Checklist

Planning and evaluating STT and LLMs for your voice app

Gladia

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

Define objectives and use cases

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.

- [ ] **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:

- [ ] **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.

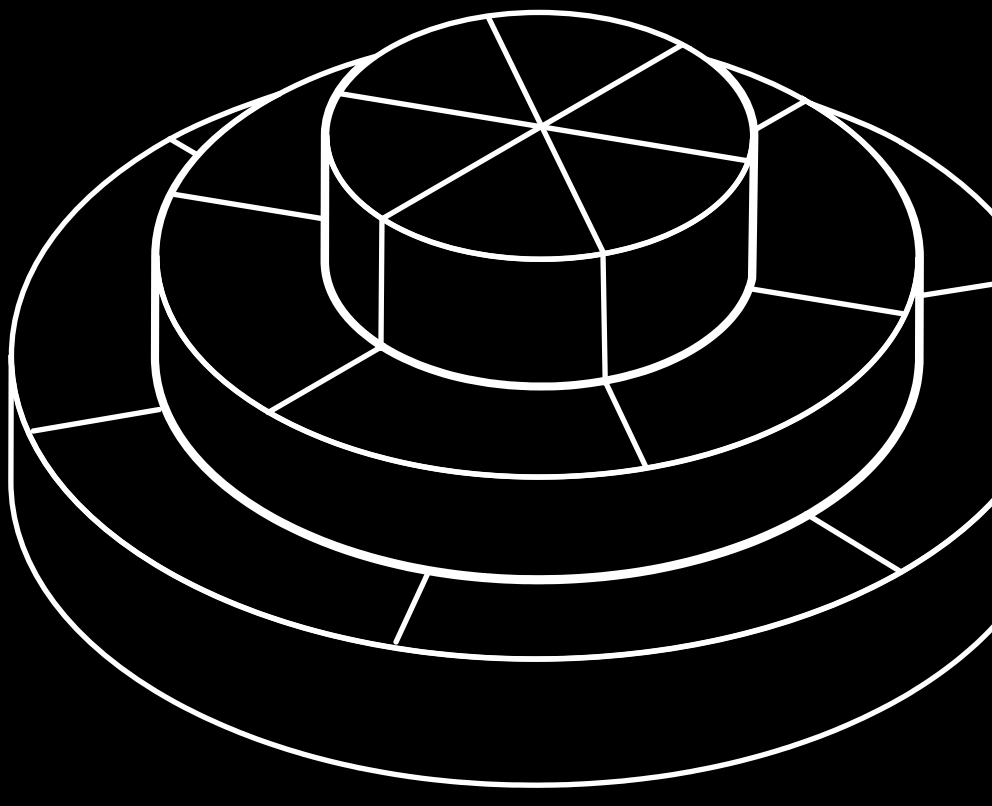
- [ ] **What is the pricing model?**

  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.

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.

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 mistaked, LLMs rained 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.

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**.

Correcting punctuation

While most commercial providers do addres 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.

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 isn’t necessarily costlier for SMEs. Many vendors offer free credits for testing different LLMs. AWS, for example, now extends its free startup credits to AI models from 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:

Don’t settle for the first acceptable result

Prompt engineering requires iterative experimentation and model-specific adjustments.

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.

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.

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.

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.

Jean-Louis Queguiner
Co-founder and CEO at Gladia

"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."

About Gladia

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

Over **150,000 users** and over **1,000 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.

Request a personalized demo to see our product in action.

Book a demo