICS model with roughly 1600 samples using the same template from the zero-shot example. Once the training is complete, we can generate a caption for our test figure again.

```
A football player in a black uniform is throwing a football.
```

Listing 9.7: Sport captioning fine-tuning output

This response is both concise, similar to the few-shot response, and accurate to the photo. We generate captions for twenty test images using all three approaches as a final comparison, and qualitatively grade the responses by hand, considering both accuracy and style. The final results are:

- Zero-shot results:
 - 7 / 20 acceptable
- In-context results:
 - 11 / 20 acceptable
- Fine-tuning results:
 - 14 / 20 acceptable

Listing 9.8: Test set captioning results

9.6.4 Conclusion

Moderately sized text/image-to-text MMLMs show considerable zero-shot capabilities but are greatly improved with fit-to-task fine-tuning. We have shown how utilizing PEFT can greatly improve image classification and open-ended captioning capabilities, even with little optimization and standard parameter choices. Production applications would clearly benefit from additional care in selecting tuning parameters, training set properties, and the MMLM architecture itself, but only a small amount of effort is required to create a moderately well-functioning image classifier from available open-source software.

References

- Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in Neural Information Processing Systems*, 35:23716–23736, 2022.
- Lalit Bahl, Peter Brown, Peter De Souza, and Robert Mercer. Maximum mutual information estimation of hidden markov model parameters for speech recognition. In *ICASSP'86. IEEE international conference on acoustics, speech, and signal processing*, volume 11, pages 49–52. IEEE, 1986.
- Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.
- Mu Cai, Haotian Liu, Siva Karthik Mustikovela, Gregory P Meyer, Yuning Chai, Dennis Park, and Yong Jae Lee. Making large multimodal models understand arbitrary visual prompts. *arXiv preprint arXiv:2312.00784*, 2023.
- Jean Carletta, Simone Ashby, Sebastien Bourban, Mike Flynn, Mael Guillemot, Thomas Hain, Jaroslav Kadlec, Vasilis Karaikos, Wessel Kraaij, Melissa Kro-

- nenthal, et al. The ami meeting corpus: A pre-announcement. In *International workshop on machine learning for multimodal interaction*, pages 28–39. Springer, 2005.
- Cerspense. Zeroscope: Diffusion-based text-to-video synthesis, 2023.
- Feilong Chen, Minglun Han, Haozhi Zhao, Qingyang Zhang, Jing Shi, Shuang Xu, and Bo Xu. X-llm: Bootstrapping advanced large language models by treating multi-modalities as foreign languages. *arXiv preprint arXiv:2305.04160*, 2023.
- Sanyuan Chen, Yu Wu, Chengyi Wang, Shujie Liu, Daniel Tompkins, Zhuo Chen, and Furu Wei. Beats: Audio pre-training with acoustic tokenizers. *arXiv preprint arXiv:2212.09058*, 2022.
- Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. See <https://vicuna.lmsys.org> (accessed 14 April 2023), 2023.
- Huyen Chip. Multimodality and large multimodal models (lmms), 2023. URL <https://huyenchip.com/2023/10/10/multimodal.html>.
- A Chowdhery, S Narang, J Devlin, M Bosma, G Mishra, A Roberts, P Barham, HW Chung, and C Sutton. S. gehrmannet al.,“palm: Scalinglanguage modeling with pathways,”. *arXiv preprint arXiv:2204.02311*, 2022.
- Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.
- Chenhang Cui, Yiyang Zhou, Xinyu Yang, Shirley Wu, Linjun Zhang, James Zou, and Huaxiu Yao. Holistic analysis of hallucination in gpt-4v (ision): Bias and interference challenges. *arXiv preprint arXiv:2311.03287*, 2023.
- Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.
- Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.
- Yuxin Fang, Wen Wang, Binhui Xie, Quan Sun, Ledell Wu, Xinggang Wang, Tiejun Huang, Xinlong Wang, and Yue Cao. Eva: Exploring the limits of masked visual representation learning at scale. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19358–19369, 2023.
- Kirill Gavrilyuk, Ryan Sanford, Mehrsan Javan, and Cees GM Snoek. Actor-transformers for group activity recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 839–848, 2020.
- Rohit Girdhar, Alaaeldin El-Nouby, Zhuang Liu, Mannat Singh, Kalyan Vasudev Alwala, Armand Joulin, and Ishan Misra. Imagebind: One embedding space to bind them all. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15180–15190, 2023.

- Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14953–14962, 2023.
- Vaishnavi Himakunthal, Andy Ouyang, Daniel Rose, Ryan He, Alex Mei, Yujie Lu, Chinmay Sonar, Michael Saxon, and William Yang Wang. Let’s think frame by frame: Evaluating video chain of thought with video infilling and prediction. *arXiv preprint arXiv:2305.13903*, 2023.
- Geoffrey E Hinton and Russ R Salakhutdinov. A better way to pretrain deep boltzmann machines. *Advances in Neural Information Processing Systems*, 25, 2012.
- Jordan Hoffmann et al. Training compute-optimal large language models, 2022.
- Shaohan Huang, Li Dong, Wenhui Wang, Yaru Hao, Saksham Singhal, Shuming Ma, Tengchao Lv, Lei Cui, Owais Khan Mohammed, Barun Patra, et al. Language is not all you need: Aligning perception with language models. *Advances in Neural Information Processing Systems*, 36, 2024.
- Yupan Huang, Zaiqiao Meng, Fangyu Liu, Yixuan Su, Nigel Collier, and Yutong Lu. Sparkles: Unlocking chats across multiple images for multimodal instruction-following models. *arXiv preprint arXiv:2308.16463*, 2023.
- Yuqi Huo, Manli Zhang, Guangzhen Liu, Haoyu Lu, Yizhao Gao, Guoxing Yang, Jingyuan Wen, Heng Zhang, Baogui Xu, Weihao Zheng, et al. Wenlan: Bridging vision and language by large-scale multi-modal pre-training. *arXiv preprint arXiv:2103.06561*, 2021.
- Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- COL Stephen A LaRocca, John J Morgan, and Sherri M Bellinger. On the path to 2x learning: Exploring the possibilities of advanced speech recognition. *Calico Journal*, pages 295–310, 1999.
- Hugo Laurençon et al. Obelics: An open web-scale filtered dataset of interleaved image-text documents, 2023.
- Gen Li, Nan Duan, Yuejian Fang, Ming Gong, and Dixin Jiang. Unicoder-vl: A universal encoder for vision and language by cross-modal pre-training. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 11336–11344, 2020a.
- Lei Li, Zhihui Xie, Mukai Li, Shunian Chen, Peiyi Wang, Liang Chen, Yazheng Yang, Benyou Wang, and Lingpeng Kong. Silkie: Preference distillation for large visual language models. *arXiv preprint arXiv:2312.10665*, 2023.
- Linjie Li, Yen-Chun Chen, Yu Cheng, Zhe Gan, Licheng Yu, and Jingjing Liu. Hero: Hierarchical encoder for video+ language omni-representation pre-training. *arXiv preprint arXiv:2005.00200*, 2020b.
- Ruilong Li, Shan Yang, David A Ross, and Angjoo Kanazawa. Ai choreographer: Music conditioned 3d dance generation with aist++. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13401–13412, 2021.
- Shengzhi Li and Nima Tajbakhsh. Scigraphqa: A large-scale synthetic multi-turn question-answering dataset for scientific graphs. *arXiv preprint arXiv:2308.03349*, 2023.

- Xiujun Li, Xi Yin, Chunyuan Li, Pengchuan Zhang, Xiaowei Hu, Lei Zhang, Lijuan Wang, Houdong Hu, Li Dong, Furu Wei, et al. Oscar: Object-semantics aligned pre-training for vision-language tasks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXXI 16*, pages 121–137. Springer, 2020c.
- Zheng Lian, Licai Sun, Mingyu Xu, Haiyang Sun, Ke Xu, Zhuofan Wen, Shun Chen, Bin Liu, and Jianhua Tao. Explainable multimodal emotion reasoning. *arXiv preprint arXiv:2306.15401*, 2023.
- Junyang Lin, An Yang, Yichang Zhang, Jie Liu, Jingren Zhou, and Hongxia Yang. Interbert: Vision-and-language interaction for multi-modal pretraining. *arXiv preprint arXiv:2003.13198*, 2020.
- Haohe Liu, Qiao Tian, Yi Yuan, Xubo Liu, Xinhao Mei, Qiuqiang Kong, Yuping Wang, Wenwu Wang, Yuxuan Wang, and Mark D Plumbley. Audioldm 2: Learning holistic audio generation with self-supervised pretraining. *arXiv preprint arXiv:2308.05734*, 2023.
- Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36, 2024.
- Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. *Advances in neural information processing systems*, 32, 2019.
- Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 689–696, 2011.
- Munan Ning, Bin Zhu, Yujia Xie, Bin Lin, Jiaxi Cui, Lu Yuan, Dongdong Chen, and Li Yuan. Video-bench: A comprehensive benchmark and toolkit for evaluating video-based large language models. *arXiv preprint arXiv:2311.16103*, 2023.
- Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.
- Shuhuai Ren, Linli Yao, Shicheng Li, Xu Sun, and Lu Hou. Timechat: A time-sensitive multimodal large language model for long video understanding. *arXiv preprint arXiv:2312.02051*, 2023.
- Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.
- Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5–9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.
- Shin’ichi Satoh and Takeo Kanade. Name-it: Association of face and name in video. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 368–373. IEEE, 1997.

- Bowen Shi, Wei-Ning Hsu, Kushal Lakhotia, and Abdelrahman Mohamed. Learning audio-visual speech representation by masked multimodal cluster prediction. *arXiv preprint arXiv:2201.02184*, 2022.
- Chen Sun, Austin Myers, Carl Vondrick, Kevin Murphy, and Cordelia Schmid. Videobert: A joint model for video and language representation learning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 7464–7473, 2019.
- Zhiqing Sun, Sheng Shen, Shengcao Cao, Haotian Liu, Chunyuan Li, Yikang Shen, Chuang Gan, Liang-Yan Gui, Yu-Xiong Wang, Yiming Yang, et al. Aligning large multimodal models with factually augmented rlhf. *arXiv preprint arXiv:2309.14525*, 2023.
- Yi Tay, Mostafa Dehghani, Vinh Q Tran, Xavier Garcia, Jason Wei, Xuezhi Wang, Hyung Won Chung, Dara Bahri, Tal Schuster, Steven Zheng, et al. UI2: Unifying language learning paradigms. In *The Eleventh International Conference on Learning Representations*, 2022.
- Shengbang Tong, Zhuang Liu, Yuexiang Zhai, Yi Ma, Yann LeCun, and Saining Xie. Eyes wide shut? exploring the visual shortcomings of multimodal llms. *arXiv preprint arXiv:2401.06209*, 2024.
- Hugo Touvron et al. Llama 2: Open foundation and fine-tuned chat models, 2023.
- Gokhan Tur, Andreas Stolcke, Lynn Voss, Stanley Peters, Dilek Hakkani-Tur, John Dowding, Benoit Favre, Raquel Fernández, Matthew Frampton, Mike Frandsen, et al. The calo meeting assistant system. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(6):1601–1611, 2010.
- Alessandro Vinciarelli, Maja Pantic, Hervé Bourlard, and Alex Pentland. Social signal processing: state-of-the-art and future perspectives of an emerging domain. In *Proceedings of the 16th ACM international conference on Multimedia*, pages 1061–1070, 2008.
- Xiao Wang, Guangyao Chen, Guangwu Qian, Pengcheng Gao, Xiao-Yong Wei, Yaowei Wang, Yonghong Tian, and Wen Gao. Large-scale multi-modal pre-trained models: A comprehensive survey. *Machine Intelligence Research*, pages 1–36, 2023.
- Zirui Wang, Jiahui Yu, Adams Wei Yu, Zihang Dai, Yulia Tsvetkov, and Yuan Cao. Simvlm: Simple visual language model pretraining with weak supervision. *arXiv preprint arXiv:2108.10904*, 2021.
- Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. Visual chatgpt: Talking, drawing and editing with visual foundation models. *arXiv preprint arXiv:2303.04671*, 2023a.
- Jiayang Wu, Wensheng Gan, Zefeng Chen, Shicheng Wan, and S Yu Philip. Multi-modal large language models: A survey. In *2023 IEEE International Conference on Big Data (BigData)*, pages 2247–2256. IEEE, 2023b.
- Shengqiong Wu, Hao Fei, Leigang Qu, Wei Ji, and Tat-Seng Chua. Next-gpt: Any-to-any multimodal llm. *arXiv preprint arXiv:2309.05519*, 2023c.
- Haiyang Xu, Ming Yan, Chenliang Li, Bin Bi, Songfang Huang, Wenming Xiao, and Fei Huang. E2e-vlp: end-to-end vision-language pre-training enhanced by visual learning. *arXiv preprint arXiv:2106.01804*, 2021.

- Peng Xu, Xiatian Zhu, and David A Clifton. Multimodal learning with transformers: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.
- Le Xue, Ning Yu, Shu Zhang, Junnan Li, Roberto Martín-Martín, Jiajun Wu, Caiming Xiong, Ran Xu, Juan Carlos Niebles, and Silvio Savarese. Ulip-2: Towards scalable multimodal pre-training for 3d understanding. *arXiv preprint arXiv:2305.08275*, 2023.
- Shukang Yin, Chaoyou Fu, Sirui Zhao, Ke Li, Xing Sun, Tong Xu, and Enhong Chen. A survey on multimodal large language models. *arXiv preprint arXiv:2306.13549*, 2023.
- Zhenfei Yin, Jiong Wang, Jianjian Cao, Zhelun Shi, Dingning Liu, Mukai Li, Xiaoshui Huang, Zhiyong Wang, Lu Sheng, Lei Bai, et al. Lamm: Language-assisted multi-modal instruction-tuning dataset, framework, and benchmark. *Advances in Neural Information Processing Systems*, 36, 2024.
- Quanzeng You, Hailin Jin, Zhaowen Wang, Chen Fang, and Jiebo Luo. Image captioning with semantic attention. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4651–4659, 2016.
- Peter Young, Alice Lai, Micah Hodosh, and Julia Hockenmaier. From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions. *Transactions of the Association for Computational Linguistics*, 2:67–78, 2014. doi: 10.1162/tacl_a_00166. URL <https://aclanthology.org/Q14-1006>.
- Tianyu Yu, Yuan Yao, Haoye Zhang, Taiwen He, Yifeng Han, Ganqu Cui, Jinyi Hu, Zhiyuan Liu, Hai-Tao Zheng, Maosong Sun, et al. Rlhf-v: Towards trustworthy mllms via behavior alignment from fine-grained correctional human feedback. *arXiv preprint arXiv:2312.00849*, 2023.
- Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414*, 2022.
- Xunlin Zhan, Yangxin Wu, Xiao Dong, Yunchao Wei, Minlong Lu, Yichi Zhang, Hang Xu, and Xiaodan Liang. Product1m: Towards weakly supervised instance-level product retrieval via cross-modal pretraining. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11782–11791, 2021.
- Duzhen Zhang, Yahan Yu, Chenxing Li, Jiahua Dong, Dan Su, Chenhui Chu, and Dong Yu. Mm-llms: Recent advances in multimodal large language models. *arXiv preprint arXiv:2401.13601*, 2024a.
- Ge Zhang, Xinrun Du, Bei Chen, Yiming Liang, Tongxu Luo, Tianyu Zheng, Kang Zhu, Yuyang Cheng, Chunpu Xu, Shuyue Guo, et al. Cmmmu: A chinese massive multi-discipline multimodal understanding benchmark. *arXiv preprint arXiv:2401.11944*, 2024b.
- Hang Zhang, Xin Li, and Lidong Bing. Video-llama: An instruction-tuned audio-visual language model for video understanding. *arXiv preprint arXiv:2306.02858*, 2023.

- Jiawei Zhang, Tianyu Pang, Chao Du, Yi Ren, Bo Li, and Min Lin. Benchmarking large multimodal models against common corruptions. *arXiv preprint arXiv:2401.11943*, 2024c.
- Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuhui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.
- Min Zhao, Fan Bao, Chongxuan Li, and Jun Zhu. Egsde: Unpaired image-to-image translation via energy-guided stochastic differential equations. *Advances in Neural Information Processing Systems*, 35:3609–3623, 2022.
- Zijia Zhao, Longteng Guo, Tongtian Yue, Sihan Chen, Shuai Shao, Xinxin Zhu, Zehuan Yuan, and Jing Liu. Chatbridge: Bridging modalities with large language model as a language catalyst. *arXiv preprint arXiv:2305.16103*, 2023.
- Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592*, 2023.



Chapter 10

LLMs: Evolution and New Frontiers

Abstract This concluding chapter provides an overview of the evolution of LLMs, emphasizing significant trends and developments. It explores the shift toward synthetic data to sustain model scaling, the expansion of context windows enhancing interpretative capabilities, the progression of training techniques that streamline efficiency and depth of knowledge transfer, and the transition from traditional Transformer architectures to alternative approaches such as state space models, which offer improved scalability and efficiency. Further discussion highlights the trends of smaller models, technology democratization, and domain-specific models, illustrating a movement toward more customized, accessible, and industry-specific AI solutions. Finally, the chapter delves into the frontiers of LLM technologies and their use in agent-based applications and search engines, which are increasingly replacing traditional technologies.

10.1 Introduction

The evolution of large language models encompasses significant architectural advancements, training techniques, and application trends. Innovations in model architecture and training efficiency have propelled LLMs to new heights, enabling them to handle more complex and extensive tasks. The shift toward synthetic data and larger context windows exemplifies the ongoing efforts to enhance model capabilities and performance. Emerging trends such as small language models, democratization through open-source initiatives, and domain-specific language models highlight the diverse applications and accessibility of LLMs. Additionally, new frontiers in LLM agents and enhanced search capabilities are setting new standards for complex task execution and information retrieval, further expanding the potential of LLMs in various fields.

10.2 LLM Evolution

10.2.1 Synthetic Data

As AI models increase in size and exhaust readily available high-quality internet data, there is a pressing need to shift toward synthetic data to sustain model development and achieve the necessary scaling. This trend assumes that increasing data quantities will enhance model performance, particularly for complex, rare tasks. While some argue that synthetic data may not advance state-of-the-art models because it mirrors existing data distributions, others believe that their diversity could improve models.

Anthropic leverages synthetic data extensively in its AI models, notably Claude 2.1, to enhance robustness by accurately refusing questions it cannot answer. Their approach, Constitutional AI (CAI), uses synthetic data in two primary ways: critiquing responses based on a set of ethical principles and generating pairwise preference data to train models using RLHF, a process known as RLAIF, as discussed in Chapter 5. CAI's dual approach—principled instruction correction and principle-following RLHF—has proven effective, allowing Anthropic to excel in synthetic data utilization and model training despite its relatively small team (Bai et al., 2022).

Models such as Alpaca and Vicuna utilize synthetic data for supervised fine-tuning of Llama models, enhancing performance within the 7-13B parameter range (Peng et al., 2023; Taori et al., 2023). Current trends include the use of methods such as Self-Instruct, where an LLM generates diverse instructional data from seed instructions. However, efforts are still in the initial stages to explore methods to enrich data diversity. In contrast, some still use low-quality internet prompts repurposed as training instructions by models such as GPT-4.

Synthetic preference datasets such as UltraFeedback collect user-generated prompts and model completions for RLHF training (Cui et al., 2023). Teknium1 has been actively employing synthetic instructions to train models such as OpenHermes on Mistral (Gallego, 2024). Meanwhile, Intel's recent LLM, Neural-Chat-v3-1, uses the DPO model to incorporate synthetic preferences. Berkeley's Starling model utilizes Nectar, a GPT-4-labeled ranking dataset. It aggregates prompts and scores from various models such as GPT-4, GPT-3.5-instruct, GPT-3.5-turbo, Mistral-7B-Instruct, and Llama-2-7B, resulting in a total of 3.8 million pairwise comparisons. Starling has achieved state-of-the-art performance on MT Bench 7b, although concerns about data contamination have been noted (Zhu et al., 2023a). Quality-Diversity through AI Feedback (QDAIF) employs evolutionary algorithms to boost data diversity (Bradley et al., 2023). Evol-instruct uses a rule-based system to generate diverse, high-quality instructions with feedback from GPT-4 (Xu et al., 2023).

10.2.2 Larger Context Windows

The context window of an LLM acts as a lens, providing perspective and functioning as short-term memory, and is useful for generation-based and conversation-based tasks. Larger context windows enhance an LLM's ability to learn from prompts by allowing for the input of more extensive and detailed examples, which results in more accurate and relevant responses. Additionally, a substantial context window enhances the model's ability to understand and connect information across distant parts of the text, which is especially beneficial for tasks requiring detailed document summarization, question-answering, and chatbot conversations, where larger context windows help maintain coherence over longer interactions.

The evolution of GPT models has shown substantial increases in context window size. Starting from a 2,000-token limit with GPT-3, the capacity expanded to 4096 tokens in the initial GPT-4 model. This was extended to 32768 tokens in the GPT-4-32k variant. The latest model, GPT-4 Turbo, now supports up to 128000 tokens, representing a 32x improvement over the initial GPT-4 and a 4x increase from GPT-4-32k, enhancing its ability to analyze and interpret extensive text data. Claude by Anthropic supports a 9,000 token context, and its successor, Claude 2, significantly extends this capacity to 100,000 tokens, allowing it to process documents up to 75,000 words in a single prompt. Meta AI's Llama family of models also supports more than 100,000 tokens.

Rotary Position Embeddings (RoPE) enhance Transformer models by embedding token positions directly into the model (Su et al., 2024). This technique involves rotating the position embeddings relative to each token's sequence position, facilitating consistent token position identification as the context window increases. Positional Skip-wise Training (PoSE) focuses on efficient context window extension for LLMs through a novel training technique that skips positions in a controlled manner, improving the handling of extended contexts in training and inference phases (Zhu et al., 2023b). LongRoPE extends LLM context windows to more than 2 million tokens, pushing the boundaries of current context management technologies and utilizing advanced rotational embeddings to handle extremely long inputs effectively (Ding et al., 2024).

Munkhdalai et al. (2024) introduce a method for scaling LLMs to handle extremely long inputs using a new attention technique called Infini-attention. Their approach integrates compressive memory with local and long-term linear attention mechanisms, demonstrating success in handling up to 1 million tokens for context retrieval and 500,000 tokens for book summarization tasks.

10.2.3 Training Speedups

This section discusses various techniques developed to enhance the efficiency of Transformer models. Despite their significant improvements in sequence modeling tasks, Transformer models suffer from high computational and memory costs due to

their quadratic complexity with respect to sequence length. Innovations such as parameter sharing, pruning, mixed-precision, and micro-batching have addressed these challenges, enabling more practical and widespread adoption of Transformer technology (Fournier et al., 2023).

Techniques such as *gradient checkpointing* involve selectively storing activations during the forward pass, which are then recomputed during the backward pass to save memory. This trade-off between memory and computational overhead allows scaling up the number of layers without linearly increasing memory use. The *parameter sharing* approach reduces the number of trainable parameters by reusing the same parameters across different parts of the network. Techniques such as *pruning* enhance model efficiency by removing less important weights after training. It can be applied in a structured manner, affecting components such as layers or attention heads, or unstructured, targeting individual weights. Pruning helps build smaller, faster models that are better optimized for modern computational hardware.

To increase the training speed and decrease the memory consumption of deep learning models, modern GPUs and TPUs utilize mixed-precision techniques. They perform computations in half-precision (16 bits) while maintaining a master copy of weights in single-precision for numerical stability. NVIDIA's Automatic Mixed-Precision simplifies integration with frameworks like TensorFlow, PyTorch, and MXNet. GPipe facilitates model scaling and performance improvement by allowing large models to be distributed across multiple processing units through an innovative micro-batching technique. This method splits mini-batches into smaller micro-batches, enabling parallel processing and reducing memory demands during forward and backward operations. This strategy allows for significant scaling in model size proportional to the number of accelerators used, enhancing training throughput without sacrificing computational efficiency.

10.2.4 Multi-Token Generation

Traditional LLMs using conventional next-token prediction are resource intensive and often fail to capture long-term dependencies effectively. Meta's research presents a novel approach to training LLMs through multi-token prediction. This method diverges from traditional next-token prediction by forecasting several future tokens simultaneously, enhancing both efficiency and performance (Gloeckle et al., 2024). This technique triples inference speed and increases sample efficiency, particularly in larger models and coding tasks. Meta's 13-billion-parameter model demonstrated a 12% and 17% improvement in problem-solving capabilities on the HumanEval and MBPP benchmarks, respectively.

The approach relies on a shared model trunk that processes input sequences into a latent representation, with multiple output heads designed to predict different future tokens independently. This structure allows for parallel token predictions without increasing computational demands during training. During inference, the model uses

the trained output heads to generate multiple tokens simultaneously, further speeding up the process and reducing latency.

10.2.5 Knowledge Distillation

Knowledge distillation (KD) involves transferring insights from a large, sophisticated model (the teacher) to a smaller, more efficient model (the student). Given the significant computational demands and resource constraints of large-scale models, this process has become crucial for practical deployment. With the rise of LLMs such as GPT-4 and Gemini, the focus of knowledge distillation has evolved from simply reducing model size or mimicking outputs to a more intricate transfer of deep-seated knowledge.

This shift is primarily due to the rich and nuanced understanding these LLMs have developed, which cannot be fully captured by traditional compression methods such as pruning or quantization. Instead, the contemporary approach in LLM-based knowledge distillation leverages carefully crafted prompts to extract specific knowledge or capabilities. These prompts tap into the LLM's expertise across various domains, including natural language processing, reasoning, and problem solving. This strategy allows for more targeted and dynamic knowledge transfer, focusing on particular skills or areas of interest.

Moreover, the current phase of knowledge distillation extends beyond simple output replication. It aims to transfer more abstract qualities such as reasoning patterns, preference alignment, and ethical values. Modern techniques involve teaching the student model to emulate the teacher's thought processes and decision-making patterns. This is often achieved through chain-of-thought prompting, which trains the student model to understand and replicate the teacher's reasoning process, enhancing cognitive capabilities across complex tasks.

In their survey, [Xu et al. \(2024\)](#) categorize the exploration of KD into three primary facets: KD algorithms, skill distillation, and verticalization distillation, each encompassing a variety of methodologies and subtopics.

KD algorithms focus on the foundational techniques of knowledge distillation, detailing how knowledge is constructed from teacher models and integrated into student models. It covers labeling, expansion, curation, feature understanding, feedback mechanisms, and self-knowledge generation. Additionally, it discusses various learning approaches, including supervised fine-tuning, divergence minimization, reinforcement learning, and rank optimization to facilitate effective knowledge transfer, enabling open-source models to match or exceed the capabilities of proprietary models.

Skill distillation addresses enhancing specific competencies through KD, including context following, instruction adherence, retrieval-augmented generation, alignment in thinking patterns, persona/preference modeling, and value alignment. It also explores NLP task specialization, such as natural language understanding and generation, information retrieval, recommendation systems, text generation evalua-

tion, and code generation. Furthermore, this segment investigates how KD improves LLMs' ability to handle multi-modal inputs, enhancing their functionality across different contexts.

Verticalization distillation evaluates the application of KD across specialized fields such as law, healthcare, finance, and science, illustrating how KD adapts LLMs to specific industry needs. This highlights the transformative impact of KD techniques on domain-specific AI solutions, and it underscores their versatility and effectiveness in meeting the varied demands of different industries within the AI and machine learning ecosystem.

10.2.6 Post-Attention Architectures

State space models (SSMs) have emerged as a focal point in the evolution of deep learning technologies, particularly in addressing the limitations of traditional neural network architectures such as CNNs, RNNs, GNNs, and even Transformers. These models represent dynamic systems through state variables initially drawn from control theory and computational neuroscience. The Mamba model enhances computational efficiency, achieving 5x faster inference and linear scalability compared to Transformers. It features input-adaptive SSMs for better content reasoning, significantly outperforming same-sized Transformers and matching those twice its size in language, audio, and genomics tasks ([Gu and Dao, 2023](#)).

In language modeling, researchers have explored applications such as the Gated State Space (GSS) method for long-range language modeling, which offers substantial speed improvements and reduced computational overhead ([Mehta et al., 2022](#)). The Structured State Space sequence model (S4) introduces a new, more efficient parameterization for state space models, achieving significant computational savings and strong performance across benchmarks. S4 matches or surpasses previous models in tasks such as sequential CIFAR-10 and image/language modeling, performs generation 60x faster, and sets new records in the Long Range Arena benchmark, effectively handling sequences up to 16,000 in length ([Gu et al., 2021](#)).

10.3 LLM Trends

10.3.1 Small Language Models

LLMs have been central to advancements in numerous fields, yet the substantial computational resources required for these models have generally limited their use to well-resourced organizations. Increasingly, researchers are working to replicate the capabilities of large models in much smaller packages. *Small Language Models* (SLMs) are scaled-down versions of LLMs. They possess far fewer parameters—

ranging from millions to billions—than the hundreds of billions or trillions found in LLMs. The smaller size of SLMs offers several benefits:

1. **Efficiency:** SLMs consume less power and need less memory, making them suitable for deployment on smaller devices as in the case of edge computing. This capability facilitates practical applications, such as on-device chatbots and personal mobile assistants, that can operate directly from a user's device.
2. **Accessibility:** The reduced resource demands of SLMs make them more attainable for a wider spectrum of developers and organizations. This broad accessibility helps democratize artificial intelligence, enabling even small teams and independent researchers to leverage the capabilities of language models without the need for substantial infrastructure.
3. **Customization:** SLMs are simpler to adapt to specific domains and tasks, making it possible to develop specialized models that are precisely tailored to specific needs. This customization can lead to improved performance and greater accuracy in niche applications.
4. **Enhanced Security and Privacy:** A notable advantage of SLMs is their potential for improved security and privacy. Their manageable size allows for deployment on-premises or within private cloud environments, which minimizes the risk of data breaches. This feature is particularly valuable in industries that handle sensitive information, such as finance and healthcare, where maintaining control over data is crucial.

Here are some popular small language models currently making waves in the industry, although this is by no means an exhaustive list.

1. **Llama:** The Llama-3 model is an open-access, 2.7 billion-parameter tool proficient in handling nuanced language tasks, translation, and dialog generation ([Touvron et al., 2023](#)).
2. **Phi-2:** Developed by Microsoft, this model utilizes 2.7 billion parameters to achieve exceptional performance in mathematical reasoning, common sense evaluations, and logical tasks ([Jawaherip et al., 2023](#)). Phi-2 employs synthetic data for training, competing with, and sometimes surpassing, models ten times its size in tasks such as reading comprehension and text summarization.
3. **Mistral 7B:** A robust model with 7.3 billion parameters, Mistral 7B surpasses the performance of previous Llama models and approaches the capabilities of specialized code models ([Jiang et al., 2023a](#)). It integrates advanced techniques like grouped-query attention for faster processing and sliding window attention to manage longer text sequences efficiently.
4. **Gemma 2B and Gemma 7B:** These variants, both pre-trained and instruction-tuned, excel in text-based tasks, outperforming comparable open models in 11 out of 18 evaluations ([Team et al., 2024](#)). The development of Gemma models also emphasizes safety and responsibility, ensuring their reliability in practical applications.
5. **Vicuna-13B:** This open-source conversational model, based on the Llama-13B framework, is enhanced by fine-tuning on user-shared conversations. Initial evaluations, with GPT-4 as the benchmark, indicate that Vicuna-13B delivers quality

surpassing 90% of that seen in models such as ChatGPT and Google Bard. It outperforms other models such as Llama and Alpaca in the majority of tests (Peng et al., 2023).

10.3.2 Democratization

Recent months have seen transformative changes in LLMs, fueled largely by the expanding influence of the open-source community. The essence of open source—marked by its commitment to collaborative development, transparency, and free access—has profoundly impacted the progress of LLMs. LLMs’ open-source initiatives encompass various resources, including pre-training data, models and architectures, instruction-tuning datasets, alignment-tuning datasets, and even hardware.

Petals addresses the challenges of researchers who lack access to the high-end hardware necessary for leveraging LLMs such as BLOOM-176B and OPT-175B (Borzunov et al., 2022). Petals enables collaborative inference and fine-tuning of these large models by pooling resources from those who want to share their GPU cycles. It provides a solution faster than RAM offloading for interactive applications, with the ability to run inference on consumer GPUs at approximately one step per second.

Hugging Face’s ZeroGPU initiative uses Nvidia A100 GPUs to provide shared, on-demand GPU access via their Spaces app, aiming to democratize access to computational resources and reduce costs for smaller organizations.

Various datasets related to pre-training, instruction tuning, alignment tuning, and more, are continuously made available to the community. Contributors regularly release open-source datasets online, and initiatives such as [LLMDataHub](#) and [Open LLM Datasets](#) are instrumental in centralizing these resources. This central repository simplifies access and utilization for developers and researchers engaged in LLM development.

OpenLLM enables developers to operate any open-source LLM, such as Llama-2 or Mistral, through OpenAI-compatible API endpoints both locally and in the cloud (Pham et al., 2023). This platform supports a wide range of LLMs, facilitates seamless API transitions for applications, and offers optimized serving for high-performance and simplified cloud deployment using BentoML.

While open-source LLMs are discussed extensively in Chapter 8, readers seeking the latest developments can refer to the Hugging Face leaderboard at [HuggingFace](#) for ongoing updates and rankings.

10.3.3 Domain-Specific Language Models

Domain-Specific Language Models (DSLMs) address the limitations of general purpose models by specializing in particular industries or fields. These models are finely

tuned with domain-specific data and terminology, making them ideal for complex and regulated environments where precision is essential. This targeted approach ensures that DSLMs provide accurate and contextually appropriate responses, reducing the likelihood of errors and “hallucinations” that general-purpose models may produce when faced with specialized content.

DSLMs are particularly beneficial for professionals such as lawyers, medical providers, and financial analysts who rely on precise and reliable information. By focusing on a narrower scope and incorporating industry-specific jargon, these models are designed to effectively handle the specific workflows and processes of their designated fields. As enterprises increasingly recognize the value of tailored AI solutions, it is projected that by 2027, more than half of the generative AI models employed by businesses will be domain specific, serving distinct industrial or functional needs.

In the legal field, SaulLM-7B, developed by Equall.ai, is a prime example of employing legal-specific pre-training and fine-tuning to address the complexities of legal language, significantly improving task performance in legal applications (Colombo et al., 2024). In healthcare, models such as GatorTron, Codex-Med, Galactica, and Flan-PaLM have been developed to address the nuances of medical data and clinical information, pushing the boundaries of what AI can achieve in diagnosing and managing patient care (Singhal et al., 2023; Taylor et al., 2022; Yang et al., 2022, 2023). Similarly, the finance sector has seen advancements with models such as BloombergGPT and FinBERT, trained on extensive financial data to enhance tasks such as risk management and financial analysis (Liu et al., 2021; Wu et al., 2023).

10.4 New Frontiers

10.4.1 LLM Agents

LLM agents represent a framework for leveraging LLM capabilities to accomplish highly complex and sophisticated tasks. These agents are modular programs that can read in a user request, reason through the steps required to complete it, create and allocate sub-tasks to various modules, and synthesize the results into a satisfactory output. The key feature of this framework is a blend of traditional computing logic and system tools with LLMs prompted to use them intelligently.

For example, consider the task of conducting market research on different available headphones and choosing a few top options based on pricing, features, and user reviews. ChatGPT alone cannot accomplish this goal, as it can only access information based on data seen during pre-training. An LLM agent, on the other hand, can read this request, decide it needs to search the internet for information, construct the relevant search queries, execute the searches, download the resulting pages, process the information, and return a series of suggestions.

Although there are different flavors, agents generally have a few common modules, graphically illustrated in Fig. 10.1 and listed here:

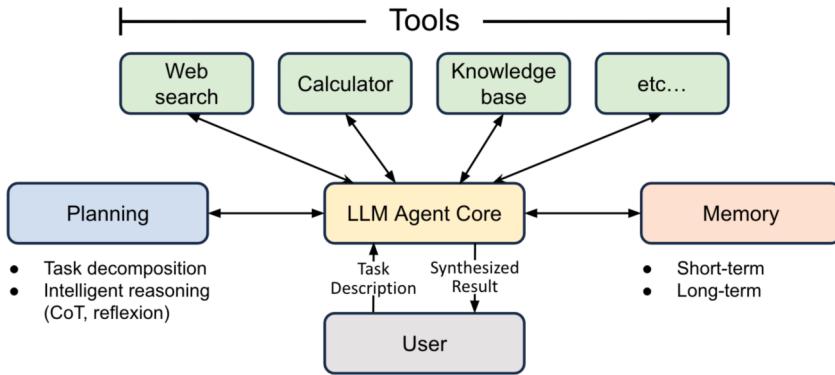


Fig. 10.1: Visualization of the high-level modules in an LLM agent. The core module takes in a user task, accesses relevant information from the memory module, allocates sub-task decomposition to the planning module, and uses the available tools to accomplish the sub-tasks. Finally, the core synthesizes the information to respond to the task and returns the result to the user.

- **Core:** This is the central module that defines the characteristics of the agent, ingests user instructions, and allocates tasks to other modules. This is accomplished by a lengthy and highly specific prompt template that instructs the LLM on how to execute these tasks.
- **Planning:** This module determines the series of steps needed to accomplish the overall task. Using reasoning approaches like Chain-of-Thought (Sect. 4.5.4) and iterative improvement algorithms like Reflexion (Shinn et al., 2023), it develops and refines a plan consisting of a sequence of sub-tasks which can be executed by the various functions of the agent.
- **Tools:** A series of tools available to the agent that go beyond the standard capabilities of LLMs. The possibilities for this section are endless but may consist of web search commands, code compilers, calculators, and API calls of any sort.
- **Knowledge:** A knowledge base that can be queried by the agent if necessary. This could be a RAG system similar to those described in Chapter 7, or a structured database that can be queried through calls (e.g. SQL) that can be generated by the language model.
- **Memory:** This module contains a record of information derived from interactions between the user and agent, which can be reviewed if deemed necessary for a given task. Sometimes, it is divided between short-term memory, which has granular details of all interactions of the current session, and long-term memory, which is a more curated list of relevant information learned over the course of many interactions.

These agents can be carefully crafted for specific tasks such as scientific writing (Ziemer et al., 2024), playing video games (Wang et al., 2023), manipulating robots (Michael Ahn, 2022), and more. Researchers have also developed generalist agents that will attempt any task given by the user. An early example is *AutoGPT*¹, which closely follows the layout in Fig. 10.1—it takes in a user command, uses crafted prompt templates in the core to establish a workflow, engages in chain-of-thought reasoning and self-criticism to generate a plan, and leverages memory modules and tools to accomplish the goal. Notably, this model accepts no user feedback on its plan, autonomously attempting the entire task-solving process. Another popular agent base is *BabyAGI*², which is similar in big picture layout to AutoGPT but iterates on its plan after every task instead of executing a decided-on string of tasks.

! Practical Tips

Many agents are built on the backs of open-source packages designed to handle complex LLM frameworks. With popular examples such as LangChain and LlamaIndex (Sect. 8.6.1), these packages implement many functions for calling LLMs, integrations for common tools, a suite of prompt templates for many use cases, and web-hosting features. BabyAGI, in particular, uses LangChain integrations in its workflow, and the symbiosis goes both directions—LangChain has integrated AutoGPT and BabyAGI into their product, allowing agent systems seamless access to the different LLMs, vector indices, and tools already implemented by LangChain.

10.4.2 LLM-Enhanced Search

Another frontier of LLM applications is enhancing search capabilities in different contexts. LLM-powered search has the potential to improve web and document search algorithms in a few different ways, which fall into a few broad categories:

- Improving top search results
- Query engineering
- Reasoning from search results

Improving top search results

Traditional search methods rely on keyword matches against a search query, but there is a major limitation to this approach – it cannot return text on a subject similar to the search query but without the exact keywords. Word vectors have expanded the search

¹ <https://github.com/Significant-Gravitas/AutoGPT>

² <https://github.com/yoheinakajima/babyagi>

range of individual terms, producing matches on semantically similar terms. Sentence embeddings with language models go a step further, matching longer phrases with semantic similarity. As embedding models improve, there is promise that the scope of search retrieval will sharpen. This is accomplished with reranking (Sect. 7.4.3.1). Reranking involves collecting several top results using an efficient search algorithm and dynamically reranking them with a slower but more powerful ranking algorithm. In the context of web search, a traditional keyword-based internet search may return hundreds of thousands of matches with a less sophisticated top ranking, and the top- k results can be reranked based on semantic similarity between the query and the contents of the web page. Such techniques have been the industry standard in web searching since the advent of Transformer models but have gained greater capabilities in the era of powerful LLM-based agents that can precisely parse the meaning of human language.

Improving queries

LLMs also offer the possibility of improving search querying. We have discussed certain of these approaches in the context of RAG (Chapter 7). These include:

- Query re-writing: Training a model to take in a human search query and refresh it into a form more likely to return relevant searches.
- Query-to-document: Generating synthetic documents of a form similar to the desired search result to create a closer match.
- Query-to-SQL: Using an LLM to convert human language queries against a structured database into a code-based call (see Listing 7.4.1.1 in Sect. 7.4.1.1).

Forward-Looking Active REtrieval augmented generation (FLARE) is an additional technique for extracting more relevant information for a search (Jiang et al., 2023b). In FLARE, an LLM generates successive queries off the back of the original query, imagining new contexts with potentially relevant information, executing those searches and incorporating the new results.

Reasoning from search results

Another way that LLMs revolutionize search can be understood by analogy to retrieval-augmented generation (Chapter 7). Semantic searching of documents is already a step in the RAG pipeline, and instead of using the resulting documents to answer the original query, a RAG-powered search engine will simply return the most semantically similar documents. This technique is viable for single-document or web domain searches and represents an improvement over traditional keyword-based searches, which struggle to detect similar, but not identical, subjects to the query.

The RAG-style search-and-describe approach is also useful for web searches. Instead of simply returning top hits to a query, an LLM-powered search engine can return top semantically similar matches from pre-indexed webpages, extract the information from the pages, and use them as context to directly answer the query. The company You.com created a web portal with similar functionality. Using a web-based query page similar to other chatbots, You.com is deeply integrated with a

Google search. Upon receiving a query, You.com searches the internet for information relevant to the question, processes the details of the pages, and uses the context to provide an answer to the query. With access to new and timely information published online, this chatbot style has become a useful source for RAG-style searches with the internet as the database. Powerful applications include synthesizing information from recent news events, market research, and general QA.

The challenges remain the same as those of other chatbot clients, namely hallucination and the reliability of source material. As a cautionary tale, recent experiments by Google in AI-based news aggregation and summarization have created quite inaccurate responses to basic questions, in some cases instructing users to eat rocks or use glue as a pizza topping (Grant, 2024). These responses resulted from satirical comments or web pages being used as context, and they highlight the danger of an uncurated dataset, such as the results of an internet search, as a ground-truth knowledge base for LLM QA. The potential harm and reputational damage resulting from inaccurate LLM search-and-reply is a serious issue and demands care when developing such products.

10.5 Closing Remarks

In this final chapter, we have attempted to provide the reader with insights into where we see the innovation frontiers for LLM innovation, or perhaps more appropriately, multi-modal LLM innovation, which as we saw in Chapter 9, are fast becoming the new paradigm in language modeling. While our views on these innovation frontiers are informed by the significant literature review effort we have undertaken to write this book itself, it is truly anyone's guess what the future may hold. Human capacity for creativity and invention remains an unpredictable quantity in this equation. After all, who would have predicted the astounding capabilities of ChatGPT when the seminal paper, *Attention Is All You Need* was published by Vaswani et al.?

And so it is true today that the wildcard of human innovation leaves us trepidatious in stating too concretely where we see the field going in the future. However, the fact that *human innovation* is still a factor in this question at all is indicative of where LLMs and their capabilities lie relative to human intelligence. There is still lots more to learn, lots more to understand, lots more to fail at, and many more successes for us to experience on this endeavour to create machines that can complete tasks as or more competently than ourselves, collectively and individually.

One hope we have for the content of this book, and the research it curates, is that it will be a valuable resource for those lucky individuals with the skills, interest, intelligence, or opportunity to contribute to this most exciting chapter in human technological evolution. More importantly, however, is our hope that those individuals push our technological capabilities forward responsibly, ethically, and with the utmost deference to human dignity. All technology is a double-edged sword, but none more so than technology that has the potential to exceed human competency in such a broad range of tasks.

With that said, we hope the reader enjoyed the journey that we have navigated, and we hope it contributes to your understanding and mastery of large language model research and utilization. As the field of AI marches on, and the content of this book requires a refresh, we look forward to future editions, and we hope you do too.

References

- Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKittrick, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.
- Alexander Borzunov et al. Petals: Collaborative inference and fine-tuning of large models. *arXiv preprint arXiv:2209.01188*, 2022. URL <https://arxiv.org/abs/2209.01188>.
- Herbie Bradley, Andrew Dai, Hannah Teufel, Jenny Zhang, Koen Oostermeijer, Marco Bellagente, Jeff Clune, Kenneth Stanley, Grégory Schott, and Joel Lehman. Quality-diversity through ai feedback. *arXiv preprint arXiv:2310.13032*, 2023.
- Pierre Colombo et al. Saullm-7b: A pioneering large language model for law. *arXiv preprint arXiv:2403.03883*, 2024.
- Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Wei Zhu, Yuan Ni, Guotong Xie, Zhiyuan Liu, and Maosong Sun. Ultrafeedback: Boosting language models with high-quality feedback. *arXiv preprint arXiv:2310.01377*, 2023.
- Yiran Ding, Li Lyra Zhang, Chengruidong Zhang, Yuanyuan Xu, Ning Shang, Jiahang Xu, Fan Yang, and Mao Yang. Longrope: Extending llm context window beyond 2 million tokens. *arXiv preprint arXiv:2402.13753*, 2024.
- Quentin Fournier, Gaétan Marceau Caron, and Daniel Aloise. A practical survey on faster and lighter transformers. *ACM Computing Surveys*, 55(14s):1–40, 2023.
- Victor Gallego. Configurable safety tuning of language models with synthetic preference data. *arXiv preprint arXiv:2404.00495*, 2024.
- Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. *arXiv preprint arXiv:2404.19737*, 2024.
- Nico Grant. Google’s a.i. search errors cause a furor online. *The New York Times*, 2024. URL <https://www.nytimes.com/2024/05/24/technology/google-ai-overview-search.html>.
- Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.
- Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.
- Mojan Javaheripi et al. Phi-2: The surprising power of small language models. *Microsoft Research Blog*, 2023.
- Albert Q Jiang et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023a.

- Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation, 2023b.
- Zhuang Liu et al. Finbert: A pre-trained financial language representation model for financial text mining. In *Proceedings of the twenty-ninth international conference on international joint conferences on artificial intelligence*, pages 4513–4519, 2021.
- Harsh Mehta, Ankit Gupta, Ashok Cutkosky, and Behnam Neyshabur. Long range language modeling via gated state spaces. *arXiv preprint arXiv:2206.13947*, 2022.
- Noah Brown et al. Michael Ahn, Anthony Brohan. Do as i can, not as i say: Grounding language in robotic affordances, 2022.
- Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. Leave no context behind: Efficient infinite context transformers with infini-attention. *arXiv preprint arXiv:2404.07143*, 2024.
- Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277*, 2023.
- Aaron Pham et al. OpenLLM: Operating LLMs in production, June 2023. URL <https://github.com/bentoml/OpenLLM>.
- Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023.
- Karan Singhal et al. Large language models encode clinical knowledge. *Nature*, 620(7972):172–180, 2023.
- Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.
- Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. Alpaca: A strong, replicable instruction-following model. *Stanford Center for Research on Foundation Models*. <https://crfm.stanford.edu/2023/03/13/alpaca.html>, 3(6):7, 2023.
- Ross Taylor et al. Galactica: A large language model for science. *arXiv preprint arXiv:2211.09085*, 2022.
- Gemma Team, Mesnard, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.
- Hugo Touvron et al. Llama 2: Open foundation and fine-tuned chat models, 2023.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fb053c1c4a845aa-Paper.pdf.
- Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents, 2023.

- Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabrowski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. Bloomberggpt: A large language model for finance, 2023.
- Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*, 2023.
- Xiaohan Xu, Ming Li, Chongyang Tao, Tao Shen, Reynold Cheng, Jinyang Li, Can Xu, Dacheng Tao, and Tianyi Zhou. A survey on knowledge distillation of large language models. *arXiv preprint arXiv:2402.13116*, 2024.
- Xi Yang et al. A large language model for electronic health records. *NPJ digital medicine*, 5(1):194, 2022.
- Zhichao Yang et al. Surpassing gpt-4 medical coding with a two-stage approach. *arXiv preprint arXiv:2311.13735*, 2023.
- Banghua Zhu, Evan Frick, Tianhao Wu, Hanlin Zhu, and Jiantao Jiao. Starling-7b: Improving Ilm helpfulness & harmlessness with rlaif, 2023a.
- Dawei Zhu, Nan Yang, Liang Wang, Yifan Song, Wenhao Wu, Furu Wei, and Sujian Li. Pose: Efficient context window extension of llms via positional skip-wise training. *arXiv preprint arXiv:2309.10400*, 2023b.
- Caleb Ziems, William Held, Omar Shaikh, Jiaao Chen, Zhehao Zhang, and Diyi Yang. Can large language models transform computational social science?, 2024.

Appendix A

Deep Learning Basics

A.1 Basic Structure of Neural Networks

Neural networks, inspired by the human brain, consist of interconnected nodes or “neurons” that process information in layers.

$$y = f(\mathbf{w} \cdot \mathbf{x} + b) \quad (\text{A.1})$$

where y is the output, \mathbf{w} is the weight vector, \mathbf{x} is the input vector, b is the bias, and f is the activation function.

- **Neurons:** The fundamental processing units of a neural network.
- **Weights:** Values that determine the strength of connections between neurons.
- **Biases:** Offset values added to the weighted input before passing through an activation function.
- **Activation Functions:** Functions like the sigmoid ($\sigma(x) = \frac{1}{1+e^{-x}}$), tanh, and ReLU ($f(x) = \max(0, x)$) that introduce nonlinearity to the network.

A.2 Perceptron

Perceptrons are a type of linear classifier, which means they make their classifications based on a linear predictor function combining a set of weights with the feature vector. The algorithm uses these weights to make decisions by applying a sign function, thus distinguishing between two classes. This can be expressed mathematically as follows:

$$h(\mathbf{x}) = \text{sign} \left(\sum_{i=0}^d w_i x_i \right) \quad (\text{A.2})$$

where $h(\mathbf{x})$ represents the hypothesis or prediction function, \mathbf{x} is the input feature vector, w_i are the weights, and d is the dimensionality of the input vector.

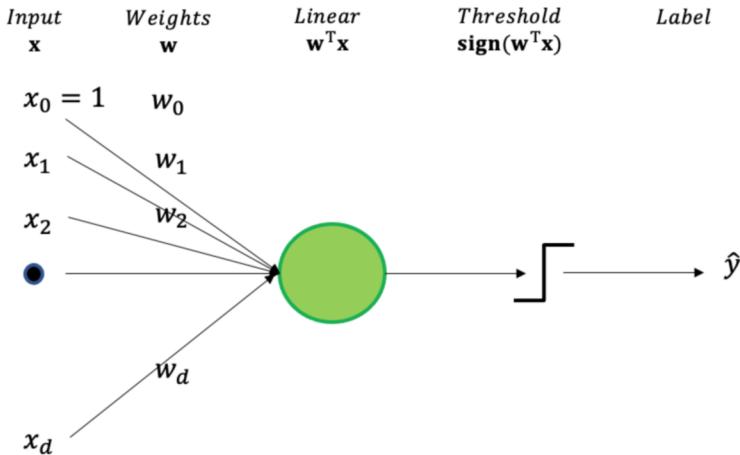


Fig. A.1: Perceptron model

The training of a perceptron occurs on linearly separable datasets. The process involves initializing weights and iteratively adjusting them when misclassifications occur. The key steps in the training algorithm are as follows:

1. Initialize the weights to zero or small random values.
2. For each example in the training set, perform the following steps:
 - a. Calculate the output using the current weights.
 - b. If the output is incorrect, update the weights:

$$w_i \leftarrow w_i + \eta(y - \hat{y})x_i$$

where η is the learning rate, y is the true label, and \hat{y} is the predicted label.

3. Repeat until the weights converge or after a predetermined number of iterations (to handle non-separable cases via the pocket algorithm).

This algorithm attempts to find a hyperplane that separates the two classes. If the dataset is not linearly separable, modifications such as the pocket algorithm ensure that the best hyperplane during the iterations, which achieves the lowest error, is retained.

A.3 Multilayer Perceptron

Multilayer Perceptrons (MLPs) extend the perceptron model by adding one or more layers of neurons, each consisting of perceptrons connected in a feed-forward man-

ner. MLPs replace the simple step function of perceptrons with differentiable nonlinear activation functions, enabling them to capture complex patterns and relationships in data.

A.3.1 Structure and Function of MLPs

An MLP consists of an input layer, one or more hidden layers, and an output layer. Each layer is fully connected to the next layer, meaning that every neuron in one layer connects to every neuron in the subsequent layer. The output of each neuron is computed as:

$$\mathbf{h}^{(l)} = g(\mathbf{W}^{(l)} \mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}) \quad (\text{A.3})$$

where $\mathbf{h}^{(l-1)}$ represents the output from the previous layer, $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the weight matrix and bias vector of layer l , and g is a nonlinear activation function such as sigmoid or ReLU.

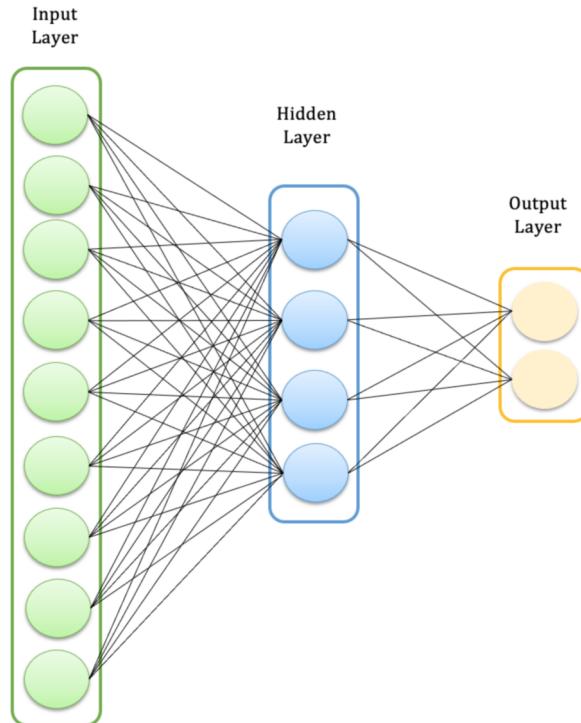


Fig. A.2: Multilayer Perceptron with an input layer, one hidden layer, and an output layer.

A.3.2 Training MLPs

Training an MLP involves optimizing the weights and biases across all layers to minimize the error in predictions. This is typically achieved through the following steps:

1. **Forward Propagation:** Calculate the output for a given input by processing it through each layer of the network:

$$\mathbf{h}^{(l)} = g(\mathbf{W}^{(l)} \mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}) \quad (\text{A.4})$$

2. **Error Computation:** After obtaining the final output $\hat{\mathbf{y}}$, compute the loss E using a loss function such as the mean squared error (MSE):

$$E(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{2} \sum_i (\hat{y}_i - y_i)^2 \quad (\text{A.5})$$

3. **Backpropagation:** Calculate gradients of the loss with respect to each weight and bias by applying the chain rule:

$$\frac{\partial E}{\partial \mathbf{W}^{(l)}} = \frac{\partial E}{\partial \mathbf{h}^{(l)}} \cdot \frac{\partial \mathbf{h}^{(l)}}{\partial \mathbf{W}^{(l)}} \quad (\text{A.6})$$

4. **Weight Update:** Adjust the weights and biases using an optimization technique, such as stochastic gradient descent (SGD):

$$\mathbf{W}^{(l)} = \mathbf{W}^{(l)} - \eta \frac{\partial E}{\partial \mathbf{W}^{(l)}} \quad (\text{A.7})$$

$$\mathbf{b}^{(l)} = \mathbf{b}^{(l)} - \eta \frac{\partial E}{\partial \mathbf{b}^{(l)}} \quad (\text{A.8})$$

where η is the learning rate.

These steps are iterated upon for multiple epochs over the training data until the network's performance no longer improves significantly. Each iteration of this process is designed to incrementally adjust the parameters of the network to reduce prediction errors, thus refining the model accuracy over time.

A.4 Deep Learning

Deep learning is a subset of machine learning that involves networks capable of learning from data through layers composed of multiple levels of abstraction. Although often equated with neural networks, deep learning specifically refers to networks with multiple layers that help capture complex data patterns more effectively than networks with fewer layers.

The distinction between “deep” and “shallow” neural networks can vary, with deep networks typically having more layers that enable them to learn more complex functions and hierarchical abstractions of data. These networks learn using back-propagation and gradient-based optimization methods, making them highly effective in various domains, including image and speech recognition, natural language processing, and more.

A.4.1 Key Components of Deep Neural Networks

In deep neural networks, several factors influence the design and effectiveness of the model:

- **Activation Functions:** These functions introduce nonlinearities into the network, which are crucial for learning complex patterns. Common choices include ReLU, sigmoid, and tanh.
- **Loss Functions:** Also known as error functions, loss functions measure how well the network’s predictions conform to the actual data. Examples include mean squared error and cross-entropy.
- **Optimization Methods:** Techniques such as stochastic gradient descent, Adam, and RMSprop are used to update the weights of the network to minimize the loss function.
- **Regularization Techniques:** Methods such as dropout, L2 regularization, and batch normalization help to prevent overfitting and improve the generalizability of the network.

This section will explore these components in detail, emphasizing their role in architecting and training deep neural networks to tackle sophisticated tasks more effectively.

A.4.2 Activation Functions

Activation functions introduce nonlinearities essential for deep neural networks to learn complex patterns. These functions are applied at each neuron and significantly impact the network’s ability to converge and the speed of convergence.

A.4.2.1 Sigmoid

The sigmoid function is one of the most traditional activation functions, and is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (\text{A.9})$$

It maps input values to the (0,1) range, providing a smooth gradient necessary for backpropagation. However, the sigmoid can lead to vanishing gradients when outputs approach the function's bounds.

A.4.2.2 Tanh

The hyperbolic tangent function, or tanh, modifies the sigmoid function by scaling its output to a range between -1 and 1:

$$f(x) = \tanh(x) = 2\sigma(2x) - 1 \quad (\text{A.10})$$

This zero-centered property makes it generally preferable to the sigmoid function in the hidden layers of a neural network.

A.4.2.3 ReLU

The Rectified Linear Unit (ReLU) has become the default activation function for many types of neural networks due to its computational simplicity and efficiency:

$$f(x) = \max(0, x) \quad (\text{A.11})$$

ReLU facilitates faster and more effective training by allowing the gradient to pass through unchanged when the input is positive. However, it can lead to “dead neurons”, where some neurons stop learning entirely due to negative input values.

A.4.2.4 Leaky ReLU and Variants

To address the dying neuron problem of ReLU, Leaky ReLU allows a small, nonzero gradient when the unit is inactive:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases} \quad (\text{A.12})$$

where α is a small coefficient. Variants like Parametric ReLU (PReLU) and Exponential Linear Unit (ELU) further adapt this concept by adjusting α during training or changing the negative part of the function to an exponential decay.

A.4.2.5 Softmax

For classification tasks, the softmax function is often applied in the output layer to normalize the output of the last linear layer to a probability distribution over the predicted output classes:

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \quad (\text{A.13})$$

where x_i are the inputs to the softmax function from the last network layer.

A.4.3 Loss Functions

Loss functions, also known as cost functions or objective functions, quantify how well a model's predictions match the actual data. By minimizing the loss function, we optimize the model's parameters.

A.4.3.1 Mean Squared (L_2) Error

Mean Squared Error (MSE) is used to compute the squared discrepancies between predictions and targets. It highlights large errors due to the squaring of the error terms, which makes it sensitive to outliers. Commonly applied in regression, it can be adapted for binary classification:

$$E(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (\text{A.14})$$

A.4.3.2 Mean Absolute (L_1) Error

Mean Absolute Error (MAE) measures the absolute differences between predicted values and actual targets, making it robust against outliers by not squaring the differences:

$$E(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i| \quad (\text{A.15})$$

A.4.3.3 Negative Log Likelihood

Negative Log Likelihood (NLL) is primarily utilized for multi-class classification problems. It calculates a log probability cost that penalizes the divergence of the predicted probabilities from the actual class labels, effectively a form of multi-class cross-entropy:

$$E(\hat{\mathbf{y}}, \mathbf{y}) = -\frac{1}{n} \sum_{i=1}^n (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)) \quad (\text{A.16})$$

A.4.3.4 Cross-Entropy Loss

Used for classification problems, this loss measures the dissimilarity between the true label distribution and the predicted probabilities.

$$E(\hat{\mathbf{y}}, \mathbf{y}) = - \sum_{i=1}^n y_i \log(\hat{y}_i) \quad (\text{A.17})$$

where y_i is the true label and \hat{y}_i is the predicted probability for class i .

A.4.3.5 Hinge Loss

Hinge loss is suitable for binary classification tasks. Although it is not differentiable, it is convex, which makes it helpful as a loss function.

$$E(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{i=1}^n \max(0, 1 - y_i \hat{y}_i) \quad (\text{A.18})$$

A.4.3.6 Kullback–Leibler (KL) Loss

Kullback–Leibler (KL) divergence is a statistical measure that quantifies the difference between two probability distributions. It is particularly useful in generative network scenarios where the output involves continuous probability distributions. The KL divergence loss for comparing the predicted distribution $\hat{\mathbf{y}}$ to the target distribution \mathbf{y} is computed as:

$$E(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^n \left(y_i \cdot \log \left(\frac{y_i}{\hat{y}_i} \right) \right) \quad (\text{A.19})$$

This formula calculates the expected logarithmic difference between the distributions, weighted by the probabilities of the actual distribution. Thus, it provides a measure of how one probability distribution diverges from a second expected probability distribution.

A.4.4 Optimization Techniques

Optimization techniques aim to adjust the model's parameters to minimize the loss function, quantifying the difference between the predicted and actual outcomes. The choice of an optimization algorithm can significantly impact the model's training speed and final performance.

A.4.4.1 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) is a variant of the gradient descent algorithm that updates the model's weights using only a single data point or a mini-batch at each iteration, making it more suitable for large datasets.

$$\theta_{t+1} = \theta_t - \eta \nabla L(\theta_t) \quad (\text{A.20})$$

where θ_t is the parameter vector at iteration t , η is the learning rate, and $\nabla L(\theta_t)$ is the gradient of the loss function with respect to the parameters.

A.4.4.2 Momentum

Momentum is designed to help SGD converge faster by reducing oscillations. It adds a fraction of the previous update vector to the current update, thus aiding in moving over flat regions and dampening oscillations across ravines:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta E(\theta_t), \quad (\text{A.21})$$

$$\theta_{t+1} = \theta_t - v_t, \quad (\text{A.22})$$

where θ_t denotes the parameters at iteration t , γ is the momentum coefficient, and η is the learning rate.

A.4.4.3 Adaptive Gradient (Adagrad)

Adaptive Gradient (Adagrad) adjusts the learning rate individually for each parameter based on the gradient history. It is effective in scenarios with sparse data:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \nabla_{\theta,i} E(\theta_t), \quad (\text{A.23})$$

where $G_{t,ii}$ accumulates the squares of past gradients, and ϵ is a small constant to prevent division by zero.

A.4.4.4 RMSprop

RMSprop modifies Adagrad to improve its robustness by using an exponentially decaying average of squared gradients, thereby addressing its rapidly diminishing learning rates:

$$\mathbb{E}[g^2]_t = \rho \mathbb{E}[g^2]_{t-1} + (1 - \rho) g_t^2, \quad (\text{A.24})$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\mathbb{E}[g^2]_t + \epsilon}} g_t, \quad (\text{A.25})$$

where ρ is a decay factor typically set close to 1.

A.4.4.5 Adaptive Moment Estimation (ADAM)

Adaptive Moment Estimation (ADAM) combines the benefits of Adagrad and RMSprop, adjusting learning rates based on both the first and second moments of the gradients:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \quad (\text{A.26})$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \quad (\text{A.27})$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad (\text{A.28})$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}, \quad (\text{A.29})$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t, \quad (\text{A.30})$$

where β_1 and β_2 are decay rates for the first and second moment estimates, respectively.

A.4.4.6 AdamW

AdamW is a variant of the Adam optimizer that decouples the weight decay from the optimization steps. This modification helps in achieving better training performance and generalization.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla L(\theta_t) \quad (\text{A.31})$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \nabla L(\theta_t)^2 \quad (\text{A.32})$$

$$\theta_{t+1} = (\theta_t - \eta \lambda \theta_t) - \eta \frac{m_t}{\sqrt{v_t} + \epsilon} \quad (\text{A.33})$$

where λ is the weight decay coefficient. The weight decay update is decoupled from the optimization step, leading to the modified update rule.

A.4.5 Model Training

The primary objective of machine learning is to minimize the generalization error, effectively balancing overfitting and underfitting. Due to their numerous parameters, deep learning models are particularly susceptible to overfitting—they are even capable of fitting entirely random training data with zero error.

On the other hand, these models typically settle in local minima during training due to the NP-complete nature of non-convex optimization. However, local minima tend to approximate the global minimum sufficiently in well-regularized networks. In contrast, poorly regularized networks may find local minima with unacceptably high losses.

Achieving minimal disparity between training and validation loss is key to effective model training, requiring careful selection of architecture and training strategies.

A.4.5.1 Early Stopping

Among these methods, early stopping is a prominent technique for preventing overfitting. It involves halting training when the validation error ceases to decrease, despite ongoing reductions in training error, ensuring that the model that performs best on the validation set is chosen. This method assumes proper dataset division into separate training, validation, and testing sets to maintain testing integrity and prevent data leakage.

Early stopping stands out for its simplicity and effectiveness, making it a widely adopted form of regularization in deep learning.

A.4.6 Regularization Techniques

Regularization techniques are essential for preventing overfitting in machine learning models. Overfitting occurs when a model performs exceptionally well on the training data but poorly on unseen data. Regularization adds a penalty to the loss function, constraining the model and making it more general.

A.4.6.1 L1 Regularization (Lasso)

L1 regularization, also known as Lasso regression, adds a penalty proportional to the absolute value of the coefficients. This can lead to some coefficients becoming exactly zero, effectively selecting a simpler model with fewer features.

$$\mathcal{L}_{L1}(\theta) = L(\theta) + \lambda \sum_i |\theta_i| \quad (\text{A.34})$$

where $L(\theta)$ is the original loss function, θ_i represents each coefficient in the model, and λ is the regularization strength.

A.4.6.2 L2 Regularization (Ridge)

L2 regularization, also known as ridge regression, adds a penalty proportional to the square of the magnitude of the coefficients. This tends to shrink the coefficients but does not necessarily make them zero.

$$\mathcal{L}_{L2}(\theta) = L(\theta) + \lambda \sum_i \theta_i^2 \quad (\text{A.35})$$

A.4.6.3 Dropout

Dropout is a regularization technique that is specific to neural networks. During training, random subsets of neurons are “dropped out” or temporarily removed from the network, preventing the co-adaptation of hidden units.

$$h'_i = \begin{cases} 0 & \text{with probability } p \\ h_i & \text{with probability } 1 - p \end{cases} \quad (\text{A.36})$$

where h'_i is the output of a neuron after applying dropout, h_i is the original output, and p is the dropout probability.

A.4.6.4 Batch Normalization

Batch normalization is a technique for improving the training of deep neural networks. It normalizes the output of each layer to have a mean of zero and a variance of one. This can have a regularizing effect and helps in faster convergence.

$$\mu_B = \frac{1}{m} \sum_{i=1}^m x_i \quad (\text{A.37})$$

$$\sigma_B^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2 \quad (\text{A.38})$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad (\text{A.39})$$

$$y_i = \gamma \hat{x}_i + \beta \quad (\text{A.40})$$

where x_i is the input, μ_B is the batch mean, σ_B^2 is the batch variance, \hat{x}_i is the normalized input, and γ and β are learnable parameters.

Appendix B

Reinforcement Learning Basics

Reinforcement learning (RL) is a branch of machine learning in which learning occurs by rewarding desired behaviors and/or punishing undesired behaviors. It is a powerful paradigm for training intelligent agents to make sequential decisions in dynamic environments. Unlike supervised learning, which relies on labeled input-output pairs, reinforcement learning operates by trial-and-error, allowing agents to learn from direct interactions with an environment. The prominent role of RL in contemporary research can be attributed to its numerous practical applications for solving complex sequential decision-making.

B.1 Markov Decision Process

The Markov Decision Process (MDP) is a foundational mathematical framework for RL, as it models situations within a discrete-time, stochastic control process.

In an MDP, as shown in Fig. B.1, a decision-making entity, an agent, engages with its surrounding environment through a series of chronological interactions. The agent obtains a representation of the environmental state at every discrete time interval. By utilizing this representation, the agent proceeds to choose an appropriate action. Subsequently, the environment transitions to a new state, and the agent receives a reward for the consequences of the prior action. During this procedure, the agent's primary objective is to maximize the cumulative rewards obtained from executing actions in specific states.

- **State:** A state represents the current situation or environment in an RL problem. A set of states denoted by (\mathcal{S}).
- **Action:** An action is a decision made by the agent that affects the state of the environment. Represented by A_t , with the set of actions denoted by (\mathcal{A}). At each time step t , the agent receives some representation of the environment's state S_t . Based on this state, the agent selects an action A_t . This gives us the state-action pair (S_t, A_t) . The next increment is $t + 1$, and the environment is transitioned to

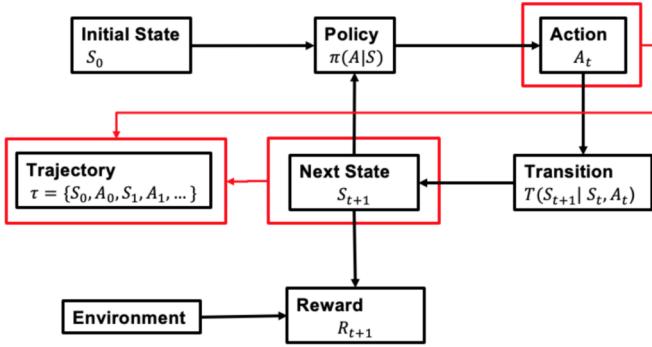


Fig. B.1: Detailed Markov Decision Process for RL

a new state (S_{t+1} , based on the numerical reward R_{t+1} the agent receives for the state-action pair (A_t, S_t) .

- **Reward:** A reward is a numerical value the agent receives as feedback for actions. Rewards are represented by R_t , and the set of all rewards is given by \mathbb{R} . Obtaining a reward can be conceptualized as a generic function that associates state-action pairs with their corresponding rewards and can be given as:

$$f(S_t, A_t) = R_{t+1} \quad (\text{B.1})$$

Given that the sets (\mathbb{S}) and (\mathbb{R}) are finite, the random variables encompassed within these sets, i.e., S_t and R_t , possess well-defined probability distributions. For every state $s = S_t$, subsequent state $s' = S_{t+1}$, action a , and reward r , the transition probability to state s' with associated reward r , upon executing action a in state s , is defined as follows:

$$p(s', r|s, a) = P\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\} \quad (\text{B.2})$$

B.1.1 Tasks

Based on the duration of the agent's interaction with the environment, reinforcement learning has two categories of tasks.

1. **Episodic Tasks:** Episodic tasks are characterized by a well-defined starting point and a terminal state, signifying the end of an episode. Each episode consists of a sequence of states, actions, and rewards experienced by the agent as it interacts with the environment. Once the agent reaches the terminal state, the episode terminates, and the agent's interaction with the environment resets to a new initial state.

2. **Continuing Tasks:** Continuing tasks, also known as non-episodic tasks, do not have a clear terminal state, and the agent's interaction with the environment continues indefinitely. In such tasks, the agent continually perceives states, takes actions, and receives rewards without the interaction resetting.

B.1.2 Rewards and Return

In reinforcement learning, rewards and return are intrinsically linked, as they collectively quantify the success of an agent's actions and guide the learning process toward optimizing its decision-making abilities within the environment. Expected and discounted returns are fundamental concepts used to quantify the long-term value of a particular decision in RL.

1. **Expected Return:** The expected return, or a cumulative or total reward, is the sum of rewards an agent anticipates accumulating over a finite or infinite time horizon, starting from the current state. Mathematically, the expected return G_t for a specific time step t can be defined as the sum of rewards from that time step onward:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \dots + R_T \quad (\text{B.3})$$

where $t = T$ is the final time step.

2. **Discounted Return:** The discounted return is a modification of the expected return that incorporates a discount factor (γ) to account for the preference of an agent to receive rewards sooner rather than later. The discount factor ranges between 0 and 1, with higher values indicating greater importance to future rewards. The discounted return G_t at a specific time step t is given by:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{T-1} R_T \quad (\text{B.4})$$

where $t = T$ is the final time step.

The discounted return emphasizes the agent's preference for immediate rewards and mitigates the potential issue of infinite expected returns in cases where the agent operates in an infinite-horizon environment. By optimizing the discounted return, the agent learns a policy that balances the trade-off between short-term and long-term rewards, contributing to more effective decision-making in complex environments.

By rewriting the above equation, we can show how the returns at the steps are related.

$$G_t = R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots + \gamma^{T-2} R_T) \quad (\text{B.5})$$

$$G_t = R_{t+1} + \gamma G_{t+1} \quad (\text{B.6})$$

Even though the return at time t is a sum of an infinite number of terms, the return is finite as long as the reward is nonzero and the discount factor $\gamma < 1$. This is a critical feature for continuous tasks, which could accumulate a theoretically infinite reward without discounting because they lack a defined endpoint. Consider a scenario where the reward at each time step remains constant (1) and $\gamma < 1$, then the return is:

$$G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma} \quad (\text{B.7})$$

with the reward decreasing over infinite time steps denoted by k .

B.1.3 Policies and Value Functions

Next, we address two fundamental questions in the context of reinforcement learning: (1) the likelihood of an agent selecting a particular action within a given state (policies) and (2) evaluating the desirability or quality of action about a specific state for the agent (value functions).

Policy: A policy is a strategy or set of rules that dictate the agent's actions in each state. A policy is denoted by $(\pi(a|s))$, representing the probability of taking action a in the state s . Thus, for each state, $s \in \mathbb{S}$, π is a probability distribution over action $a \in \mathbb{A}$.

For example, considering again our robot traversing its maze, one could have a random policy where left, right, up, or down are equally probable at each step. A smarter policy would modify these numbers based on the specific state, for example, reducing the probability (perhaps to 0) of moving in a direction that bumps into a wall. Reinforcement learning involves using trials and rewards to refine an agent's policy into a desirable state.

Value function: The value function estimates the expected cumulative reward the agent will receive if it follows a specific policy from a particular state onward. Formally, it is represented by $(v_{\pi(s)})$, denoting the expected cumulative reward from following policy (π) starting in the state (s) . There are two fundamental value functions.

1. **State-Value Function** The state-value function represents the return (accumulated discounted rewards) an agent can obtain from a particular state, s , when following a specific policy, π . Mathematically, the state-value function under a policy π is defined as:

$$v_{\pi}(s) = E_{\pi} [G_t | S_t = s] \quad (\text{B.8})$$

$$v_{\pi}(s) = E_{\pi} \left[\sum_{k=1}^{\infty} \gamma^{k-1} R_{t+k} \middle| S_t = s \right] \quad (\text{B.9})$$

where S_t is the state at time t and k are all time steps after t . R_{t+k} is the reward received after transitioning to state S_{t+k} , and $\gamma \in [0, 1]$ is the discount factor, which determines the relative importance of immediate rewards over future rewards. The expectation, E_π , is taken over all possible trajectories generated by following policy π .

2. **Action-Value Function** The action-value function, also known as the *Q-function* and denoted as $q_\pi(s, a)$, represents the expected long-term return that an agent can obtain from taking a particular action, a , in a given state, s , and subsequently following a specific policy, π . The output from any state-action pair is called the *Q-value*. The symbol Q represents the quality of taking a specific action in a particular state. Mathematically, the action-value function under a policy π is defined as:

$$q_\pi(s, a) = E_\pi [G_t | S_t = s, A_t = a] \quad (\text{B.10})$$

$$q_\pi(s, a) = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a \right] \quad (\text{B.11})$$

where A_t is the action at time t . Like the state-value function, the expectation, E_π , is taken over all possible trajectories generated by following the policy π .

The state-value and action-value functions are closely related, and one can be derived from the other. The state-value function can be obtained from the action-value function as follows:

$$v_\pi(s) = \sum_{a \in \mathbb{A}} \pi(a|s) q_\pi(s, a) \quad (\text{B.12})$$

where s is the current state, a is the action, \mathbb{A} is the set of actions, and $\pi(a|s)$ is the probability of taking action a in state s under policy π .

In this equation, the term $\pi(a|s)q_\pi(s, a)$ represents the expected value of taking action a in state s when following policy π . By summing this value over all possible actions in the set \mathbb{A} , we obtain the state-value function, $v_\pi(s)$, which represents the expected long-term return for being in state s and subsequently following policy π . The action-value function can be obtained from the state-value function as:

$$q_\pi(s, a) = \sum_{s' \in \mathcal{S}} P(s'|s, a) [R(s, a, s') + \gamma v_\pi(s')] \quad (\text{B.13})$$

where s is the current state, a is the action taken, s' is the next state, $R(s, a, s')$ is the immediate reward for taking action a in state s and transitioning to state s' , and γ is the discount factor, which determines the relative importance of immediate rewards over future rewards. $P(s'|s, a)$ is the state transition probability, representing the probability of transitioning from state s to state s' when taking action a .

In this equation, the term $\sum_{s' \in \mathcal{S}} P(s'|s, a)R(s, a, s')$ calculates the expected immediate reward for taking action a in state s , while the term $\gamma \sum_{s' \in \mathcal{S}} P(s'|s, a)v_\pi(s')$

calculates the expected discounted return for subsequent states, weighted by the state transition probabilities. These two terms yield the action-value function, $q_\pi(s, a)$, representing the expected long-term return for taking action a in state s and following policy π .

B.1.4 Optimality

The objective of reinforcement learning algorithms is to discover a policy that maximizes the accumulation of rewards for the agent when it adheres to that policy. More specifically, reinforcement learning algorithms aim to identify an approach that outperforms all other policies by generating a higher return for the agent.

- **Optimal Policy:** A policy π is considered superior to or equal to policy π' based on its return, where the anticipated return of policy π is greater than or equal to the anticipated return of policy π' for all states. Mathematically, this is

$$\pi > \pi' \text{ if and only if } v_\pi(s) > v_{\pi'}(s) \forall s \in \mathbb{S} \quad (\text{B.14})$$

- **Optimal State-Value Function:** The optimal state-value function, denoted as $v_*(s)$, represents the highest expected long-term return achievable by an agent starting in state s when following an optimal policy. The optimal state-value function can be expressed as:

$$v_*(s) = \max_{\pi} v_{\pi}(s) \quad (\text{B.15})$$

for all states $s \in \mathbb{S}$.

- **Optimal Action-Value Function:** The optimal action-value function, known as the optimal Q-function and denoted as $q_*(s, a)$, represents the highest expected long-term return achievable by an agent starting in state s , taking action a , and following an optimal policy. The optimal action-value function can be expressed as:

$$q_*(s, a) = \max_{\pi} q_{\pi}(s, a), \quad (\text{B.16})$$

for all states $s \in \mathbb{S}$ and actions $a \in \mathbb{A}$.

- **Bellman Optimality Equation:** Using the Q-function, the Bellman optimality equation states that the optimal Q-value for a given state-action pair equals the immediate reward obtained from taking that action in the current state plus the maximum expected return achievable from the next state after the optimal policy. The Bellman optimality equation for the Q-function can be rewritten as follows, in terms of the expected reward R_{t+1} obtained by taking action a in state s and the maximum expected discounted return that can be achieved from any possible next state-action pair (s', a') :

$$q_*(s, a) = \mathbb{E} \left[R_{t+1} + \gamma \max_{a'} q_*(s', a') \mid S_t = s, A_t = a \right] \quad (\text{B.17})$$

where:

- R_{t+1} is the immediate reward obtained by taking action a in state s .
- γ is the discount factor that balances the importance of immediate and future rewards.
- $\max_{a'} q_*(s', a')$ represents the maximum expected discounted return that can be achieved from any possible next state-action pair (s', a') , given that the agent follows the optimal policy thereafter.

B.2 Exploration/Exploitation Trade-off

An additional important concept to consider for RL is the *exploration/exploitation trade-off*. This refers to two opposing strategies an algorithm can use to obtain rewards. If there is a strong bias toward exploration, the agent will be motivated to find its way to unfamiliar states to discover a large reward. On the other hand, if there is a greater preference for exploitation, the agent will repeatedly capitalize on states with known rewards, thus losing out on potentially higher rewards in unexplored states. The extent to which a given algorithm prioritizes exploration is integral to its design. The most straightforward strategy is greedy, which means always choosing the action with the highest reward at every step. This is a maximally exploitative strategy with no exploration at all. Since it is typically necessary to allow the agent to explore, a variant called ϵ -greedy is often employed instead. This introduces a value ϵ that dictates how often the agent should take random actions rather than behave greedily. For instance, $\epsilon = 0.4$ would result in a random action being taken 40% of the time, with the remaining 60% of actions aiming to achieve the maximum reward.

In practice, the value of ϵ is often adjusted throughout training. In early iterations, a higher value is used to collect feedback on as many state-action pairs as possible. Later on, reducing ϵ will increase the likelihood of converging on an optimal policy. Notably, this idea is aligned with the principle of learning rate scheduling, which is prevalent across many areas of deep learning.

B.3 Reinforcement Learning Algorithms

Various RL algorithms have been developed, each with unique attributes that make them suitable for specific problems. Value-based algorithms, such as *Q-Learning* and *Deep Q-Networks* (DQN), estimate the value or quality of each action taken in each state. These algorithms revolve around a value function, which assigns a value to each possible state-action pair based on the expected cumulative reward.

On the other hand, policy-based algorithms, such as *Proximal Policy Optimization* (PPO), directly optimize the policy, i.e., the mapping from states to actions. These

algorithms can handle high-dimensional action spaces and are particularly effective in continuous control tasks.

In addition to the value and policy-based distinction, RL algorithms can be categorized based on whether they are model-based or model-free. Model-based methods incorporate a model of the environment into the learning process, allowing the agent to plan ahead by predicting the consequences of its actions. Model-free methods, including Q-Learning, DQN, and PPO, do not require a model of the environment and learn solely from direct interaction with the environment.

Furthermore, off-policy and on-policy algorithms distinguish themselves by how they use data to learn. Off-policy algorithms, such as Q-Learning and DQN, can learn from historical data generated by any policy, not necessarily the agent's current policy. On-policy algorithms, such as PPO, require data generated by the current policy, making them more data-hungry but often yielding more stable learning.

In addition to these, RL algorithms can be categorized based on their sampling and exploration strategies. Sampling-based methods involve generating and evaluating candidate solutions to optimize the agent's policy. These methods, which include Monte Carlo Tree Search, are particularly effective in environments with large action spaces but relatively small state spaces.

Algorithm Class	Type	Policy	Description
Q-Learning	Value-based	Off-policy	Employs a value-based strategy focusing on the maximization of the total reward by learning the value of actions in given states.
Deep Networks	Q- Value-based	Off-policy	Enhances Q-Learning by integrating deep learning, improving its ability to handle high-dimensional state spaces through value-based strategies.
Proximal Policy Optimization	Policy-based	On-policy	Utilizes a policy-based approach to directly learn the policy function while ensuring small updates, enhancing stability and performance in training.

In the following sections, we delve into the fundamental concepts and mathematical principles of three central RL algorithms: Q-Learning, a value-based and off-policy method; DQN, an extension of Q-Learning that integrates deep learning; and some of the Policy Gradient methods such as TRPO and PPO that have proven effective in complex, continuous control tasks.

B.3.1 Q-Learning

Q-Learning is an off-policy RL algorithm developed by Chris Watkins. The Q in Q-learning stands for quality, reflecting the algorithm's aim to iteratively learn the quality of actions, determining how valuable a given action is in a given state.

The Q-table is at the core of Q-Learning, a matrix where each row represents a possible state, and each column represents a possible action, as shown in Fig. B.2. Q-Learning seeks to estimate the action-value function $q(s, a)$, which represents the expected cumulative discounted reward of taking action a in state s and following an optimal policy. The optimal action-value function, denoted as $q_*(s, a)$, satisfies the Bellman optimality equation.

	Actions				
	A_1	A_2	\dots	A_m	
S_1	$q(S_1, A_1)$	$q(S_1, A_2)$			$q(S_1, A_m)$
S_2	$q(S_2, A_1)$	$q(S_2, A_2)$			$q(S_2, A_m)$
\dots				\dots	\dots
S_n	$q(S_n, A_1)$	$q(S_n, A_n)$		\dots	$q(S_n, A_m)$

Fig. B.2: Q-Table

Q-Learning is an iterative algorithm that updates the $q(s, a)$ estimates online. At each time step t , the agent observes the current state (S_t), selects and executes an action (A_t), and then receives a reward (R_t) and observes the next state (S_{t+1}). The Q-value for the current state-action pair (S_t, A_t) is updated using the observed reward and the estimate of the optimal future value:

$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a q(S_{t+1}, a) - q(S_t, A_t) \right] \quad (\text{B.18})$$

In this equation:

- $q(S_t, A_t)$ - Q-value of the state-action pair (S_t, A_t) at time step t .
- α - Learning rate, determining how much the Q-value changes in each iteration.

- R_{t+1} - Reward obtained at the next time step $t + 1$.
- γ - Discount factor, determining the importance of future rewards relative to immediate rewards.
- $\max_a q(S_{t+1}, a)$ - Maximum Q-value over all possible actions a in the next state S_{t+1} .
- $q(S_t, A_t)$ - Current Q-value of the state-action pair.

An exploration-exploitation trade-off strategy usually dictates the agent's action selection in Q-Learning. A common approach is to use an ϵ -greedy policy, which selects a random action with probability ϵ and the action with the highest Q-value estimate with probability $1 - \epsilon$. As learning progresses, ϵ is typically decreased to favor exploitation over exploration.

B.3.2 Deep Q-Network (DQN)

DQN is an extension of Q-learning that uses a deep neural network to approximate the Q-function. This was a significant breakthrough, allowing Q-Learning to handle environments with high-dimensional state spaces, such as those in video games.

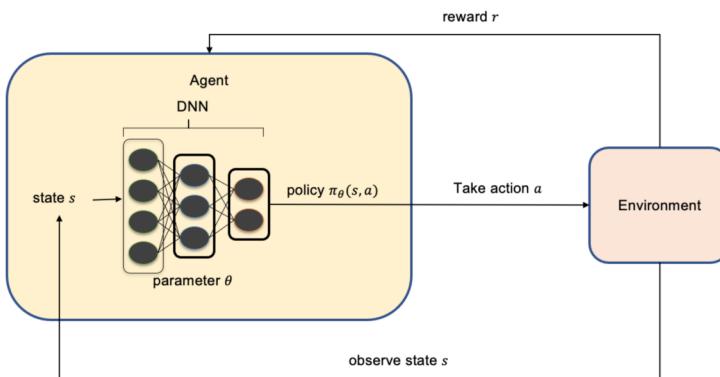


Fig. B.3: Deep Q-Network in the state-action framework.

In DQN, the role of the Q-table is replaced by a deep neural network, which takes the state as input and outputs the Q-value for each action, as shown in Fig. B.3. One key innovation of DQN is the use of a technique called Experience Replay. Rather than updating the network based on each new piece of experience individually, the agent stores the experience in a replay buffer and later samples a batch of experiences to update the network. This allows for greater data efficiency and stability [4].

The loss function for DQN at each iteration i is given by:

$$\mathcal{L}(\theta) = \mathbb{E}_{(S_t, A_t, R, S_{t+1}) \sim U(D)} \left[\left(R + \gamma \max_{A_{t+1}} q_{\text{target}}(S_{t+1}, A_{t+1}; \theta^-) - q_{\text{predicted}}(S_t, A_t; \theta) \right)^2 \right] \quad (\text{B.19})$$

where:

- θ - Parameters of the Deep Q-Network.
- $\mathbb{E}_{(S_t, A_t, R, S_{t+1}) \sim U(D)}$ - Expectation over a mini-batch of experience samples from the replay buffer.
- (S_t, A_t, R, S_{t+1}) - Current state, action, reward, and next state.
- $U(D)$ - Uniform distribution over the replay buffer.
- R - Immediate reward received after taking action A_t in state S_t .
- γ - Discount factor, emphasizing the importance of future rewards.
- $\max_{A_{t+1}}$ - Maximum over all possible actions at the next step A_{t+1} .
- $q_{\text{target}}(S_{t+1}, A_{t+1}; \theta^-)$ - Target Q-value using the target network parameters θ^- .
- $q_{\text{predicted}}(S_t, A_t; \theta)$ - Predicted Q-value using the current network parameters θ .

B.3.3 Policy Gradient-based Methods

Policy Gradient-based (PG) methods form the basis of several policy optimization algorithms that seek to optimize policies directly. They involve optimizing the policy parameters by directly estimating the gradient of the expected return concerning the policy's parameters. By maximizing the policy gradient, agents can learn to take actions that lead to higher rewards in a given environment, enabling them to improve their performance over time.

The basic policy gradient loss is calculated as the negative log probability of the action multiplied by the corresponding advantage estimate (which estimates how much better or worse an action is compared to the average action taken in that state).

:

$$\mathcal{L}^{\text{PG}} = -\frac{1}{N} \sum_{i=1}^N \log(\pi(A_t | S_t)) \cdot A^{\text{adv}}(S_t, A_t) \quad (\text{B.20})$$

where:

- N - the number of samples in the batch used for averaging the gradient estimates.
- $\log(\pi(A_t | S_t))$ - the logarithm of the probability of choosing action A_t given state S_t under the policy.
- $A^{\text{adv}}(S_t, A_t)$ - the advantage estimate, quantifying the relative value of action A_t in state S_t .

In PG methods, policy updates are typically performed using stochastic gradient ascent, which can lead to large updates and instability.

B.3.3.1 Trust Region Policy Optimization(TRPO)

addresses this by constraining the policy update to a region where it is likely to improve without deviating too far from the current policy. The key idea behind TRPO is to maximize the performance objective while ensuring that the updated policy remains close to the previous policy within a specified trust region. A maximum allowable KL divergence between the updated and old policies defines this trust region. To achieve this, TRPO solves a constrained optimization problem. It computes the policy update that maximizes the performance objective, subject to the trust region constraint. The objective function is typically a surrogate objective that approximates the expected improvement in performance. The TRPO loss ($\mathcal{L}^{\text{TRPO}}$) and its corresponding constraints are fundamental components of the Trust Region Policy Optimization (TRPO) algorithm. The TRPO algorithm aims to optimize the policy parameters while ensuring a reasonable update from the old to the new policy.

The TRPO loss is defined as follows:

$$\mathcal{L}^{\text{TRPO}} = \mathbb{E}_t \left[\frac{\pi(A_t|S_t)}{\pi_{\text{old}}(A_t|S_t)} \cdot A^{\text{adv}}(S_t, A_t) \right] \quad (\text{B.21})$$

where:

- \mathbb{E}_t - the expectation over time step t , evaluating policy updates.
- $\pi(A_t|S_t)$ - the probability of selecting action A_t given state S_t under the current policy.
- $\pi_{\text{old}}(A_t|S_t)$ - the probability of selecting action A_t given state S_t under the old policy, used for baseline comparison.
- $A^{\text{adv}}(S_t, A_t)$ - the advantage estimate, indicating the extra gain from action A_t compared to the average in state S_t .

The TRPO algorithm also imposes constraints to ensure a conservative update to the policy. The constraints are formulated as follows:

$$\mathbb{E}_t [\mathbb{KL}[\pi_{\text{old}}(\cdot|S_t), \pi(\cdot|S_t)]] \leq \delta \quad (\text{B.22})$$

where:

- \mathbb{KL} denotes the Kullback-Leibler divergence, which measures the difference between two probability distributions.
- $\pi_{\text{old}}(\cdot|S_t)$ is the probability distribution under the old policy.
- $\pi(\cdot|S_t)$ is the probability distribution under the updated policy.
- δ represents a threshold or limit on the KL divergence. A sensible default value would be approximately 0.1, but it can be optimized through a hyperparameter search.

B.3.3.2 Proximal Policy Optimization (PPO)

PPO addresses some of the computational inefficiencies of Trust Region Policy Optimization (TRPO) while maintaining effective policy updates. One of the main challenges with TRPO is the need to solve a constrained optimization problem, which can be computationally expensive. PPO simplifies this by reformulating the problem as an unconstrained optimization. Instead of explicitly enforcing a trust region constraint, PPO introduces a clipping mechanism in the objective function. The key idea behind PPO is to construct a surrogate objective function that approximates the expected improvement in performance while simultaneously constraining the policy update to be within a reasonable range. The surrogate objective combines the new and old policy probabilities ratio multiplied by the advantage estimate. The advantage estimate represents the relative value of an action in a given state. The clipping mechanism in PPO limits the policy update to a “trusted” region by constraining the surrogate objective. This effectively prevents huge policy updates and ensures that the new policy remains close to the old policy.

$$\mathcal{L}^{\text{PPO}} = \mathbb{E}_t \left[\min \left(r_t(\theta) \cdot A^{\text{adv}}(S_t, A_t), \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \cdot A^{\text{adv}}(S_t, A_t) \right) \right] \quad (\text{B.23})$$

where:

- \mathbb{E}_t denotes the expectation over time step t .
- $A^{\text{adv}}(S_t, A_t)$ is the advantage estimate, which represents the relative value of action A_t in state S_t .
- $\text{clip}(\cdot, a, b)$ is a function that clips its input between a and b .
- θ represents the policy parameters.
- S_t denotes the state at time step t .
- A_t denotes the action at time step t .
- $\pi(A_t|S_t)$ is the probability of selecting action A_t given state S_t under the current policy.
- $\pi_{\text{old}}(A_t|S_t)$ is the probability of selecting action A_t given state S_t under the old policy and is used as a reference.
- ϵ is a hyperparameter that controls the magnitude of the clipping.

Furthermore, PPO utilizes multiple epochs of optimization on collected data, which allows for more efficient policy updates and better sample utilization. This helps to improve the sample efficiency of the algorithm.

Index

- 3D Objects, 378
AdaLoRA, 152
Adam Optimizer, 58
AdamW Optimizer, 58
Adaptive Pre-training, 15–17, 343
Adversarial Attacks, 248
Agents, 188
AI Assistants, 317
ALBERT, 64, 112
Alignment, 222
Alignment Tuning, 18–19, 178–184, 358
AlignScore, 222
Alpaca, 424
Alternate Language Modeling, 53
Answer Engineering, 113–115
Answer Faithfulness, 294, 297–300
Answer Mapping, 94
Answer Relevance, 294, 300–301
Answer Search, 93–94
Anthropic, 11, 206, 212, 214, 341, 424
ArXiv, 400
Attention, 9, 11, 15, 32–33, 41, 339
Attribute Conditioning, 251
Audio-Text Modality, 384
AudioLDM, 411
AutoGPT, 433
Automated Answer Mapping, 115
Automated Prompt, 103
Autonomous Vehicle Navigation, 379
AutoPrompt, 106, 110
Autoregressive Decoding, 58
AWS Bedrock, 346, 351
AWS Sagemaker Jumpstart, 346
BabelNet, 231
BabyAGI, 433
Backpropagation, 185, 253
BART, 110
BatchNorm, 39
Bayesian Networks, 8
Beam Search, 59, 210
BEATs, 379
BentoML, 430
BERT, 10, 43, 61–64, 76, 86, 325
BERTScore, 323, 325–326
BGE, 283
Bias, 229–247
Bias Mitigation, 241
Big-bench workshop, 190
BigScience, 321
Bingo, 399
BioMistral, 322
BitFit, 154
Bitwidth, 158
Black-box Probing, 258
BLEU, 323–324

- Blocklisting, 252
 BLOOM, 159, 321, 430
 BloombergGPT, 17, 431
 BOLD, 236, 238
 Bradley-Terry Model, 195, 211
 Breadth-first Search, 168
 Brevity Penalty, 324
 BriVL, 381
 C-Former, 379
 Cassandra, 346
 Catastrophic Forgetting, 244
 Categorical Bias Score, 235
 Causal Decoder Architecture, 16, 55
 Chain-of-Hindsight, 183
 Chain-of-Thought, 21, 166–167, 328
 Chatbots, 22, 47, 180, 189, 213, 281, 316–317, 376
 ChatGPT, 1–2, 10–12, 137, 161, 232, 248, 257, 276, 318, 342, 358
 Chess, 209
 Chinchilla, 39, 47, 72, 404
 Chroma, 346
 CIFAR-10, 428
 Claude, 280
 Claude 2, 137
 Claude 2.1, 424
 Claude 3, 72
 ClinicalBERT, 49
 CLIP, 377
 CLIP ViT, 378
 Cloud Modality Encoding, 379
 Cloud Services, 346
 Cloze Prompt, 101–114
 CMMU, 398
 CNNs, Convolutional Neural Networks, 8, 249
 Co-Occurrence Bias Score, 236
 Code Auto-completion, 319
 Code-Switching, 321
 Codex-Med, 431
 Coding, 318
 Command R, 73
 Confabulation, 167
 ConfAIde, 259
 Constitutional AI, 206, 207
 Content Creation, 317–318
 Context Association Test, 236
 Context Consolidation, 290–291
 Context Length, 315
 Context Relevance, 294–297
 Context Window, 425
 Contextual Embeddings, 234, 325
 Contextualized Embedding Association Test, 234
 Continuous Batching, 352
 Continuous Integrate-and-Fire Alignment Mechanism, 379
 Continuous Prompt, 108–111, 242
 Contrastive Learning, 385–386
 Conversational AI, 317
 Cosine Similarity, 285, 301
 Counterfactual Data Augmentation, 238, 241
 Counterfactual Data Substitution, 241
 Counterfactual Robustness, 294, 305–306
 Counterfactual Supporting Information, 306
 CRITIC, 228
 Cross-Attention, 382
 Cross-Stream Architecture, 380
 CrowS-Pairs Score, 235
 DALL-E 2, 377
 Data Governance, 361
 Data Parallelism, 60
 Data Security, 316, 341–342
 Data Versioning, 356
 Decoder, 34
 DeepMind, 194
 DeepSpeed, 60
 Demographic Representation, 236
 Denoising Autoencoder, 55
 Depth-first Search, 168
 Differential Privacy, 260
 Dify.Ai, 345
 DisCo, 234
 Discrete Prompt, 107

- DistilBERT, 64
DistilGPT-2, 169
DLite, 214
Domain Adaptation, 142–144, 251
Domain-Specific Language Models, 430–431
Dot Product, 32
DPO, Direct Preference Optimization, 210–212
DSPy, 345
Dynamic Memory Network, 84

Early Concatenation, 382
Early Summation, 382
ELECTRA, 64
Elo, 209
Embedding Models, 280, 283
Embeddings, 233, 346
Emergent Abilities, 12–13
Encoder, 33
Encoder Architecture, 57
Encoder Decoder Architecture, 16, 29, 55
Encoder-Decoder Attention, 38
Encoder-Only Architecture, 15, 30
Encoder-only Architecture, 30
End-User Prompting, 20
Entity Tagging, 96
Error Correction Rate, 306
Error Detection Rate, 306
Eva-CLIP-ViT, 378
Exact Match, 302, 304
Expectation-Maximization Algorithm, 8
Expected Maximum Toxicity, 249
Experiment Tracking, 356
Exploding Gradients, 31

F1 Score, 326
Fact Retrieval, 107
Fact Verification, 222
FacTool, 228
FActScore, 221, 225
Factual Consistency, 222
FactualityPrompt, 223
Fairness, 233

FAISS, 346
Faithfulness, 284
Feed-Forward Neural Network, 36, 42
Feedback Model, 208
Few-Shot, 89, 285
Few-Shot Prompts, 164
Financial Services, 342
FinBERT, 431
Fine-tuning, 19, 305, 340
Flamingo, 401–405
Flan-PaLM, 431
FLARE, 434
Flowise, 344
Frame Order Modeling, 390

G-Eval, 323, 328–329
Galactica, 431
GAR, Generation-Augmented Retrieval, 285
Gated State Space, 428
GatorTron, 431
GeGLU Activation, 41
Gemma, 73, 429
Generative Discriminator, 253
Geometric Mean, 324
Gesture-Based Control with Audio Feedback, 376
git (version control), 356
Github Copilot, 318
Github Gists, 320
GLUE, 67
Google Vertex AI, 346
Gopher, 39, 72
GPipe, 426
GPT-1, 68, 69
GPT-2, 11, 68–70, 77, 106, 110, 157, 196, 251, 256, 401
GPT-3, 11, 58, 68–70, 84, 141, 142, 155, 158, 166, 178, 222
GPT-3.5, 68, 73, 280
GPT-4, 169, 220, 259, 280, 283, 299, 347, 362, 399
GPT-4V, 399
GPT-Neo-6, 257

- GPT4All, 262
- Gradient Checkpointing, 426
- Gradient Descent, 210
- Greedy Search, 58–59, 210
- Ground-truth Answers, 301
- Groundedness, 189, 190, 193, 298
- GSM8K, 166
- Hallucination, 167, 219–229, 275, 285, 297, 399
- HaluEval, 225
- Harmlessness, 178, 182–183, 206
- Head-based Tuning, 91
- Healthcare, 342
- Helpfulness, 178–180, 209
- HERO, 389
- Hierarchical Attention, 382
- Hit Rate, 296
- Honest Metric, 237
- Honesty, 178, 181–182
- HuBERT, 379
- HuggingFace, 87, 147, 320, 344, 430
- Human Evaluation, 221–222, 322
- Human Preferences, 212
- Human-based Judgement, 298
- Human-Generated Content, 318
- Human-in-the-Loop Processes, 359
- HurtLex corpus, 237
- HyDE, Hypothetical Document Encoding, 286
- ICL, In-Context Learning, 20–21, 110, 118–119, 341
- Image and Video Captioning, 376
- Image Classification, 377
- Image-Text Generation, 388–389
- Image-Text Matching, 388
- Image-Text Modality, 384
- Image-Text Retrieval, 376
- ImageBind, 377
- In-training Mitigation, 242–244
- Index Storage, 280
- Indexing, 276
- Inference Latency, 337–339
- Infini-attention, 425
- Information Integration, 294, 303–304
- Information Retrieval, 318
- Input Projector, 379–383
- Instruct-FinGPT, 143
- InstructGPT, 11, 68, 200, 213
- InstructGPT , 141–142
- Instruction Tuning, 18, 136–144, 358
- INT8, 158, 159
- InterBERT, 382
- Intrinsic Dimensions, 150, 153, 155
- Iterative Retrieval, 289
- Jail-Breaking, 184
- Jailbreak, 342
- JavaScript, 319, 344
- JetBrains, 320
- K-F1; Knowledge-F1, 298
- K-Precision, Knowledge-Precision, 298
- Kaiming Initialization, 157
- Key-Value Caching, 353
- Knowledge Base, 95, 228, 275, 281, 305
- Knowledge Distillation, 427–428
- Knowledge Extraction, 94
- Knowledge Inherited Pre-training, 50
- KOSMOS-1, 377
- Kronecker Adapters, 151, 153–154
- Kullback-Leibler Divergence, 199, 212, 237
- LaMDA, 189–191
- LAMM, 400
- LangChain, 288, 344, 433
- LangKit, 363
- Language Modeling, 7–8
- Layer Normalization, 37
- LayerNorm, 39
- Learning Rate, 58
- Lexical-based Metrics, 298
- Licensing, 316, 341–342
- LIME, 247
- Llama, 72, 143–144
- Llama-2, 73, 75, 179, 180, 280, 347

- Llama-3, 429
LlamaIndex, 276, 283, 288, 291, 293, 307, 344, 433
LLM Agents, 431–433
LLM Evaluation, 323
LLM.int8, 160
LLMCritic, 299
LLMDATAHub, 430
LLMOps, 316, 345, 354–361
LoftQ, 161–162
Log-Probability Bias Score, 235
Logistic Regression, 8
Longest Common Subsequence, 325
LongRoPE, 425
LoRA, 19, 145, 151–153, 155, 161, 168
LoRAHub, 152
- M3DBench, 399
Machine Translation, 55, 149
Manual Answer Mapping, 115
Manual Prompt, 101–103, 242
Markov Decision Process, 186–188
Masked Language Modeling, 51, 62, 387
Masked Multi-Headed Attention, 38
Masked Object Classification, 387–388
Megatron-LM, 76
Metaverse, 379
Milvus, 346
Mistral 7B, 322, 429
Mixtral, 73–76
Mixture of Experts, 57, 73
MLE, Maximum Likelihood Estimation, 212
MLOps, 354
MMBench, 399
MMCBench, 399
MMHAL-BENCH, 399
MMMU, 398
MMVP, 399
Modality Alignment, 379
Modality Encoder, 378–379
Modality Generator, 398
- Modality Matching Loss, 386–387
Modality-Switching Instruction Tuning, 410
Model Governance, 361
Model-based Metrics, 298–300
Modular RAG, 281
MOE, Mixture of Experts, 74–76
Monitoring, 357–358
MoverScore, 326–328
MRR, Mean Reciprocal Rank, 296
MS COCO, 410
Multi-Head Attention, 36
Multi-Hop Question, 289
Multi-Prompt Inference, 116–120
Multi-task Learning, 65
Multi-turn Conversation, 190
Multilingual Large Language Models, 321
Multimodal Chain-of-Thought, 391, 394–395
Multimodal In-Context Learning, 392–394
Multimodal Instruction Tuning, 391–392
Multimodal Large Language Models, 321, 375–415
Multimodal RLHF, 395–397
MultiNLI, 68
Multitoken Generation, 426–427
MXNet, 426
- Natural Language, 3–5
Natural Language Inference, 61, 107, 222, 299
Natural Language Understanding, 5, 319
nDCG, Normalized Discounted Cumulative Gain, 297
Negative Contexts, 302
Negative Rejection, 294, 302–303
NER, Named-Entity Recognition, 10, 18, 299
Neural-Chat-v3-1, 424
New York Times, 342
Next Sentence Prediction, 54, 62

- NExT-GPT, 377, 401, 408–411
 NLP, Natural Language Processing, 5–8
 Noise Robustness, 294, 301–302
 Nomic AI, 262
 NVIDIA’s Triton Inference Server, 352
 OCR, 399
 One-Shot, 89
 Ontological Knowledge, 220
 OpaquePrompts, 261
 OpenAI, 68, 140, 196, 261, 277, 308, 318
 OpenHermes, 424
 OpenLLM, 430
 OPENWEBTEXTCORPUS, 251
 Operant Conditioning, 185
 OPT-175B, 430
 Output Projector, 397
 Overfitting, 244, 256
 PaLM, 11
 Parallel Adapters, 146, 149–150, 154
 Parallelism, 338
 Parametric Knowledge, 94, 225, 228, 301, 306
 Parametric Memory, 228
 Pass at k, 329
 PBL, Prompt-based Learning, 138
 PEFT, Parameter-efficient fine-tuning, 14, 19–20, 144–157, 384
 Perplexity, 259, 291, 323
 Persona-Assigned Prompts, 248
 Phi-1.5, 225
 Phi-2, 429
 PII, Personally Identifiable Information, 256
 Pinecone, 346
 Plug and Play Language Model, 253
 Positional Encodings, 35–36
 Positional Skip-wise Training, 425
 Post-processing Mitigation, 246–247
 Post-training Quantization, 159
 Postgres, 346
 PPO, Proximal Policy Optimization, 197, 227
 Precision, 295, 324, 326
 Predibase, 354
 Prefix Decoder Architecture, 16, 56
 Prefix Prompt, 100–101
 Prefix Tuning, 110
 PrefixLM, 387
 Privacy, 183, 256–262
 Prompt Addition, 92–93
 Prompt Chaining, 164–165
 Prompt Decomposition, 119–120
 Prompt Engineering, 97–168, 341, 347–349
 Prompt Ensembling, 116
 Prompt Hub, 349
 Prompt Mining, 103–104
 Prompt Paraphrasing, 104–105
 Prompt Versioning, 357
 Prompt-based fine-tuning, 113, 136
 Prompt-based Learning, 88–92
 Promptmetheus, 348
 ProPILE, 257
 Pseudo-log-likelihood, 235
 Pythia, 256
 pytorch, 346, 426
 QLoRA, 153, 161, 168
 Quantization, 158–162, 339
 Quantization Aware Fine-tuning, 160–162
 Quantization-aware Training, 158
 Query Rewriting, 284–285
 Query-to-Document, 285
 query2doc, 286
 Question Answering, 222
 Question NLI, 68
 Question-Answer Pairs, 303
 Question-Supporting Information-Answer Triplets, 304
 RAG Evaluation, 293–307
 RAG, Retrieval Augmented Generation, 23, 276–311
 RAG-Fusion, 285

- Ragas, 300
Random Search, 59–60
Random Token Substitution, 52
Rank-Agnostic Metrics, 294
Rank-Aware Metrics, 294, 297
RealToxicityPrompts, 203, 236, 238
Recall, 294, 295, 326
Reciprocal Rank Fusion, 285
Red Teaming, 206
Reddit, 197
RedditBias, 238
Reflexion, 432
Reinforcement Learning, 184–188, 202
ReLU Activation, 37
Reparameterization, 146, 150–157
Replaced Token Detection, 51
Representational Harm, 230
Reward Model, 188, 197–199, 201, 227
Reweighting, 240–241
Risk Management, 359–360
RLHF, Reinforcement Learning from Human Feedback, 189–204
RNNs, Recurrent Neural Networks, 31
RoBERTa, 64, 155, 157
ROUGE, 323, 325
- Safety, 190
Safety Risks, 341
SaulLM-7B, 431
Scale, 12
SciGraphQA, 400
Segmentation, 281
Self-Attention, 33
Self-Consistency, 167
Self-Debiasing, 254
Self-Supervised Contrastive Learning, 380
Self-Supervised Language Modeling, 43
Self-Supervised Learning, 50
SELFCHECKGPT, 229
- Semantic Similarity, 283, 286, 289, 300, 323
SemCor, 232
Sensibleness, Specificity, and Interestingness (SSI), 189–191
Sentence Boundary Objective, 53–54
Sentence Encoder Association Test, 234
Sentence Order Prediction, 54
SentencePiece, 190
Sentiment Analysis, 94
Sentiment Classification, 92
Sequence Transduction, 379
Series Adapters, 146, 148–149
SHAP, 247
Shuffled Token Detection, 52
Single-Stream Architecture, 380
Small Language Models, 428
SNLI, 68
Social Bias, 229
Social Group Substitutions, 236
Social Signal Processing, 377
SpanBERT, 64
Sparkles, 400
Special Tokens, 62
Speech Recognition, 377
SQuAD, 67
SqueezeLLM, 159, 160
Stable Diffusion, 411
State Space Models, 428
Statistical Filtering, 46
StereoSet, 236
Stereotypical Associations, 237
Storage Costs, 339
STS Benchmark, 280
Summarization, 110, 151, 222
SuperGLUE, 67, 112
Superlinked, 280
Supervised Fine-tuning, 214
Supervised Learning, 85–86
Support Vector Machines, 8
Swapped Language Modeling, 52–53
SwiGLU Activation, 41
Sycophancy, 221
Synthetic Data, 424

- T5, 11, 64–68, 108, 285
- Task Adaptive Pre-training, 49
- Task-Agnostic Metrics, 322
- Task-Specific Metrics, 322
- Teknium1, 424
- Temperature, 59, 347
- Tensors, 61
- Text Chunking, 279
- Text Classification, 96, 108
- Text Generation, 96
- TimeIT, 399
- TinyTransformer, 410
- Tokenization, 47
- Top-k Sampling, 60, 210
- Total Cost of Ownership, 333–335, 340
- Toxicity, 247–255
- Toxicity Probability, 250
- Transformers, 33–43
- Translation Language Modeling, 53
- Tree-of-Thoughts, 167–168
- Trigger Tokens, 106
- TruLens, 300, 350, 363
- Trust Region Policy Optimization, 195
- TrustGPT, 238
- Truthfulness, 228
- TruthfulQA, 181, 203, 221, 225, 226
- ULIP, 379
- ULIP-2, 379
- ULMFiT, 43
- UltraFeedback, 424
- Vanishing Gradients, 31
- Vector Database, 306, 346–347
- Vector Index, 287
- Vector Search, 347
- VeRA, 151, 155–157
- Version Control, 356–357
- Vicuna, 424, 429
- Video-Bench, 399
- Video-LLaMA, 401, 405–407
- Video-Subtitle Matching, 389–390
- Video-Text Modality, 384
- Video-Text Retrieval, 376
- VilBERT, 382
- Vim, 320
- ViP-Bench, 399
- Visual Basic Code, 320
- Visual Question Answering, 376
- ViT, Vision Transformer, 378
- Vocabulary Shifting, 252
- VoyageAI, 283
- Wasserstein Distance, 237
- Whisper, 379
- White-box Probing, 258
- WinoBias, 237, 244
- Winogender, 237
- Winograd Schema Challenge, 237
- Word Embedding Association Test, 234
- Word Embeddings, 9
- Word Filtering, 252
- word2vec, 9
- Workflow Orchestration, 354
- You.com, 434
- Zero-Shot, 48, 89, 135
- Zero-Shot Prompts, 163
- ZeroGPU, 430
- ZeroQuant, 159
- Zeroscope, 411