

The Ultimate Guide to Using LLMs with Speech Recognition to Build Voice Apps



Index

Executive summary	4
The new era of LLM-powered apps is here	5
Part 1: Large Language Models (LLMs)	7
Market overview	7
Types of LLMs and what they're good for	11
Open-source vs. proprietary LLMs for voice apps	16
Key LLM benchmarks and their limitations	21
Key challenges of LLMs and how to mitigate them	23
1. Hallucinations	23
2. Limited context windows	24
3. Predominantly English-language training data	26
How to improve LLM's performance: techniques and best practices	27
1. Prompt engineering	27
2. Fine-tuning	29
3. Retrieval-augmented generation (RAG)	30
4. Function calling	33
Part 2: Speech to Text	37
A brief introduction to ASR systems	38
Open Source vs. API for voice apps	40
1. Building in-house with an open-source model	40
2. Opting for Big Tech or specialized provider	42
Key factors to consider when picking a STT provider	44
1. Latency	45
2. Features	46
3. Accuracy	49
4. Language support	50
5. Security and regulatory compliance	52
6. Hosting in the cloud, on-premise, or air gap	52

Checklist: Planning and evaluating STT and LLMs for your voice app	56
Part 3: 5 best practices for using STT and LLMs for voice apps	59
Practice #1: Use LLMs to improve STT output and diarization	59
Practice #2: Divide and conquer with a multi-model approach	60
Practice #3: Don't resort to fine-tuning too early	61
Practice #4: Be mindful of context window size	62
Practice #5: Don't forget context windows' language bias	63
Conclusion	65
About Gladia	66

Executive summary

Large Language Models (LLMs) and **Speech to Text (STT)** have much to offer to product builders, but making full use of them – both individually and in tandem – is not as simple as it may seem.

This guide results from our work with hundreds of audio-first companies, including **AI-powered meeting note-takers and contact centers** as well as extensive interviews with some of our customers such as Circleback and Spoke.

We took into account key considerations when building voice-powered platforms to provide you with a comprehensive overview of the state-of-the-art LLMs and STT models, and **how to leverage them together to build advanced AI features for your voice app.**

The guide is jam-packed with **insights and hands-on advice** so you can confidently start implementing these cutting-edge capabilities today.

If you're a CTO, CPO, engineer, or developer looking to harness the full potential of STT and LLMs, you're in the right place.

The new era of LLM-powered apps is here

Previously reserved for organizations with extensive resources and experts with deep technical knowledge, the introduction of the first **generative-pretrained transformer (GPT)** in 2018 by OpenAI has taken the world by storm.

The widespread adoption of LLMs has lowered entry barriers in terms of resources and AI expertise and opened the door for businesses to build advanced AI-driven products and services.

It's estimated that there will be **over 750 million apps using LLMs by 2025**, and companies are facing pressure to integrate AI-powered features into their products to remain competitive and retain users.

Today, voice-first businesses like note-taking assistants and contact centers are increasingly **combining STT models with LLMs to embed new AI features for a range of use cases**: from generating meeting summaries in the blink of an eye, to assisting call center agents with real-time guidance.

It's clear LLMs and STT models have a lot to offer, **but making full use of them is not as simple as it may seem.**

Should you leverage open-source models or access them via an API? Is it cost-effective to use proprietary models or will an open-source model do for the task at hand? What are the different techniques you can employ to overcome common issues like hallucinations?

This guide will equip you with the knowledge to make an informed decision that is aligned with your user needs, budget, and tech stack. It's divided into three chapters that cover the most essential concepts about LLMs, STT models, and how to make the best use of these complementary technologies to build competitive products.

Let's start with LLMs.

Part 1

Large Language Models (LLMs)


Market overview







LLMs have come a long way: from simple rule-based systems to complex intelligence models that excel in natural language processing (NLP) tasks such as text generation, automatic translation, sentiment analysis, and document summarization.






The global LLM market is projected to grow from \$1.590 million in 2023 to \$259.8 million in 2030. During the 2023-2030 period, the CAGR is predicted to be at 79.80%.



There are hundreds of models currently available on the market. Listing them all would be nearly impossible, and the list would probably soon be outdated because of how quickly they are being developed and released.

Based on our internal customer survey, here are some of **the most popular LLMs used by voice platforms** today.

Issued by	Model example	Known for	Use cases	Pricing
 OpenAI	GPT models such as ChatGPT and InstructGPT.	Powerful text generation.	Chatbots, content creation, and code generation.	Multiple models with different price points.

Issued by	Model example	Known for	Use cases	Pricing
 Meta AI	LLaMa family	LLaMa 2 has been praised for its efficiency and ease of deployment on consumer-grade hardware.	Widely used in research and commercial applications, including chatbots and content creation.	The quota applies to each region where the models are available. The quota is specified in queries per minute (QPM).
 Google	PaLM	Released in 2022, PaLM leverages a Mixture of Experts (MoE) architecture to train models with up to 540 billion parameters.	Excels in various NLP tasks, including translation, summarization, and question answering.	It can be used for free.
 Google	Gemini	Designed to handle more complex tasks with improved contextual understanding and generative capabilities.	Complex tasks like coding and creative writing.	Free tier and flexible pricing as you scale.
 OpenAI	GPT models such as ChatGPT and InstructGPT.	Powerful text generation.	Chatbots, content creation, and code generation.	Multiple models with different price points.
 Anthropic	Claude	Extensive safety measures and fine-tuning to avoid generating harmful or biased content.	Sensitive data handling, educational tools, and healthcare support.	Free access with pro subscription starting at \$20 per month.
 Mistral	Ministral 8B	Powerful edge model with extremely high performance/price ratio.	Privacy-first inference for applications such as on-device translation, internet-less smart assistants, and autonomous robotics.	Two types of models: free and premier.

Issued by	Model example	Known for	Use cases	Pricing
 NVIDIA and Mistral AI	Mistral-NeMo-Minitron 8B	A miniaturized version of the recently released Mistral NeMo 12B model, delivering high accuracy combined with the compute efficiency to run the model across GPU-accelerated data centers, clouds, and workstations.	Chatbots, virtual assistants, and content generation.	Open-access model available on Hugging Face.
 Technology Innovation Institute in Abu Dhabi	Falcon	Released under the Apache 2.0 license, the Falcon Mamba 7B, Falcon 2, 180B, 40B, 7.5B, and 1.3B parameter AI models form a suite of offerings.	Some of the use cases include text classification and generation, sentiment analysis, and question answering.	Can vary depending on the specific model (e.g., Falcon 7B, Falcon 40B) and whether you are using it via cloud-based services or self-hosting.
 Databricks	Databricks' Dolly 2.0	Databricks' Dolly is an instruction-following large language model trained on the Databricks machine-learning platform.	Text classification, closed QA, generation, information extraction, open QA, and summarization.	It's an open-source model that you download, use, and modify.
 Stability AI	StableLM	The open-source StableLM family focuses on general NLP tasks.	Use cases include text generation, conversational AI applications, translation, and more.	StableLM models are primarily open-source and freely available for anyone to download, use, and modify. However, there are still costs involved with deploying and running them.
 Microsoft	Phi open models	A family of powerful, small language models with groundbreaking performance at low operation cost and low latency.	Potential use cases include real-time captioning for audio and video, on-device sentiment analysis, real-time language translation, and more.	Available for free for real-time deployment through the Azure AI model catalog, Hugging Face, and Ollama. Also available with pay-as-you-go billing via inference APIs.

Issued by	Model example	Known for	Use cases	Pricing
 Alibaba Group	Qwen	The powerful base models in the Qwen series are pre-trained on massive multilingual and multimodal datasets.	High-quality text creation and processing, coding assistance, and multilingual translation.	The pricing depends on the model. The fees are calculated based on the number of API calls.
 Google	Gemma open models	Gemma is a family of lightweight, state-of-the-art open models built from the same research and technology used to create the Gemini models.	CodeGemma and PaliGemma have their own specific use cases. But in general, Gemma can be used for tasks, such as building conversational AI assistants and chatbots, text generation, text summarization, and more.	The Gemma models' terms of use make them freely available for individual developers, researchers, and commercial users for access and redistribution.

Types of LLMs and what they're good for

LLMs owe their **exceptional performance to the Transformer architecture**, which fundamentally shapes how these models learn from training data.

Introduced in 2017, the Transformer enabled revolutionary mechanisms like positional encodings, attention mechanisms, and, most notably, self-attention.

This self-attention mechanism is what gives LLMs their edge, allowing them to produce coherent, context-aware outputs across diverse business cases.

By understanding relationships within data—be it text, customer behavior patterns, or online meeting transcripts—businesses can **unlock automation, deliver personalized experiences, and extract actionable insights from audio and beyond**.

Depending on the architecture, LLMs can be categorized by **decoder-only model, encoder-only, encoder-decoder-models**, and **mixture of expert (MoE) models**.

Each architecture has its unique strengths that can be further improved by techniques such as fine-tuning and prompt engineering, explained later in this chapter.

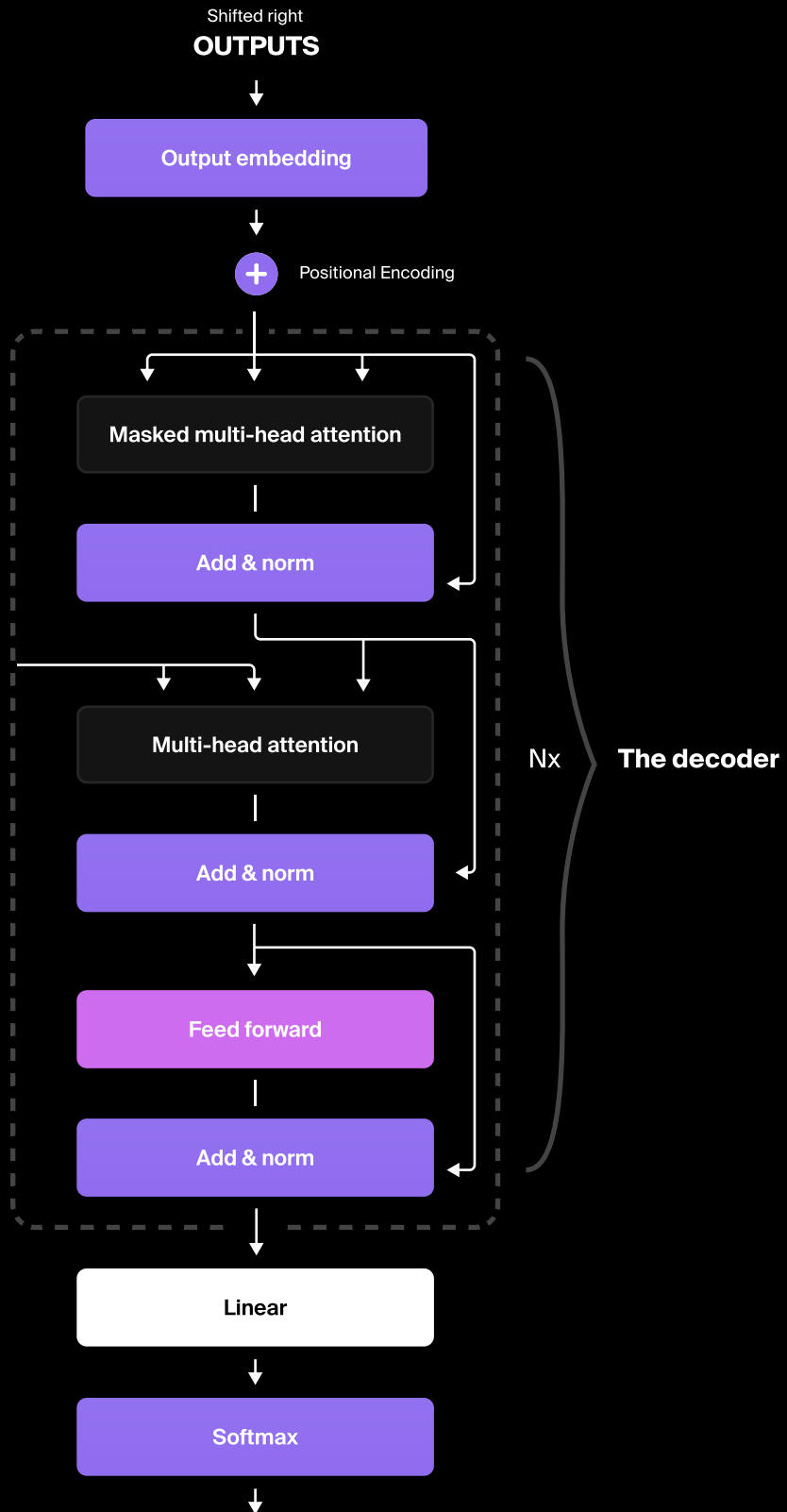
1. Decoder-only models

Models such as **OpenAI's GPT** series or **Meta's LLaMA**, use decoder-only architecture to generate the next part of the input sequence based on the previous context.

Good for

They can't comprehend the entire input but excel at generating the next probable word, which makes them effective for text-generation tasks like **creative writing** and **dialogue generation**.

GPT Decoder



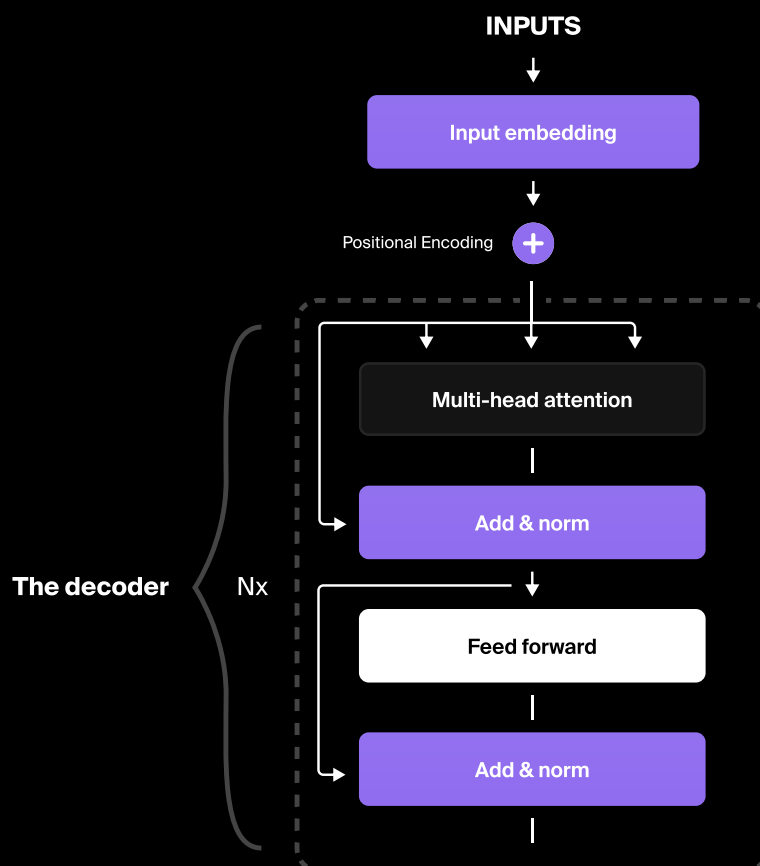
2. Encoder-only models

Models like **Google's BERT** leverage encoder-only architecture to turn input into contextualized representations without directly generating new sequences.

Unlike previous models that processed language in a unidirectional manner, BERT reads text bidirectionally, considering the context from both the left and right sides simultaneously.

✔ Good for

This bidirectional approach is crucial for tasks requiring a deep understanding of context and semantics, including **sentiment analysis**, **question answering**, and **named entity recognition (NER)**.

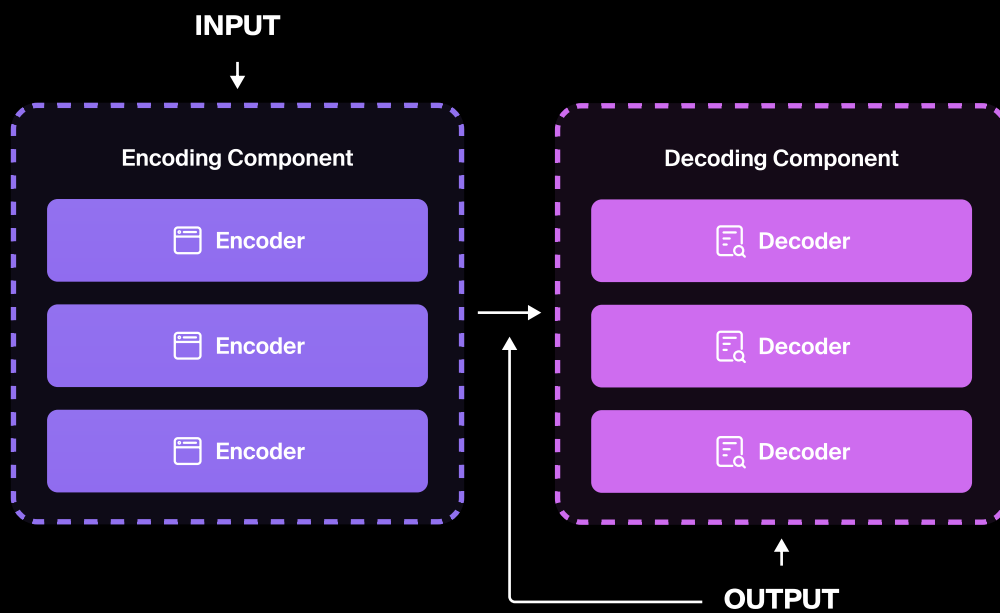


3. Encoder-decoder models

Models like **Google's T5** and **Meta's BART** consist of both encoder-decoder components. The encoder is responsible for processing the input sequence and the decoder generates the output sequence.

✓ **Good for**

The dual-process architecture excels in tasks where understanding the entire input before generating output is crucial. These include **translation**, **text summarization**, and **classification**.

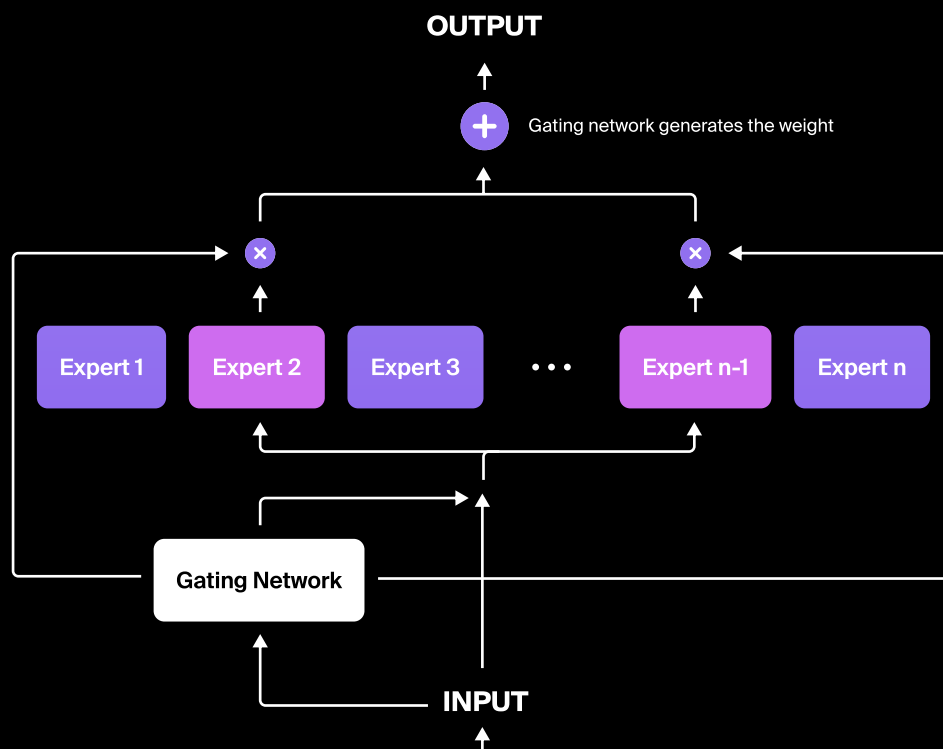


4. Mixture of expert (MoE) models

Models like [Mistral's 8x7B](#) and [Google's Switch Transformer](#) introduce a more modular and specialized framework for building LLMs that can dramatically increase the model size while maintaining computational efficiency. By dividing the model into multiple "expert" sub-networks, each of which specializes in handling different types of inputs or tasks, each model, or 'expert' undergoes a specialized training process. Over time, the gating network improves understanding of each model's strengths and fine-tunes its routing decisions accordingly.

✓ Good for

The MoE approach has led to notable advancements in fields such as **text generation**, **multilingual understanding**, and **domain-specific tasks**. For example, some experts might specialize in grammar and syntax, while others focus on specific domains like medical or legal language.



Open-source vs. proprietary LLMs for voice apps

Choosing the right model for your voice app goes beyond architecture—you'll also need to **decide between open-source and proprietary LLMs**. Key considerations include tradeoffs between **flexibility, security, and cost**.

Let's break down the pros and cons of each.

Open-source LLMs

Open-source models, typically created by developers and researchers, are available for anyone to use, modify, and distribute.

Flexibility

The main advantages of open-source models include transparency and flexibility; you can optimize them for more efficient performance and customize them depending on your specific needs.

Notable examples include:

1. **Falcon LLM** – excels at creative text generation and problem-solving.
2. **StarCoder (from Hugging Face)** – a highly effective coding assistant.
3. **LLaMA** – optimized for tasks like language translation.
4. **MistralAI** – offers versatile models available as pre-trained weights, designed for flexibility and adaptability across a range of applications, from text understanding to generative tasks.

Cost

Unlike proprietary models, **open-source LLMs don't charge licensing fees**, making them attractive to smaller organizations with tight budgets.

However, they come with **hidden costs**, such as infrastructure and setup, often requiring in-house expertise.

For example, while Mistral's models are free to use, **hosting and fine-tuning on cloud platforms can cost \$0.05–\$0.15 per 1,000 tokens**, depending on the provider and setup.

Similarly, Meta's LLaMA models are open-source, but deploying them for commercial use involves **significant cloud infrastructure expenses, especially for fine-tuning and scaling on platforms like AWS or Google Cloud**.

Security

Open-source LLMs allow for **greater control over data and model behavior**. You can deploy them on-premise or within a private cloud environment and prevent sensitive user data from being stored and potentially breached on a third-party server.

Proprietary LLMs

Proprietary models, developed by private organizations, are usually offered via APIs, commercial licenses, or subscriptions. Their access is restricted, with the underlying code and data kept private.

Companies like OpenAI, Microsoft, and AWS offer proprietary LLMs as a service. They provide public APIs to models such as OpenAI's GPT-4 or those hosted on Microsoft Azure, allowing for easy deployment and access.

Flexibility

Proprietary LLMs come with notable constraints compared to open-source options. Direct access to model weights or extensive fine-tuning is often off-limits, with customization usually limited to prompt engineering or API-specific tweaks. While this can feel restrictive, **it simplifies deployment and reduces operational complexity—a tradeoff many businesses are willing to make**.






That said, **these models shine in specialization. Whether fine-tuned for customer service, coding, or creative writing, proprietary LLMs are purpose-built for precision in specific use cases.** This focus can deliver exceptional performance but may limit versatility outside their domain.

Cost

Pricing typically follows a pay-as-you-go model calculated by token usage. Costs can escalate quickly with high-volume applications, but many providers offer free credits for trial runs—**ideal if you're a startup looking to test the waters.**

For enterprise users, discounts and **custom pricing can make scaling more cost-effective**, ensuring your ROI aligns with usage.

As of the time of writing, these are the prices for some popular proprietary models.

Provider	Model/version	Pricing estimate
 Cohere	Command R	\$0.01–\$0.04 per 1,000 tokens
 Azure OpenAI	GPT-3.5, GPT-4	\$0.03–\$0.06 per 1,000 tokens
 Anthropic	Claude 2, Claude 3	\$0.03–\$0.10 per 1,000 tokens
 OpenAI	GPT-3.5, GPT-4	\$0.002–\$0.12 per 1,000 tokens
 Google	Gemini 1, Gemini 1.5	\$0.03–\$0.12 per 1,000 tokens

Good to know:








- 1. OpenAI:** pricing depends on token usage and varies by model version, with fine-tuning costs determined by base model and usage.
- 2. Anthropic:** pricing is often not publicly listed, and estimates may vary.
- 3. Google Gemini:** pricing is mostly enterprise-focused and based on usage and capacity for models available on Google Cloud.
- 4. OpenAI Azure:** pricing varies by region, so it's essential to calculate expenses specific to your location.

Security

Proprietary LLMs can offer **enhanced security in areas like access control, data privacy, and regulatory compliance**. However, they are not immune to risks, including insider threats and transparency issues.

The security of a proprietary LLM depends on how it is developed, deployed, and maintained. **These models are often designed with strict data privacy measures, particularly for companies adhering to regulations like GDPR and HIPAA**. Strong access control features, such as user authentication and rate-limiting, are also common.

Below we compare the **pros and cons of open-source versus proprietary models** to help you make an informed decision.

Criteria	Open-sources models	Proprietary models
 Budget constraints	Generally free, making them ideal for projects with limited budgets.	Can be expensive, particularly for small businesses or individual developers.
 Customization	Offer the flexibility to modify and adapt models.	Limited scope for customization.
 Hallucinations	Can generate inconsistent or incorrect data.	Can generate inconsistent or incorrect data, but the commercial providers usually perform optimizations to improve these issues.
 Security	Risks include potential exposure of classified information or misuse.	More secure use of sensitive data and regularly updated to address vulnerabilities.
 Resources	Implementing and maintaining an open-source LLM can be resource-intensive.	API access enables fast and affordable implementation.
 In-house expertise	Low accessibility means substantial technical expertise is required.	High accessibility means little to no technical expertise is required.
 Reliability and support	Community-driven so often lacking dedicated support.	Controlled and reliable environment with dedicated support.

Key takeaway

When choosing the right LLM for your project, it's important to consider your needs, budget, tech stack, and in-house expertise. You don't have to choose either an open-source or a proprietary LLM; depending on your use case, a hybrid approach may be most effective.

If you want lots of customization, open-source models may be better suited. However, if you're looking to prioritize security, reliability or dedicated support, opting for a proprietary model is advisable.

Key LLM benchmarks and their limitations

For CTOs and engineering teams, **benchmarks are critical for assessing an LLM's performance** across tasks like coding, question-answering, common-sense reasoning, language translation, and text summarization.

These benchmarks evaluate models by assigning predefined tasks, measuring outcomes using specific metrics, and scoring results. These tasks typically include **coding challenges, math problems, and conversation samples**.

However, benchmarks often rely on static datasets that may not reflect real-world variability or evolving use cases, **making it crucial to supplement them with domain-specific testing**.

Testing happens with three techniques, which we'll explore in more detail later:

1. **Few-shot**: learns from a few examples.
2. **Zero-shot**: tackles tasks without examples.
3. **Fine-tuned**: trained on similar data for better results.

The model's output will then be compared to the expected answer and scored, usually from 0 to 100.

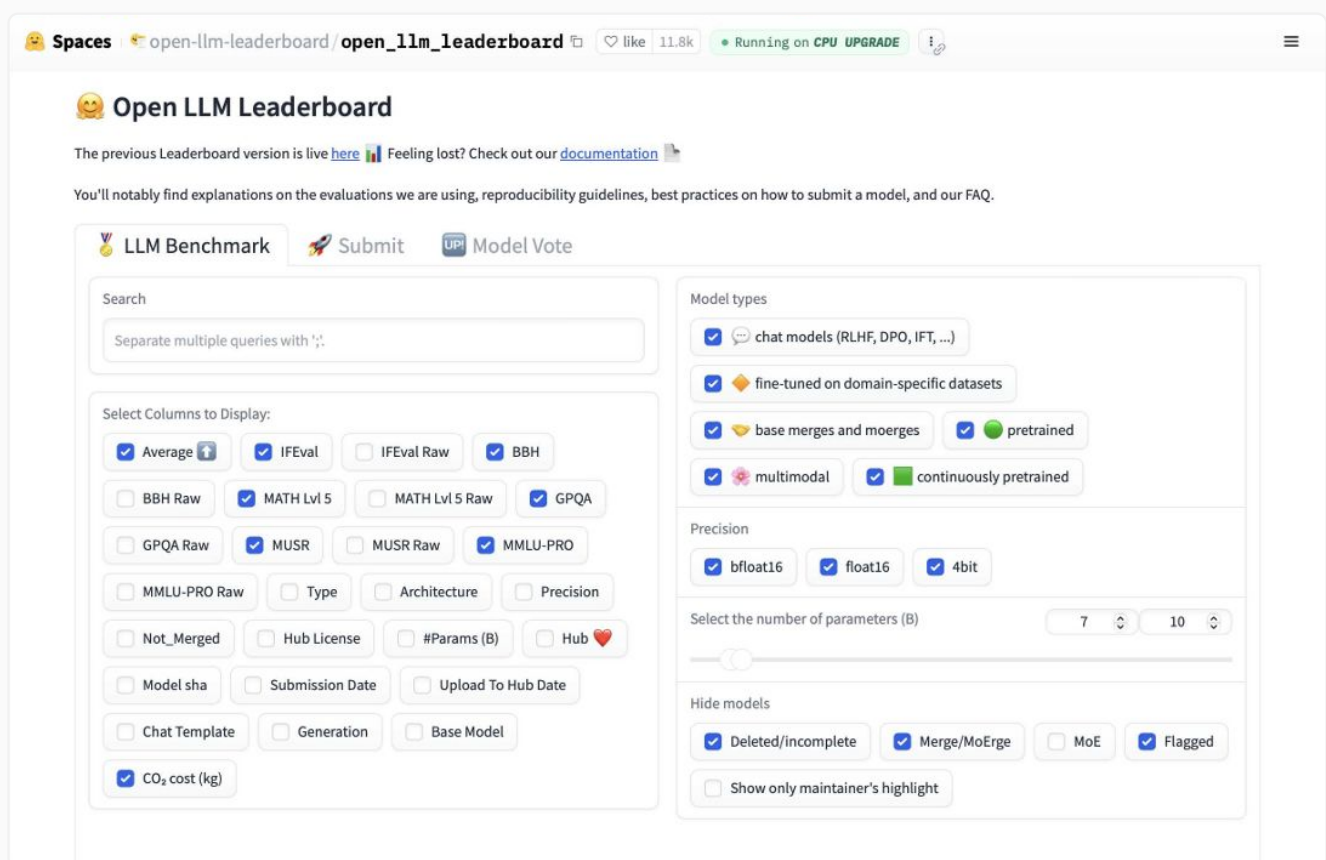
Here are the common metrics used for benchmarking LLMs:

1. **TruthfulQA** – Truthfulness
2. **MMLU** – Language understanding
3. **HellaSwag** – Commonsense reasoning
4. **BIG-Bench Hard** – Challenging reasoning tasks
5. **HumanEval** – Coding challenges
6. **CodeXGLUE** – Programming tasks
7. **Chatbot Arena** – Human-ranked ELO-based benchmark

Popular tools that **compare different models and their performance** for specific tasks are known as **LLM leaderboards**.

An example is the [Hugging Face Open LLM Leaderboard](#), an automated evaluation system that evaluates models across six tasks including reasoning and general knowledge. Other leaderboards that you can also use to evaluate a model include [MTEB](#) and [LLM-Perf](#) leaderboards.

Open LLM Leaderboard by Hugging Face



Benchmarks can help you identify the model's strengths and weaknesses, helping you through the process of optimizing a model's performance with techniques like **fine-tuning** or **retrieval-augmented generation (RAG)**, which we discuss below.

However, you need to take evaluations with a grain of salt. **Benchmarks have limitations. They don't always do a good job predicting how a model will perform in real-world situations** and can sometimes lead to what is called 'overfitting' – meaning they perform well on tests but fail during practical use.

Based on our experience and that of our customers, **benchmarks do a poor job of capturing how the average person interacts with the models being tested.** Our internal findings were aligned with other analyses, including [Surge AI's research about HellaSwag](#), which concluded that **36% of this popular benchmark contains errors.**

In addition, LLM benchmarks use diverse data and **may not assess performance well in specialized or unique cases that require tailored data.**

● Key performance metrics for speech recognition providers, including speed and accuracy, are covered later in a dedicated section.

Key challenges of LLMs and how to mitigate them

Despite their remarkable capabilities, LLMs come with their own set of challenges that may negatively impact your product and user experience. Below are some you should anticipate, and techniques you can use to overcome them.

Hallucinations


Hallucination usually occurs when the model **lacks knowledge** or **doesn't have enough context** on the topic. Although models can produce outputs tailored to a request, they can only reference information that existed at the time of their training, and that may not be up-to-date. The model will create coherent responses by filling in gaps with **information that sounds plausible but is incorrect.**

The extent of hallucinations can vary from minor inaccuracies to entirely fictional statements, often delivered with high confidence.

There are several types of hallucinations in LLMs, such as:

1. **Factual incorrectness:** involves the misrepresentation of existing data, such as providing the wrong medical reading, which can be harmful in critical fields like healthcare.
2. **Misinterpretation:** occurs when the model either misunderstands the user's input or misclassifies information from its knowledge base.
3. **Fabrications:** the model generates entirely fictional content. These kinds of hallucinations can have serious societal consequences, such as spreading misinformation, creating legal risks, undermining public health, eroding trust in AI systems, and amplifying biases.

There are different strategies that you can employ to address hallucinations in LLMs, including training the model on a diverse, expansive dataset, balancing prompts and data, and leveraging advanced techniques such as retrieval-augmented generation (RAG).

 For additional insights, refer to the “How to improve LLM’s performance: techniques and best practices” section next.







Limited context windows

LLMs generate predictions based on the overall context, not just the preceding word—this is enabled by the context window, a key factor in their power and versatility.

A context window defines the amount of input text the model can process at once. Larger windows allow the model to analyze more information, leading to richer, more accurate responses.

However, **most LLMs have limits on context window size due to RAM constraints.** For example, transcribing an hour of audio can produce around 25,000 tokens—challenging for many models. This issue is amplified in low-resource languages, where more tokens may be required to represent a single word.

Here's a breakdown of context window limits for some leading LLMs at the time of writing.

Model	Context window size	Notes
 GPT-3.5	4,096	Limited to a few thousand tokens and suitable for shorter interactions.
 LLaMa 2	4,096	Smaller context window, focusing on efficiency.
 Mistral 7B	32,800	Allows for detailed analysis and response to large text sequences.
 GPT-4o	128,000	Suitable for long-form content.
 Claude 3	~200,000	Large context window, ideal for long-form documents
 Mistral 7B	2,000,000	One of the largest context window sizes, suitable for handling long-form documents.

A key challenge with larger context windows is the **"lost in the middle"** or **"needle in a haystack"** problem. Traditional LLMs often lose focus on crucial information in the middle of a conversation, **prioritizing the beginning and end instead**. This imbalance in attention leads to **gaps in understanding and precision**.

Another issue is **"catastrophic forgetting"**, where models retain details from the start and finish of a document but struggle to recall the middle content accurately. These limitations can significantly impact performance in tasks requiring deep comprehension of longer inputs.

💡 There are different techniques to tackle these issues, including zero-shot and few-shot prompting. More information on these techniques can be found in the next chapter.

Predominantly English-language training data

The limitations of multilingual capabilities in LLMs pose significant challenges for app developers and product leaders operating in a global market. Language barriers can **disrupt user experiences, leading to lost engagement or revenue opportunities for your business.**

Since the majority of LLMs are trained on English-language data, they often fail to understand or generate nuanced responses in other languages. Low-resource languages are particularly affected, experiencing **higher rates of hallucinations** and inaccuracies due to insufficient representation in training data. This can lead to misaligned user expectations and reduced trust in AI-driven features, especially in markets where precise localization is critical.

While multilingual models like **mBERT** and **XLM-R** offer improved cross-lingual performance compared to English-dominant models, **they still struggle with capturing cultural context and linguistic subtleties**, which can hinder seamless global scalability for apps.

Key takeaway

If you're a CPO or CTO building apps for diverse audiences, you should prioritize integrating models that are fine-tuned for target languages or consider a hybrid approach, such as combining LLMs with traditional localization frameworks.

- 1. Diverse multilingual datasets:** Training LLMs on large, diverse datasets that include underrepresented languages helps the model learn cross-lingual representations. For example, models like mT5 (multilingual T5) excel in multilingual tasks by leveraging such data.
- 2. Task-specific and cross-lingual fine-tuning:** Fine-tuning on specific tasks across multiple languages simultaneously enhances performance for both high- and low-resource languages, improving the model's contextual understanding and accuracy.

How to improve LLM's performance: techniques and best practices

In this section, we'll cover some key techniques and best practices you can apply to improve the performance of an LLM, namely: prompt engineering (zero-shot, few-shot chain-of-thought), fine-tuning, and retrieval-augmented generation (RAG).

Prompt engineering

Prompts are the input that guides an LLM's output and task execution. Different types of prompts—zero-shot, few-shot, and chain-of-thought (CoT)—**allow for tailored model behavior and influence output quality based on task complexity.**

Zero-shot prompting

Zero-shot prompts rely on the model's pre-trained knowledge without providing specific examples. This approach works well for **straightforward tasks**, for example:

Prompt	Output
Complete the sentence by filling in the [mask]: Sentence: The weather today is [mask].	Sunny

Few-shot prompting

Few-shot prompts include one or more examples to guide the model, **improving performance for complex tasks.** When a single example is used, it's called one-shot prompting, for example:

Prompt	Output
I love this product! // Positive The service was terrible! // Negative What an amazing experience! // Positive That restaurant was awful! //	Negative

Chain-of-thought (CoT)

CoT prompts guide the model to break tasks into **intermediate reasoning steps**, **enhancing performance in problem-solving and multi-step calculations**. While most effective with few-shot prompts, zero-shot CoT can also be applied to encourage step-by-step reasoning, for example:

Prompt (with CoT)

A farmer has a total of 80 apples. He wants to divide them into baskets such that each basket has an equal number of apples. If he places 12 apples in each basket, he will have 4 apples left over. How many baskets can he fill? Let's think step by step.

Output

First, we know the farmer has 80 apples in total. If he puts 12 apples in each basket and has 4 apples left over, we can subtract the leftover apples from the total:

$80 \text{ apples} - 4 \text{ apples} = 76 \text{ apples}.$

Now, we divide the remaining apples by the number of apples per basket:

$76 \text{ apples} / 12 \text{ apples per basket} = 6 \text{ baskets}$ (with a remainder). So, the farmer can fill 6 baskets completely with 12 apples each.

✓ Good to know

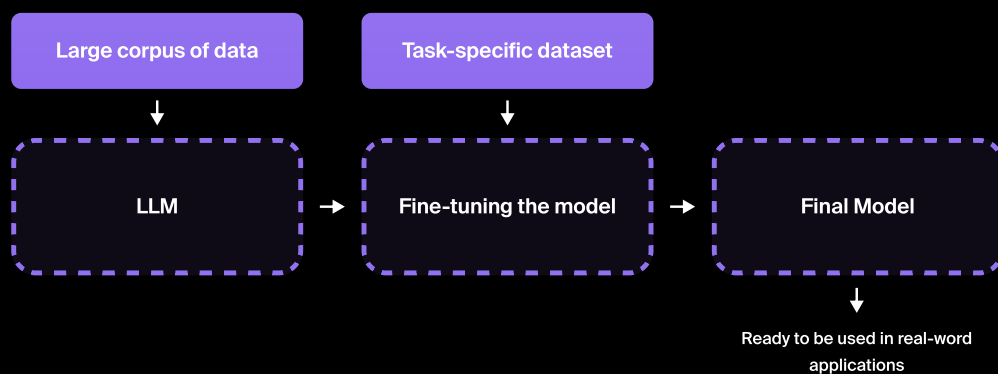
Whatever the technique, remember to **match the size of the prompt to the data** provided. For example, pairing a one-sentence data input with a 50-sentence prompt can cause the model to overemphasize the prompt structure instead of the data content. Ensuring this proportionality is key to avoiding hallucinations.

Fine-tuning

Fine-tuning lets you **take a pre-trained model and tailor it to your specific needs**. Instead of building a model from scratch using pre-training, you start with a model already skilled in general language understanding and **refine it with task-specific data**.

During fine-tuning, the model's architecture remains unchanged, but its **internal weights are adjusted** to better fit the new dataset or domain. For instance:

- 1. Medical applications:** Models like **Med-PaLM** are fine-tuned with medical data, including research papers and health queries, enabling them to handle specialized tasks in healthcare.
- 2. Programming:** **Code LLaMA** is optimized for coding, offering powerful features like autocompletion, debugging, and multi-language code translation.
- 3. Speech recognition:** Fine-tuned models enhance automatic speech recognition (ASR) systems like **Whisper**, helping them tackle domain-specific terminology and complex language structures in fields like healthcare or low-resource languages.



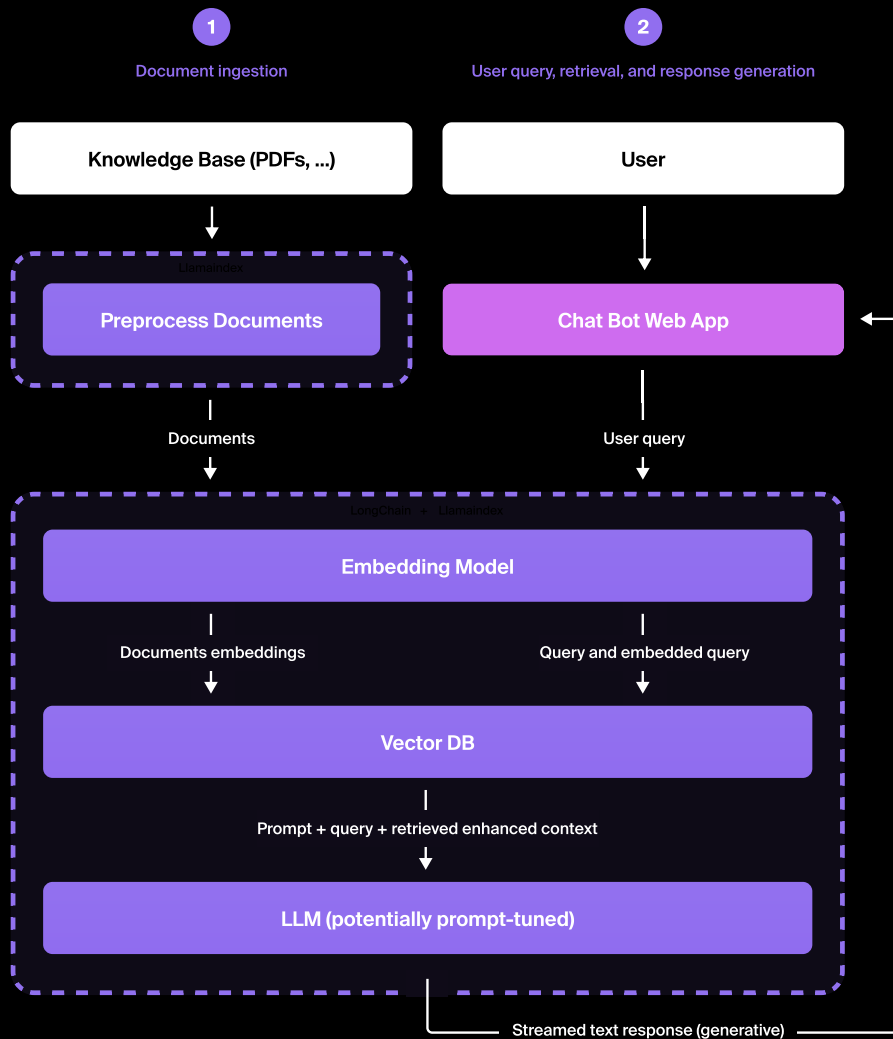
Retrieval-augmented generation (RAG)

RAG enhances LLM accuracy by integrating **real-time retrieval of external data into the prompt**. By accessing up-to-date information from sources like customer documentation, web pages, or third-party applications, **RAG enables LLMs to deliver highly accurate, context-aware responses**.

This approach ensures that your model remains **relevant and reliable**, no matter how dynamic or specialized your queries are.

Here is how the retrieval process works:

- 1 User prompt:** The user gives a specific query and triggers LLMs to create a response. RAG converts the query into vectorized representations called **embeddings**. Each element in an embedding corresponds to a specific property within the query's text that the model can understand.
- 2 Semantic search:** RAG then performs a similarity search using AI algorithms to match the query embeddings with the embeddings in a **vector database** that contains external knowledge. Vector databases store these embeddings in chunks. Each chunk contains a segment of data corresponding to a particular domain. Algorithms will compute similarity metrics to determine which chunk is closest to the query embeddings to understand the relevant context. Relevant embeddings will be fetched to provide the LLM with the correct context associated with the user's query.
- 3 Prompt:** LLM uses the context information retrieved from the vector database and the user's query as input. It combines this with the configured prompt, which provides the LLM with the necessary instructions on how to generate a response.
- 4 Post-processing:** LLM processes the input according to the prompt and provides a response.



The process of obtaining reliable external data through techniques such as **web scraping**, **API integration**, and **document indexing** allows organizations to ensure that the information being retrieved is both **current and accurate**.

✓ **Good to know**

RAG is among the most advanced techniques used to **mitigate hallucinations**. By retrieving relevant information from a trusted source in real time, it allows to significantly improve accuracy and avoid costly mistakes.

The difference between RAG and fine-tuning

Both RAG and fine-tuning are methods to enhance LLMs' output and increase its accuracy and relevance.

RAG enables you to inject real-time context into your prompts, tailored to your ingestion strategy, providing dynamic, up-to-date insights to a deployed LLM.

In contrast, **fine-tuning is confined to the static context and data present in the model's original training dataset**.

Below is a brief overview of the main differences between the two techniques.

	Fine-tuning	RAG
Adaptation	After the fine-tuning phase for a specific task, LLMs become static.	RAG is an evolving system that can learn from additional sources over time.
Data training	Fine-tuning re-trains the parameters of a model to optimize performance with new data for a specific task. Unlike RAG, however, this data needs to be prepped and cleaned so that it can be used for fine-tuning.	RAG adds information from external sources related to a specific topic, without changing the model's internal parameters.
Versatility	If a model hasn't been fine-tuned for a domain-specific task, it doesn't have sufficient knowledge to handle related queries. For example, if the model is fed with data consisting of legal employment contracts, it can only answer questions about work-related issues with legal consequences.	RAG can augment the LLM with any information source related to any domain without re-training the model on a new dataset and knowledge.
Catastrophic forgetting	Fine-tuning an LLM for a new task can lead to forgetting or losing previous knowledge learned during the pre-training phase.	Since RAG does not change the model's internal parameters, LLMs retain their pre-training knowledge.
Computational requirements	Fine-tuning a model requires extensive computational resources and the use of GPUs.	RAG-powered models can be resource-intensive.

Function calling

Function calling lets LLMs **connect to external tools**, making them much more versatile.

For instance, instead of guessing or relying on outdated information, the model **can fetch real-time data**—like today’s weather in Paris or the latest euro-to-dollar exchange rate—by tapping into the right APIs. This ensures users always get accurate, up-to-date answers.

Let's say a user is asking the following question to the model:

What is the euro-to-dollar exchange rate today?

To handle this request using function calling, the first step is to define a currency conversion function or set of functions that you will be passing as part of the OpenAI API request:

```
tools = [  
  {  
    "type": "function",  
    "function": {  
      "name": "get_currency_exchange_rate",  
      "description": "Get the current exchange rate between two currencies",  
      "parameters": {  
        "type": "object",  
        "properties": {  
          "from_currency": {  
            "type": "string",  
            "description": "The currency you want to convert from, e.g., 'EUR' for Euro"  
          },  
          "to_currency": {  
            "type": "string",  
            "description": "The currency you want to convert to, e.g., 'USD' for US Dollar"  
          }  
        },  
        "required": ["from_currency", "to_currency"]  
      }  
    }  
  }  
]
```

The `get_currency_exchange_rate` function returns the current exchange rate between the specified currencies. When you pass this function definition as part of the request, it doesn't actually execute a function but simply returns a JSON object containing the arguments needed to call the function.

You can define a completion function as follows:

```
def get_completion(messages, model="gpt-3.5-turbo-1106", temperature=0, max_tokens=300,
tools=None):
    response = openai.chat.completions.create(
        model=model,
        messages=messages,
        temperature=temperature,
        max_tokens=max_tokens,
        tools=tools
    )
    return response.choices[0].message
```

This is how you can compose the user question:

```
messages = [
    {
        "role": "user",
        "content": "What is the euro-to-dollar exchange rate today?"
    }
]
```

You can call the `get_completion` function above and pass both the messages and tools:

```
response = get_completion(messages, tools=tools)
```

The response object contains the following:

```
ChatCompletionMessage(content=None, role='assistant', function_call=None,
tool_calls=[ChatCompletionMessageToolCall(id='...',
function=Function(arguments='{"from_currency":"EUR","to_currency":"USD"}',
name='get_currency_exchange_rate'), type='function')])
```

The arguments object contains are the arguments extracted by the model that are required to complete the request.

You can then choose to call an external currency conversion API for the actual exchange rate. Once you have the exchange rate information available, you can pass it back to the model to summarize a final response based on the user question.

Key takeaway

Enhancing LLM performance comes down to choosing the right technique—or combination of techniques—for your specific goals. RAG, fine-tuning, prompt engineering, and function calling each offer unique benefits, and they're not mutually exclusive.

You might begin with RAG for real-time context and later fine-tune the model for a highly specialized task. In some cases, prompt engineering or function calling alone may meet your needs.

The key is to embrace an iterative approach of testing, learning, and refining to achieve the best results.

Transcription accuracy: The hidden key to LLM success

LLMs are transforming product development, delivering unparalleled advancements in performance through optimization techniques. But their capabilities are only as strong as the data they process. When working with spoken language, this means transcription accuracy is essential.

Errors in transcription can derail even the best LLMs, leading to outputs that miss the mark. In the next chapter, we'll help you choose the right STT model or provider to meet your specific needs and use case.

Circleback.

“For us at Circleback, the key box that a speech recognition provider needs to tick is transcription accuracy. If there are errors or misattributions, things can break on the LLM side and you'll end up with a bad output.”



Kevin Jacyna, Founder at Circleback

In the upcoming section, you'll learn:

- The pros and cons of building in-house vs working with APIs, depending on your growth phase
- Critical factors to evaluate when picking an STT provider, be it for async or real-time transcription
- Key hosting and security consideration to take into account

Part 2

Speech to Text

For a long time, the lack of fast, accurate, and low-cost transcription technology has been a major obstacle to anyone looking to embed audio-related features into their product.

But in parallel with LLM developments, speech-to-text (STT) models have been becoming increasingly robust.

Speech-to-text technology has become both advanced and affordable, paving the way for widespread adoption and unlocking opportunities to enhance productivity and streamline workflows. For example:

1. Real-time AI can enhance the efficiency of **contact center agents**.
2. AI-driven transcription and insights can elevate the effectiveness of **sales calls**.
3. LLM-powered AI assistants can deliver seamless, accurate **note-taking**.
4. **Media companies** can streamline editing and subtitle creation with time-stamped transcriptions.

To understand how STT models and APIs can be optimized for success, let's first briefly dive into the underlying mechanics that make state-of-the-art speech recognition possible.

A brief introduction to ASR systems

Speech to Text, also known as Automatic Speech Recognition (ASR), is a branch of **natural language processing (NLP)** that **uses algorithms to analyze audio data and convert spoken language into text.**

Historically, ASR relied on mechanical or rule-based systems, such as phoneme matching and statistical models like Hidden Markov Models (HMMs). Modern ASR models, however, utilize advanced machine learning techniques, particularly deep learning architectures like **RNNs, CNNs, and Transformer models.**

Trained on large datasets, these models are mostly based on the **seq2seq architecture** that maps the input speech signal directly to the text without dividing the recognition process into multiple steps.

This approach allows for **greater accuracy and better contextual understanding of language** based on the semantic proximity of each given word. A notable example of a seq2seq system is Whisper by OpenAI.



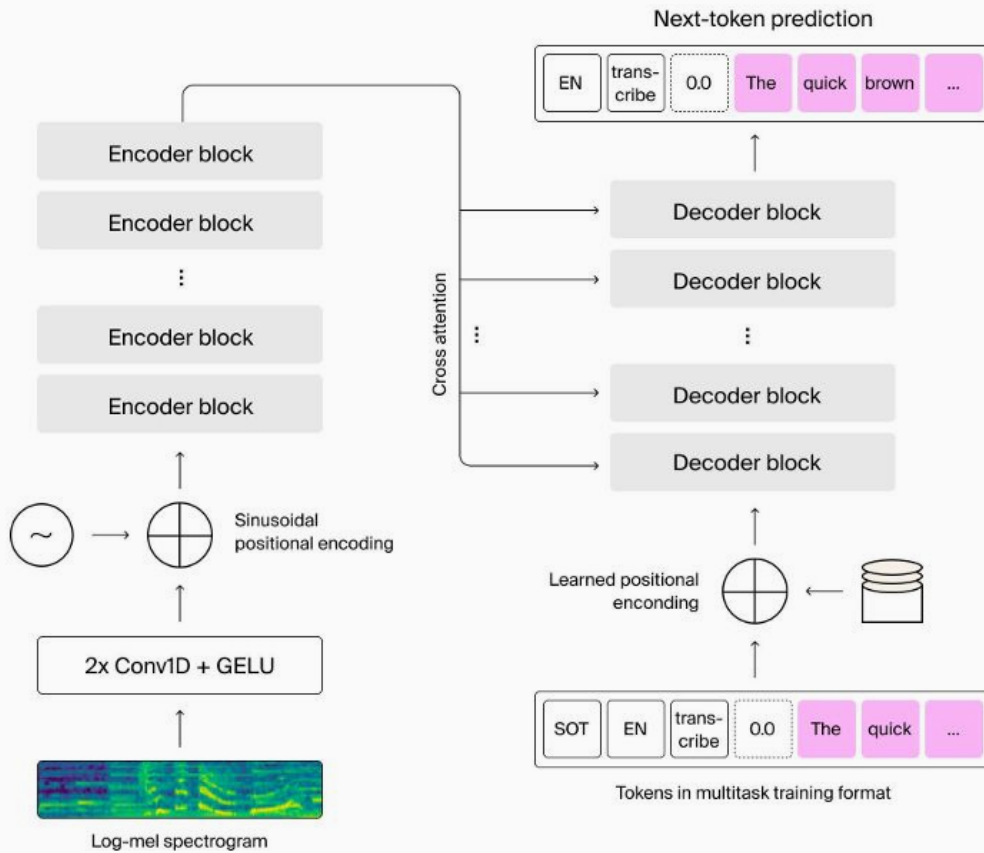
Zoom-in: Whisper ASR by OpenAI

Trained on 680,000 hours of multi-language data, Whisper became highly popular among indie developers and businesses alike. Here's why.

Transformer-based models like Whisper have a **generative component** that consists of an encoder-decoder framework and **enables them to create content based on the patterns learned from training datasets.**

This architecture of ASR systems like Whisper enables the model to **infer the broader context of sentences transcribed and “fill in” the gaps in the transcript** based on this understanding. In addition, Whisper allows you to apply **prompt engineering** techniques to improve its performance.

Thanks to this, Whisper achieves **leading transcription accuracy** and **can detect and translate speech in over 99 languages.**



While the open-source Whisper model offers impressive capabilities, it has notable limitations that remain unresolved: **hallucinations, inconsistent accuracy in under-represented languages, and the lack of real-time transcription**—an essential feature for use cases like AI-powered agent assistance in call centers.

To address these challenges, some specialized providers – including **Gladia with [Whisper-Zero](#)** – have developed more robust and optimized ASR systems built on open-source models, **designed specifically for enterprise applications**.

This brings us to a critical decision for your team: should you host an in-house solution or outsource your speech-to-text needs to a commercial provider?

Let's explore this decision together.

Open Source vs API for voice apps

When it comes to implementing an ASR system into your product, you have a range of options, from building an in-house solution to turning to a BigTech or specialized provider.

Building in-house usually entails using an open-source model while using a model-as-a-service API means working with a commercial provider.

Each approach has its advantages and drawbacks, and the right choice will depend on your specific business goals and use case.

Building in-house with an open-source model

Like with LLMs, the open source ASR ecosystem has been on the rise. **Among the most popular open-source STT models today are [Whisper](#), [DeepSpeech](#), and [Wav2vec](#).**

While open source has been a massive driver in the proliferation of voice platforms and apps, adopting them for enterprise use cases is not always recommended.

Say you're thinking of hosting Whisper in-house. First, you need to take into account the **total cost of ownership (TCO) required to host, optimize, and maintain** it as your audio volumes scale and roadmap features multiply.

There are a number of costs contributing to the TCO, including hosting (CPUs/GPUs), network, security software, human capital, and certification.

As a result of all these expenses, the **TCO of Whisper can amount to anything from \$300k to \$2 million per year**, depending at what stage your business finds itself.

Even if you have the budget for it, you need to assess whether this is actually the best use of your resources. After all, transcription is not your core source of differentiator, and you need to treat it accordingly.



RCF Framework: Risk-cost-focus of self-hosting

Early Stage: Prototyping and Validation (<5k hours/month)

If you're in the early stages of finding product-market fit and transcribing **less than 5,000 hours of audio per month**, hosting Whisper in-house may make sense. At this volume, costs are manageable, and while the vanilla model lacks optimizations, features, and accuracy in certain scenarios, it's good enough for proof-of-concept work.

The trade-offs? Whisper's tendency to hallucinate, limited support for features like diarization, and formatting inconsistencies. However, these downsides are usually acceptable for low-stakes prototyping.

Growth Phase: Scaling Usage (5k–15k hours/month)

When transcription volumes rise to **5,000–15,000 hours per month**, the equation changes. Costs increase as you'll need full-time employees to:

- 1. Optimize the model** (e.g., adding features, improving accuracy, and mitigating hallucinations).
- 2. Maintain the infrastructure**, which becomes increasingly complex with scaling demands.
- 3. Implement features** like diarization, requiring ~20% additional compute power.

Parallel transcription requests will also surge, necessitating on-demand GPU availability—significantly more expensive than reserved instances—alongside a robust queuing system.

At this stage, **hosting in-house is rarely worth the effort**. Your resources are limited, and market pressures demand you focus on your platform's core differentiators, not on perfecting transcription infrastructure.

Scale-Up and Beyond: Enterprise Level (>15k hours/month)

For platforms transcribing over **15,000 hours per month**, transcription becomes core to your business operations. While you likely have the budget to host in-house—**easily exceeding \$2M annually**—consider whether it's the best strategy.

The pace of innovation in ASR technology is rapid. Maintaining an in-house team dedicated to keeping up with advancements may detract from delivering a reactive, competitive roadmap for your core product.

All in all, **outsourcing your transcription needs to specialized STT providers is a more viable option** today for scaling platforms, enabling faster time-to-market and wiser allocation of resources to the core features of your platform.

According to the Gartner Generative AI 2024 Planning Survey, **72% of functional leaders** are buying GenAI capabilities from either an existing or new vendor.

Opting for Big Tech or specialized provider

The commercial landscape for speech-to-text APIs today consists of the **big cloud** providers **Amazon Web Services (AWS)**, **Google Cloud Platform (GCP)**, and **Microsoft Azure**, the famous outlier **OpenAI**, and **specialized contenders** like **Gladia**, **Assembly AI**, **Deepgram**, and others.

Unlike open-source models, **these solutions come in the form of plug-and-play API, helping to significantly shorten your time-to-market.** Specialized providers work with best-in-class ASR models, which they maintain and optimize over time – which comes at a price, of course.

Big Tech's speech-to-text solutions in particular, provided as part of their wider suite of services, are usually the most expensive on the market and generally perform slower.



In terms of accuracy, **BigTech providers generally have a WER of 10%-18%**. Compare this to **most specialized providers that are within the 1-10% range**, and offer more affordable options with custom discounts for large scale customers.

 **Key takeaway**

Every speech-to-text provider comes with trade-offs you'll need to consider. Higher accuracy often means sacrificing speed, while enhanced features like speaker diarization, sentiment analysis, or advanced security certifications can further drive up costs and slow processing times. Balancing these factors is crucial to finding a solution that aligns with your needs, budget, and performance expectations. It is imperative to first examine your user base, vertical and roadmap, as this will help you select a vendor able to strike the right balance for your use case.

In the following section, we dive into the key factors to consider when evaluating an STT vendor, with a dedicated checklist at the end of this chapter.

To recap the OSS vs APIs debate, here's a summary table:

	Open-source	Commercial providers
 In-house expertise	Good option for companies with a strong in-house AI fleet, which includes a dedicated DevOps division.	Suitable for companies that lack hardware and/or AI expertise, but still want to embed advanced AI features into their product. Any developer can use an API without needing any additional AI expertise.
 Customization	You're in full control of your data and can make tweaks depending on your business needs.	The majority of API providers offer an extensive set of out-of-the-box features that you pick from, usually at additional cost.

	Open-source	Commercial providers
👤 Scalability	Since commercial APIs tend to be powered by several, optimized models, replicating the same with open-source models – which tend to be smaller and more narrow in their scope – can be challenging.	With APIs, you don't need to commit to a specific volume of audio data in advance. You can scale at a reasonable cost as you grow since the overall load is shared by multiple users.
💰 Cost	While the models themselves are often free to access, infrastructure, development and maintenance costs will certainly make their way into the bill.	The absence of hardware requirements is good news for your budget, however, the API market doesn't always strike the right quality-price balance for enterprise-grade clients, which can result in high costs for some use cases.
🕒 Time and resources	Production-ready models usually take one year to be ready for deployment. Given the current pace of AI, models are becoming obsolete after a couple of years (if not months), and you'll soon need additional capital reinjection.	Having an external vendor doing the pre-integration for you will save you time and money. In addition, by working with an API provider, you can benefit from the knowledge of the whole market, since issues raised by other clients are used to improve the product.

Key factors to consider when picking a STT provider

The performance of an STT model combines many factors, among others:

1. **Latency** (batch vs real-time)
2. **Accuracy**
3. **Features**, such as speaker diarization, custom vocabulary, and sentiment analysis
4. **Language support**
5. **Security and regulatory compliance**, including deployment and maintenance requirements

Let's take a closer look at them.

Latency

When building a voice platform or app, one of your first questions will be **determining the type of transcription best suited to your product and use case.**

The key distinction between asynchronous (async) and real-time transcription lies in **how quickly the transcription is generated** and how it processes audio in relation to the speech.

Async transcription, often called "batch" processing, involves sending audio files for transcription, **with results taking several minutes to hours depending on file length and complexity.**

Real-time transcription, on the other hand, **processes audio instantly**, making it essential for use cases requiring immediate response, such as **conversational bots or live captions for conferences and videos.** It demands low latency, with industry standards ranging from 100 milliseconds to 1 second.

For example, Gladia's real-time API achieves latency under 300 milliseconds, making it ideal for applications like contact center solutions, media platforms, and AI voice assistants.

Key takeaway

Historically, choosing between async and real-time transcription involved balancing speed, quality, and cost. Achieving batch-quality accuracy in real-time often required running both modes in parallel, leading to significant expenses.

Today, advancements in real-time AI offer both speed and accuracy, though it comes at a higher cost due to the computational demands. For critical applications like contact centers, real-time transcription is indispensable for resolving customer issues instantly. Meanwhile, AI meeting assistants often find async transcription sufficient for their core functions.

Features

On top of the core functionality of transcribing audio to text, STT providers are developing **additional features** that make transcripts easier to digest and can provide insights from the audio data.

Let's take a closer look at some of the most popular features among LLM-based voice platforms.

Speaker diarization

Also known as speaker separation, **diarization** allows one to attribute what's been said in a call or meeting to a specific person.

Diarization Error Rate (DER) is the most common metric to evaluate speaker diarization accuracy. DER is calculated by summing the time duration of three distinct errors: **speaker confusion, false alarms, and missed detections**. This total duration is then divided by the overall time span.

Gladia's partner, [pyannoteAI](#), is an industry leader in this area, with diarization models that boast one of the highest precision levels and state-of-the-art solutions for voice AI, including overlapping speech detection.

Why it's useful

To generate speaker-based transcription and enable effective **indexing, analysis** and **search** of audio content for your users.

Custom vocabulary

Many industries rely on **specialized terminology, brand names, and unique language nuances**. Custom vocabulary integration allows the STT solution to adapt to these specific needs, which is crucial for capturing contextual nuances and delivering output that accurately reflects your business needs.

For instance, it allows you to create a list of domain-specific words, such as brand names, in a specific language.

✔ **Why it's useful**

Adapting the transcription to the specific vertical allows you to minimize errors in transcripts, achieving a better user experience. This feature is especially critical in fields like medicine or finance.

Named entity recognition (NER)

NER extracts and identifies key information from unstructured audio data, such as **names of people, organizations, locations**, and more. A common challenge with unstructured data is that this critical information isn't readily accessible—it's buried within the transcript.

To solve this, Gladia developed a structured **Key Data Extraction (KDE)** approach. By leveraging the generative capabilities of its Whisper-based architecture—similar to LLMs—Gladia's KDE **captures context to identify and extract relevant information directly.**

This process can be further enhanced with features like custom vocabulary and NER, allowing businesses to populate CRMs with key data quickly and efficiently.

✔ **Why it's useful**

Having key information like names, companies, and addresses enables automatic **CRM enrichment**, boosting user productivity and saving time.

Additional features to consider:

1. **Topic classification:** Categorizes content into predefined topics for easier content indexing.
2. **Sentiment analysis:** Analyzes the sentiment behind audio recordings to improve customer experiences and sales performance.
3. **Speech moderation:** Flags inappropriate or offensive speech, such as hate speech, based on customizable parameters.

Depending on your use case, you might need some or all of these features alongside transcription. You can opt for a one-stop-shop provider that supports all these functionalities or choose a provider with the best core transcription capabilities and customize the additional features you need.

Below is an overview of the features Gladia's API supports, offering flexibility and efficiency tailored to your business needs.



Transcription

- Diarization
- Word-level timestamp
- Code-switching
- Noise reduction
- Smart formatting
- Translation



Features

- Summarization
- Topic classification
- Chapterization
- Keyword extraction (NER)
- Emotion detection
- Sentiment analysis



Security

- Encryption
- Moderation
- PII redaction
- Certification
- Custom hosting

Accuracy

For businesses operating in customer service – for instance contact centers – maintaining accuracy amidst varying audio quality and background noise is essential. A transcription solution must adapt to challenging environments such as low-quality calls, network disruptions, and diverse speaker accents.

Word Error Rate (WER) is a widely used metric for assessing the accuracy of ASR technologies. It provides a standardized way to compare different speech-to-text (STT) models and providers, helping organizations evaluate their options.

$$\text{WER} = \frac{\text{N}^\circ \text{ of words substitutions} + \text{N}^\circ \text{ of word deletions} + \text{N}^\circ \text{ of word insertions}}{\text{N}^\circ \text{ of words in the reference text}}$$

WER measures the percentage of words in the output that differ from the words in the reference or ground truth text. **A lower WER indicates better performance of the system and vice versa.**

However, WER has notable limitations, particularly when applied to real-world enterprise scenarios. While designed to measure accuracy against an 'ideal' academic benchmark, **WER often falls short in reflecting performance in practical use cases.** In professional environments, the critical factor is not overall transcription accuracy but the precision of key elements—such as names, addresses, or other specific data—used for downstream processes like CRM enrichment.

Standardized benchmarks like WER rarely capture this nuance. **Disregarding a model solely based on its WER score could mean overlooking a solution that, with customization, performs exceptionally well for your specific needs.**

Not everyone needs 100% accuracy

When choosing an ASR provider, impeccable accuracy often comes with trade-offs in speed and cost. **However, not every use case demands perfect transcription.** For example, podcast editing, subtitling, or translation workflows often prioritize accuracy over speed. Conversely, applications like summarization—where AI distills key insights from spoken content—may tolerate minor transcription errors as long as the core message remains clear.

It's essential to go beyond standardized metrics like WER when assessing accuracy. Instead, focus on how well the ASR system performs in real-world conditions, such as handling background noise, diverse speaking styles, and a variety of languages and accents. **Using your own datasets during evaluation can provide a more accurate representation of how the system will function in your specific environment.**

Key takeaway

When selecting an ASR provider, it's crucial to look beyond WER and evaluate how well a model addresses the unique demands of your use case. Balance your accuracy needs with the trade-offs in speed and cost while keeping your use case in mind. By evaluating ASR systems under realistic conditions and tailoring metrics to your requirements, you can ensure the chosen solution aligns with both your operational goals and end-user expectations.

Language support

Most ASR models lean heavily towards certain languages because of the datasets they were trained on, or because they assign uneven weight to certain parameters in the transcription process.

Some of them perform well exclusively with **English and the 30 most commonly represented languages** in written media, with varying WER indicators across languages.

A few providers, like Gladia, cover 100+ languages and handle multiple accents, asynchronously as well as interchangeably in real-time through code-switching.



"Working with Gladia has opened up new geographies to us. We've acquired new users in countries like Finland and Sweden, who say it's the best transcription they ever tried and want to implement Spoke across their global teams thanks to that. Gladia has a clear-cut advantage when it comes to European languages"



Lazare Rossillon, CEO & Co-founder at Meeting BaaS and Spoke

Key takeaway

When selecting an STT provider, ensure it performs well across all relevant languages, accents, and dialects. Providers may claim broad language support, but real-world performance can vary, so thorough internal testing is essential.

Security and regulatory compliance

Given the highly confidential nature of enterprise audio data, it's becoming increasingly important to verify how a provider approaches data privacy.

Enquire with your STT provider about any security-related certifications they have, including [SOC 2 Type 1/Type 2](#), [HIPAA](#), and [ISO 27001](#) or [ISO 27701](#), validating that the company has appropriate security and compliance processes in place.

Furthermore, here are additional security-related techniques that can be applied to protect your audio data:

- 1. Encryption.** Helps to protect sensitive or confidential information contained in audio files, such as customer data, trade secrets, or intellectual property.
- 2. Speech moderation.** Allows to automatically identify and flag hate speech or other inappropriate and offensive verbal content based on predetermined parameters, internal protocols, or external regulations.
- 3. Anonymization of Personally Identifiable Information (PII).** Also known as PII redaction, is used to detect, tag and remove any personally identifying information, such as an address, card number, SSN, phone number, and more.

While it's true that self-hosting is the absolute safest option when it comes to data privacy, the level of security provided by commercial providers is achieving comparable levels.

Besides certification and features, hosting architecture options offered by your provider can further protect your data, as explained below.

Hosting in the cloud, on-premise, or air gap

If you want to embed the speech recognition system in your existing tech stack, you need to decide where the underlying network infrastructure should be located, and who you want to own it.

Cloud multi-tenant (SaaS)

All users share the same hardware and software, as well as the same instance of the software, which is provided by a third-party provider that oversees installation, maintenance, software upgrades, and potential patches.

Pros

This option is the most scalable hosting solution, enabling your company to easily add more users and scale the volume of audio on a pay-as-you-go basis. Regular software updates and patches are part of the package, and you don't need to worry about maintenance or upkeep costs.

Cons

Like with any third-party solution, potential safety hazards in the case of a cloud security breach may make this option less suitable for industries with strict privacy and compliance protocols.

Cloud single-tenant

The concept of cloud single-tenant is similar to multi-tenant, except that it has a dedicated cloud infrastructure per client, managed by an external provider. That means each user has access to their own instance of the software.

Pros

This setup includes a higher level of security and better governance since the virtual network is reserved for a single user.

Cons

As with multi-tenant, data security and privacy is dependent on the provider. It also comes with higher costs.

On-premise

Licensed software is hosted on client-controlled data centers, in an exclusive physical and virtual network that tends to be managed by the company's IT department or a third-party provider.

Pros With this setup, you have full control over what happens to company data

Cons Significant upfront deployment costs. In addition, service-level agreements (SLAs) and other commitments need to be managed internally.

Air gap

Works pretty much like on-premise, except that no third-party providers can access the system since it's completely isolated, even from the internet.

Pros This approach offers an optimal level of protection for high-security facilities with stringent internal protocols.

Cons In the case of a local issue, such as a natural disaster or business interruption, it can take a lot of time to recover and get back on track.

Key takeaway

When choosing between on-premise and cloud-based ASR hosting, consider factors like scalability, cost, and security. While on-premise solutions offer control and potentially better latency, they come with high deployment and maintenance costs and limited scalability. Cloud-based options provide greater flexibility, lower upfront costs, and easy scaling, with security concerns easily addressed by reputable providers. The right choice depends on your business's needs, growth potential, and resources.

Bridging LLMs and ASR: Best practices for building voice apps and audio features

For the last two years, we've worked closely with companies experimenting with the powerful combination of LLMs and ASR models to create pioneering voice platforms and apps. Time and again, they've shared how accurate transcription and advanced language modeling working hand-in-hand are critical to achieving exceptional results.

Drawing on our research and insights from real-world implementations, we've compiled **practical strategies** to help you maximize the potential of both technologies. These quick tips are designed to help you avoid common pitfalls, and deliver the best of both worlds.

Keep reading to discover:

- A checklist to evaluate and optimize your LLM and ASR integrations
- 5 best practices from LLM experts for improving performance

Checklist

Planning and evaluating STT and LLMs for your voice app

This checklist is designed to guide CTOs, developers, and product managers through the foundational steps of **planning and evaluating tools to build voice apps** and **add audio features** to existing products. It focuses on critical early-stage decisions, and should be used as a starting point before moving into development and implementation.

Step 1: Define objectives and use cases

- Clearly define the goals of your voice app, and map these to features
- Identify your target users and the problems the app will solve
- Outline how you will measure success
Ex: accuracy rates, user adoption, improved operational efficiency, etc.

Step 2: Make the buy vs. build decision

- Estimate infrastructure costs, time to deploy, scalability, and ongoing maintenance for both options

Consider building in-house if...

- You have the in-house expertise to implement and maintain custom solutions
- Open-source tools meet your needs without needing extensive customization
- Long-term scalability is manageable with your internal resources

Consider using APIs if...

- Speed to market is a priority
- Your team lacks the hardware or expertise to manage custom solutions
- You want a scalable and low-risk solution with ongoing improvements

Step 3: Evaluate and select tools

Questions to ask when evaluating LLMs:

- What type of model best fits our needs—open-source or proprietary?
Consider customization requirements, cost, and in-house expertise.
- How does the model perform in our key use cases?
Review benchmarks like TruthfulQA or HumanEval to assess performance in tasks like summarization, sentiment analysis, or text generation.
- What is the cost structure for this LLM?
Understand token-based pricing, fine-tuning costs, and whether free credits are available for testing.
- Does the model support our required languages and dialects?
If multilingual support is critical, evaluate the accuracy and language coverage.
- What are the security and compliance standards?
Confirm whether the provider meets SOC 2, GDPR, or other relevant certifications.
- Can this model handle our anticipated data volume and growth?
Check if the model's context window size and infrastructure can support scaling.

Questions to ask when evaluating STT systems:

- What is the system's accuracy and Word Error Rate (WER)?
Compare WER metrics and test performance under real-world conditions like noisy environments and multiple speakers.
- Does the system offer advanced audio intelligence features?
Consider features like speaker diarization, sentiment analysis, custom vocabulary, and code-switching for multilingual conversations.
- What are the latency capabilities?
For real-time applications, confirm whether the system meets low-latency requirements without sacrificing accuracy.

How well does the system support multiple languages and accents?

Test language detection and transcription accuracy for your target audience.

What is the pricing model?

Understand whether pricing is usage-based (per hour or per token) and ensure it aligns with your budget and scalability needs.

What deployment options are available?

Determine if the system can be hosted in the cloud, on-premise, or in an air-gapped environment to meet your security requirements.

Part 3

5 best practices for using STT and LLMs for voice apps

Practice #1

Use LLMs to improve STT output and diarization

As you know all too well, LLM-powered features of voice apps are directly dependent on initial transcription quality.

Choosing a top-tier STT provider is the first step to avoiding problems down the line. But while transcription APIs have certainly reached unprecedented levels of accuracy in the last few years, it may in some instances be helpful to further enhance the quality of your transcripts with the help of LLMs.

Here are some of the most common techniques used to that end.

1. Domain-specific adaptations

LLMs fine-tuned on domain-specific data can **recognize and correct jargon**, technical terms, or industry-specific phrases and **generate context-based suggestions** for specialized vocabulary.

Take healthcare as an example. ASR systems are now commonly used in clinical settings to transcribe doctor-patient interactions into written records and prescriptions. To ensure ultimate information fidelity and avoid critical mistakes, LLMs trained in medical terminology can be integrated into the post-processing of medical records and correct errors and/or hallucinations in transcriptions, ensuring that specific terms (e.g. drug names) are transcribed perfectly.

2. Correcting errors and rephrasing

Because LLMs are typically trained on larger amounts of data than ASR models, they are better suited to identify more complex language patterns, context, and syntax. This makes them useful in spotting errors in transcripts, including **misheard words, homophones, grammar issues** and **filler words**.

3. Correcting punctuation

While most commercial providers do address this issue at least to some extent, ASR systems can produce transcription output with imperfect punctuation or sentence boundaries. LLMs can be used to **add paragraph breaks, capitalization, commas, periods**, and **question marks** based on sentence context and improve readability.

4. Improving speaker diarization

For applications such as contact centers and meeting note-taking apps, knowing who spoke when and what is crucial. As per latest research, LLMs can leverage contextual hints to post-process the outputs from a speaker diarization system and **improve transcript readability, reduce DER**, and even **autofill speaker names and roles**.

Practice #2

Divide and conquer with a multi-model approach

When working with LLMs, try to combine multiple models for the optimal results. You can break tasks down into manageable chunks and assign them to different models depending on their capabilities and your objectives.

For instance, a more powerful model can orchestrate a complex task, while smaller models can handle minor ones. In the case of a note-taking app, a more powerful model would be used to generate complex **summaries** or perform **sentiment analysis**, while a smaller model fills in details and performs tasks such as fact-checking and **cross-referencing**.

Some of our clients noted good performance when using Anthropic's [Haiku 3.3](#) for smaller tasks and [Claude 3.5](#) for more complex operations.

- A multi-model approach doesn't always imply a higher cost, especially for SMEs. Many providers offer free credit that you can use to test out different LLMs. Amazon Web Services (AWS), for instance, recently expanded its free credits program for startups to cover the costs of using major AI models, including Anthropic, Meta, Mistral AI, and Cohere.

Practice #3

Don't resort to fine-tuning too early

Fine-tuning is a great technique for improving a model's output, especially in specialized domains. However, it requires substantial computational resources and is heavily dependent on the data you're using—so collecting, cleaning, and preprocessing the right data can be a significant part of the process.

According to founders we interviewed in the note-taking domain, [prompt engineering is generally a better starting point](#). Prompt engineering doesn't require access to specialized hardware or large datasets – you can often substantially improve output by experimenting with different prompts and playing around with various models.

As many of customer's success stories show, some amazingly advanced AI assistants can be developed with prompt engineering alone.

Here are some best practices when it comes to prompt engineering:

1. Don't settle for the first acceptable result

Prompt engineering requires iterative experimentation and model-specific adjustments.

2. Try various models and prompting techniques

Reiterate and experiment. One of the techniques that performed well for our clients is chain-of-thought (CoT) prompting, discussed previously.

3. Provide examples in your prompt and leverage metadata

Give examples to point towards desired outputs and include key metadata such as speaker identification, timing information, and any additional context (CRM enrichment, for example).

Keep in mind that models and their architectures tend to be quite different. Something can work with one model but not very well with another. You need to make up for that in your prompts and tweak them for each model.

Practice #4

Be mindful of context window size

When building products with LLMs, it's crucial to align the model's context window capabilities with the task requirements.



“When evaluating a model's context window, it's important to consider both the input and output lengths as these factors have a significant impact on how well it performs for certain tasks.”



Lazare Rossillon, CEO & Co-founder at Meeting BaaS and Spoke

Models like [Google's Gemini 1.5 Pro](#), with a 2-million-token input context window, excel at tasks needing vast input processing, such as [summarization](#). However, its 8,192-token output limit can pose challenges for tasks requiring extensive outputs, like [translation](#), where risks of misalignment or hallucinations increase.

In contrast, models like [ChatGPT-4](#) offer smaller, more balanced context windows (4,000–20,000 tokens) that may better suit tasks like real-time conversation or coding assistance.

Some quick tips to address this:

- 1. Break complex tasks** into smaller, manageable subtasks that fit within output token constraints. For example, split long translation tasks into sentence-level chunks to prevent autoregressive errors.
- 2. Implement guardrails** like constraining token prediction space or evaluating intermediate outputs for coherence. When large input contexts are unnecessary, preprocessing strategies such as chunking or prioritizing key sections can reduce computational costs and latency.
- 3. Experiment with different models** to identify the best fit for your use case.

Practice #5

Don't forget context windows' language bias

Context windows suffer from language bias, as the [number of tokens required to represent a concept or word varies across languages](#). This disparity is especially dominant for under-represented languages, where it may take more tokens to convey the same information.

Take Hindi as an example. In English, a single word might be represented by one token, but in Hindi, the same word could require four tokens. As a result, models working with Hindi are four times slower, less precise, and must generate a significantly larger number of tokens to achieve the same outcome as when working with English.



“When dealing with under-represented languages, the context window size is effectively reduced. If a model supports an 8,000-token context window in English, the equivalent input for Hindi might only be around 2,000 tokens. The disparity becomes even more evident in the output.”



Jean-Louis Queguiner, Co-founder and CEO at Gladia

If you're working with under-represented languages, here are some techniques to try besides fine-tuning the model:

- 1. Choose models trained on tokenizers**, optimized for compact representations of specific languages, helping to reduce the token disparity.
- 2. Optimize input preprocessing to remove unnecessary tokens**, such as reducing verbose expressions, simplifying syntax, or eliminating non-essential metadata.
- 3. Break large texts into language-specific chunks** and process them independently to ensure efficient use of the context window.

Conclusion

Large Language Models (LLMs) and Speech to Text (STT) technologies hold incredible potential for building advanced voice apps. As outlined in this guide, leveraging these tools effectively requires a deep understanding of their strengths, limitations, and best practices.

Key insights from this guide

- 1. LLMs:** We explored the wide range of LLMs available, from open-source solutions to proprietary options, and how they can be optimized for top performance through techniques such as prompt engineering, fine-tuning, and RAG to address challenges like hallucinations, limited context windows, and language biases.
- 2. STT:** From in-house, open-source models to APIs from specialized providers, we examined the key factors for selecting the right STT system for your needs, including latency, accuracy, features, and regulatory compliance.
- 3. LLM x ASR:** Implementing voice applications successfully can be achieved through a combination of best practices, such as using LLMs to enhance STT outputs, adopting a multi-model approach, avoiding premature fine-tuning, and staying mindful of the context window and language biases.

Final thoughts

Building with LLMs and STT technologies offers unparalleled opportunities to create intelligent, engaging, and responsive voice applications. Whether you're developing solutions for meeting notes, customer service, or next-generation conversational AI, the insights from this guide provide a solid foundation to get started.

The future of voice-powered platforms is full of potential – and you're now equipped to lead the charge. Start building, innovating, and shaping the next era of AI-driven voice applications today.

About Gladia

From async to live streaming, Gladia's API empowers your platform with accurate, multilingual speech-to-text and actionable insights.

Over 100,000 users and 600 enterprise customers, including Attention, Ausha, Circleback, Method Financial, Recall, and VEED.IO trust us to deliver fast and accurate transcriptions that can be easily scaled and integrated into existing tech stacks.

With Gladia, you can **accelerate your roadmap** with top-tier models for speech recognition and analysis, with industry-leading performance:

- **Latency:** Less than 300 ms to transcribe a call or meeting in real-time, with minimal additional latency to generate summaries and extract insights;
- **Accuracy:** Speech recognition without errors and hallucinations for ultimate information fidelity;
- **Language support:** Multilingual transcription and insights with enhanced support for accents, translation and code-switching;
- **Easy integration:** Our API is compatible with WebSockets, VoIP, SIP, and all other standard telephony protocols and integrate seamlessly with any stack;
- **High security:** We guarantee 100% safety of all user data per EU and US regulations and compliance frameworks.

Request a personalized demo to see our product in action.

See a demo

More information can be found at Gladia's [X](#) or [LinkedIn](#).