- **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:

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

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

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

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:

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

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:

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

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

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

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.

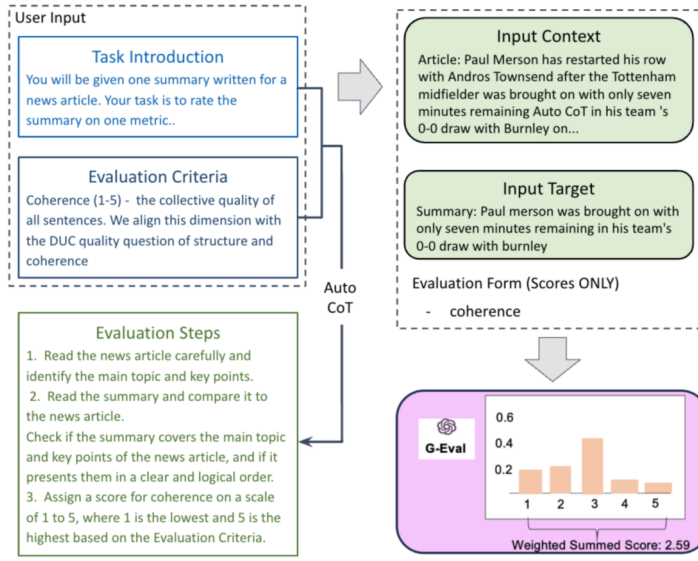

Fig. 8.1: The G-EVAL framework process. Initially, the Task Introduction and Evaluation Criteria are provided to the LLM, which then generates a Chain-of-Thoughts (CoT) outlining detailed evaluation steps. Subsequently, this CoT, along with the initial prompt, is used to assess the NLG outputs using 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:

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

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:

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

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 CodeContests 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.

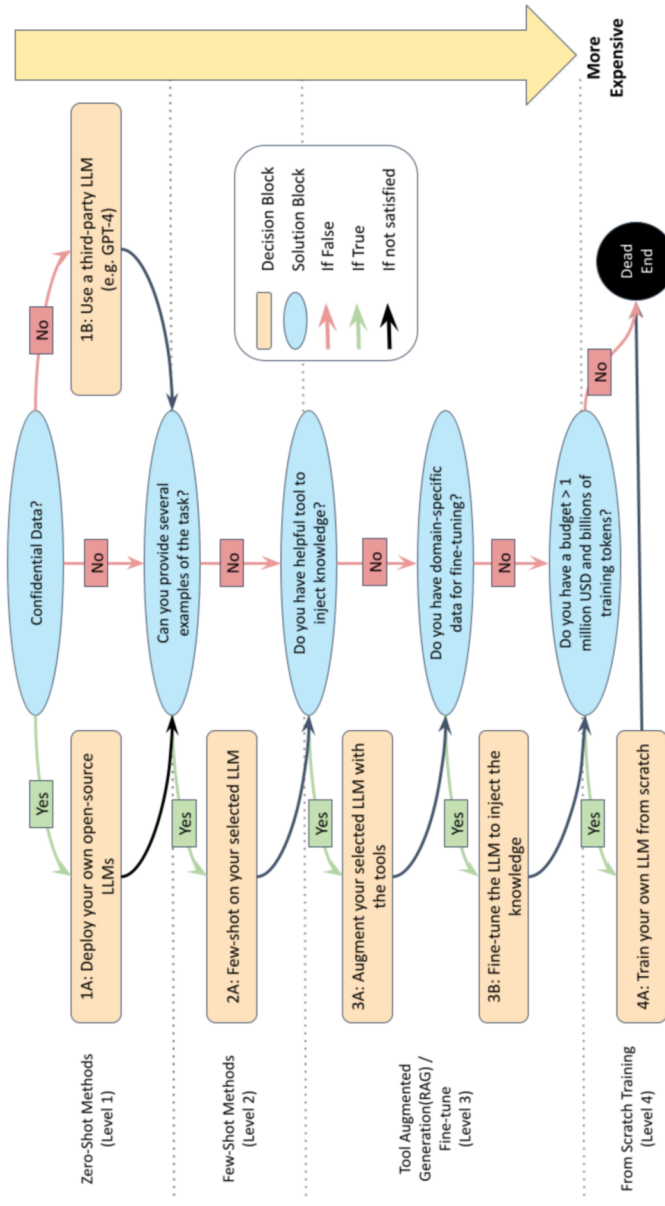

Fig. 8.2: A decision flow diagram for selecting between open-source and closed-source LLM and which customization path to follow within the financial services domain.

A good starting point for developers wishing to understand these factors and the process better is available on HuggingFace[3], which includes an interactive TCO calculator[4]. 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 transparency in the attributes from the row above. This has implications for usage in various settings, including debugging, maintenance and support of the application.	End-user access is typically through model weights. Developers are responsible 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 usage costs for open-source models. However, this higher price-point is often balanced by criteria such as ease-of-use and usage support	Usage costs for open source models can be significantly lower than closed-source models. This is in part thanks to 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
Customization	Limited to prescriptive approaches. These often come with higher cost than open source options	Customization is fully flexible through adaptive pre-training, full-parameter fine-tuning, parameter-efficient fine-tuning, and prompt-based learning.
Usage	Typically usage is through a limited set of prescriptive mechanisms such as a graphical user interface or an API	Usage is fully flexible and can be designed and 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 resourcing. 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 complex and costly responsibility for small development teams to own.
Privacy	Any data sent as input to the model is subject to the end-user agreement. This may mean the provider has permission 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 or preferences

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 and wrapper functionality delivered within their application	The model is often available with no end-user support. However, an active collaborative open-source community may be available to assist end-users with issues through Github or Huggingface projects.
Updates and Maintenence	The model owner schedules and releases model updates. These may happen without transparency and in accordance with the model owner's commercial road-map, which may not be desired 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

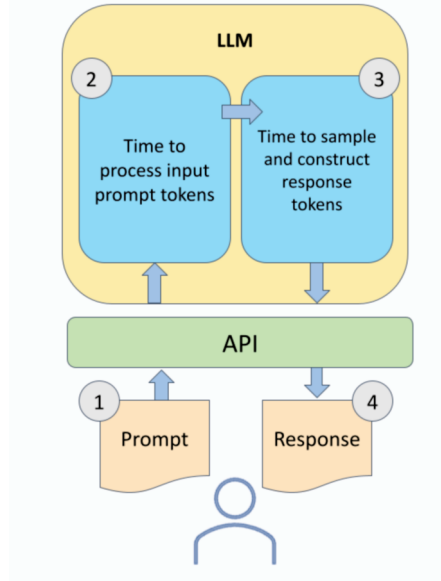

Fig. 8.3 LLM latency is the combination of 1) the time it takes for the user's prompt to reach the LLM's inference API, 2) the time it takes for the LLM to process the user's prompt, 3) the time it takes for the LLM to sample the relevant tokens and compose its response, and 4) the time it takes for 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. 8.2[5] 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

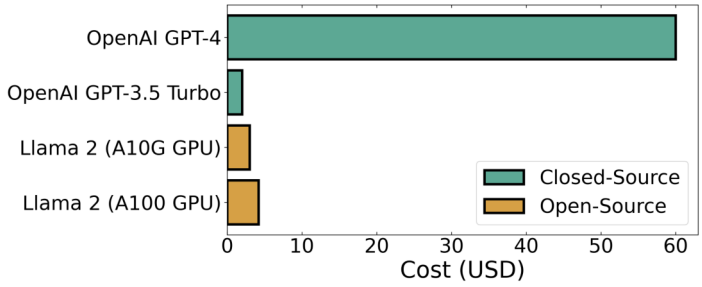

Fig. 8.4: USD cost of generating 1 million tokens. Comparison between two closed-source LLMs, OpenAI's GPT-4 and GPT-3.5 Turbo models, and one open-source LLM, Llama-2-7B parameter model running on two different GPUs, the A100 and the A10G. Although both Llama-2 generations cost more than 10x less than 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. 8.4[6]. 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-llms-bfd646ae1f74.

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 issue[7].

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-llms

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 8.3[8], 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`

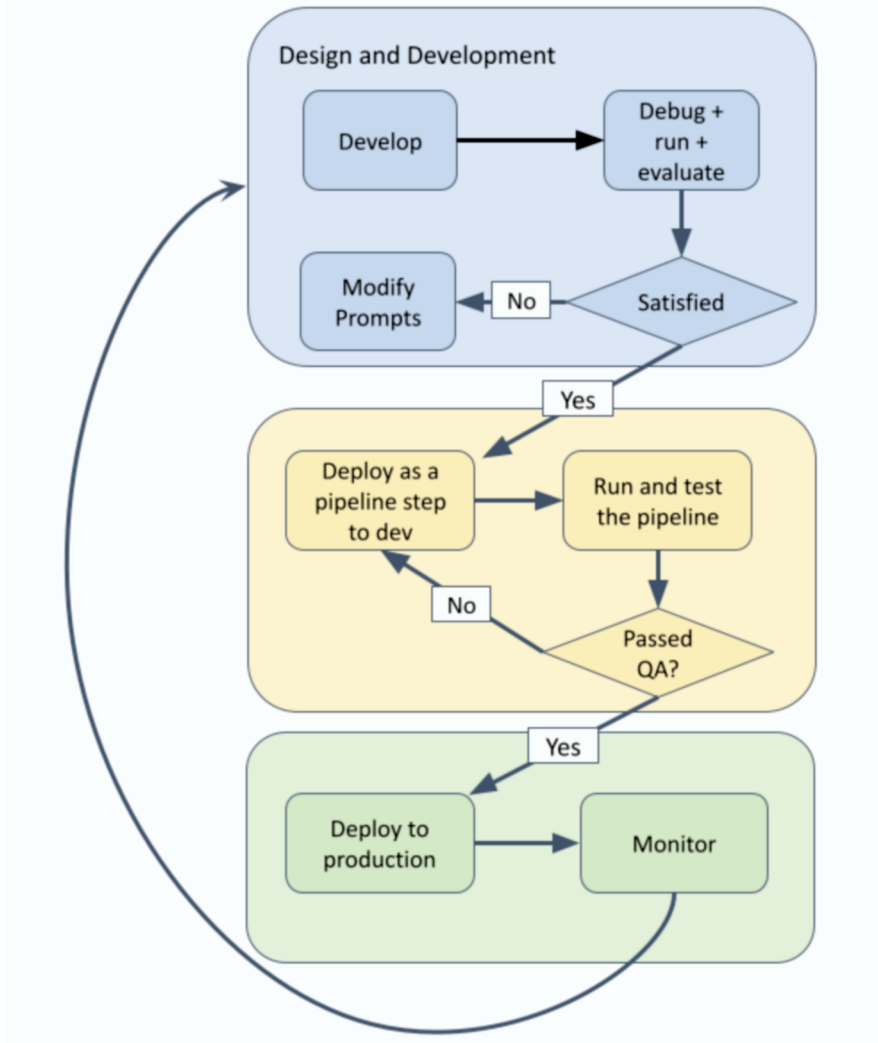

Fig. 8.5: Overview of the prompt engineering workflow.

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 dealbreaker. 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 server[16] 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

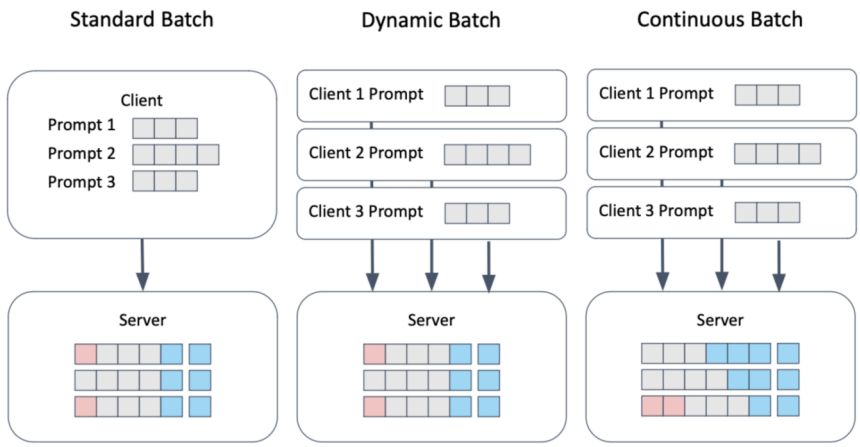

Fig. 8.6: In traditional inference architectures, it is largely up to the client to create batches. Particularly in applications where users send one request at a time, the GPU can be much more effectively utilized by dynamically aggregating multiple inputs on the server. This comes with a latency cost 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.

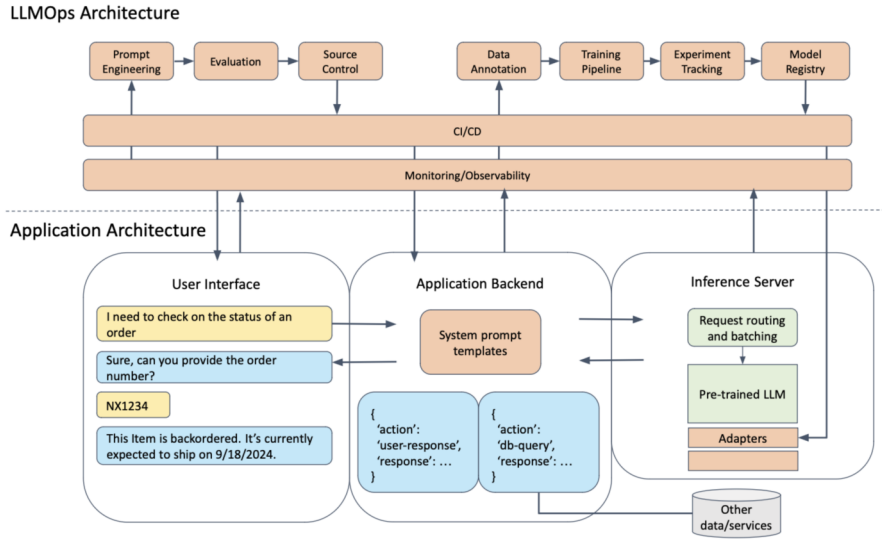

Fig. 8.7: A basic chatbot example with the corresponding LLMOps architecture. The prompt templates are developed through an iterative process and versioned in source control. The LLM in this case also uses adapters that have been trained for the specific needs of the application, thus requiring mechanisms 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['conciseness']} 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.

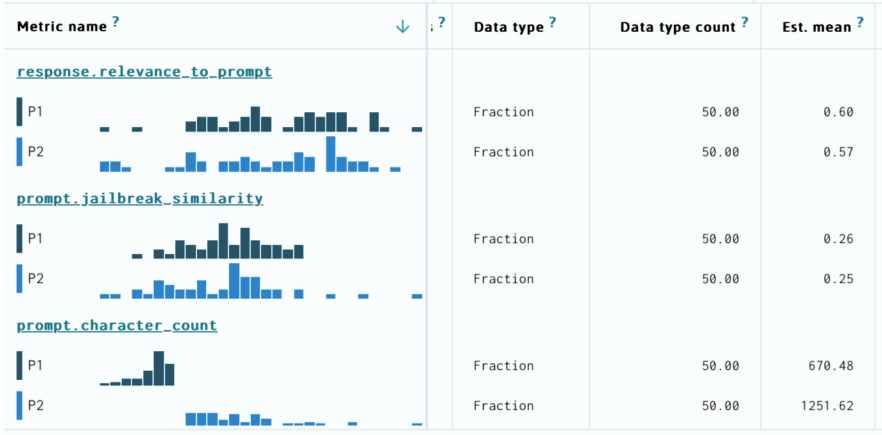

Fig. 8.8: A view of the WhyLabs monitoring dashboard, examining selected metrics to understand how they are impacted by simulated data drift on the prompts.

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 LLMOps 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 ty pologically 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, bidirectional 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. Biomistral: 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 Schrittwieser, 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, Kocmi 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, Raffaella 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 Bhattamishra, 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. Winogrande: 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. Socialiqa: 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 Herzig, 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 MMLLM 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 multifaceted 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

U. Kamath et al., *Large Language Models: A Deep Dive*,
https://doi.org/10.1007/978-3-031-65647-7_9

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.

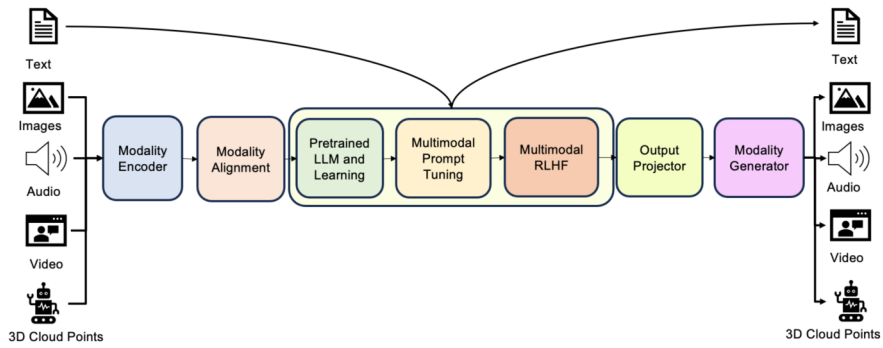

Fig. 9.1: The general framework of MMLLMs with different components providing implementation choices.

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:

$$\mathbf{F}_X = \mathrm{ME}_X(\mathbf{I}_X), \tag{9.1}$$

where $\mathbf{I}_{(X)}$ 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, $\text{IN\_ALIGN}_{X \to T}$, 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 $\{(\mathbf{I}_X, t)\}$, the primary objective is to minimize the loss associated with text generation conditioned on modality X, expressed as $\mathcal{L}_{\text{txt-gen}}$:

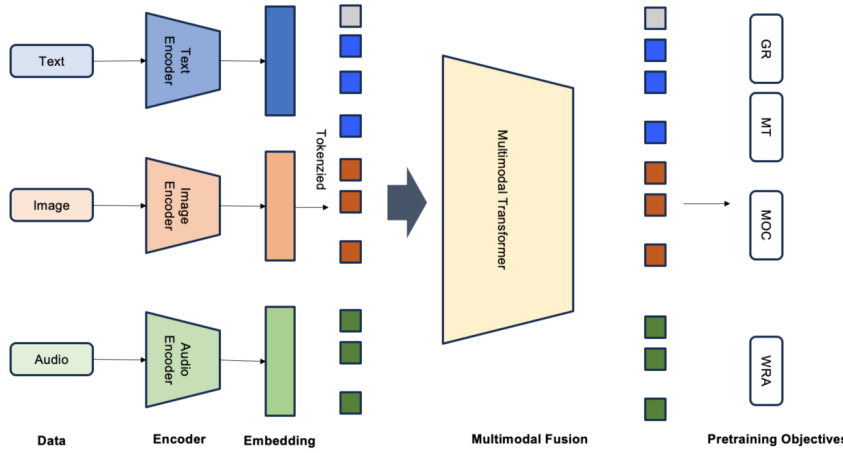

Fig. 9.2: The single-stream architecture

$$\underset{\text{IN\_ALIGN}_{X\to T}}{\arg\min} \; \mathcal{L}_{\text{txt-gen}}\left(\text{LLM}(\mathbf{P}_X, \mathbf{F}_T), t\right) \tag{9.2}$$

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

$$\mathbf{P}_X = \text{IN\_ALIGN}_{X\to T}(\mathbf{F}_X) \tag{9.3}$$

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-

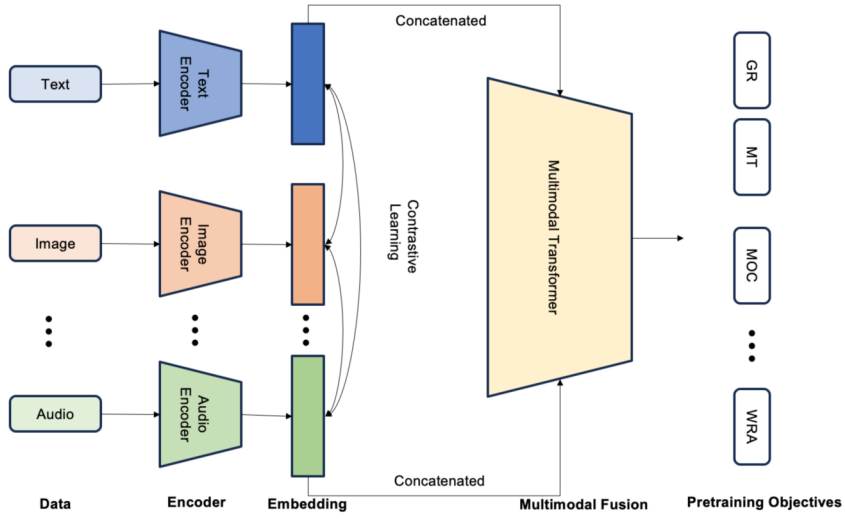

Fig. 9.3: The cross-stream architecture

modal features instead of creating fused multimodal features. Notable examples of large-scale MMLLMs 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:

$$Z(A) \leftarrow MHSA(Q_B, K_A, V_A),$$
$$Z(B) \leftarrow MHSA(Q_A, K_B, V_B),$$
$$Z \leftarrow Tf(C(Z(A), Z(B))).$$

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 MMLLMs 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:

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

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

MMLLMs 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
MSRVTT	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 $V_1, V_2, \ldots, V_N$ represent the embeddings for the N images, and $L_1, L_2, \ldots, L_N$ 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.

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

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

$$\mathcal{L}_{CL} = \mathcal{L}_{i2t} + \mathcal{L}_{t2i}. \tag{9.7}$$

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:

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

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