Speech Toolformer

You will build a speech-first AI assistant based on Gemma-3n (LFM2-Audio or another) that:

1. Takes a spoken query (in Russian or English) ← 🗣️
2. Understands the request directly from audio ← 💭
3. Emits a function call (tool invocation) in a structured format (JSON or XML) ← 🛠️
4. Optionally executes the tool (you need to implement execution logic and return the output back to LLM) ← ⏳
5. Produces a grounded, human-readable answer based on tool output (and optionally a TTS response) ← 💬

We focus here on tool invocation from speech and response generation (based on tool output). We do not need to build the app backend or GUI.

> 💡 **Note:** the model should remain generic, it needs to understand when to trigger a tool and respond in plain text otherwise.

📌 **Reference notebook - Gemma-3n**

📌 **Reference notebook - LFM2-Audio**

Which model should I use?

Gemma-3n E4B Instruct is an instruct model & is natively multi-modal, accepts audio as input and supports function calling via prompting. Gemma is multi-modal understanding model, it can only generate text.

This model can support the following workflow out of the box:

> ℹ️ *audio → Gemma-3n → tool call → tool output → Gemma-3n → final answer (text)*

The model is already highly accurate in instruction following and audio understanding and requires minimal fine-tuning.

> ‼️ **Important considerations**
> Gemma-3n supports over 140 languages. But it is expected that it's performance on English is better than on other languages. If you decide to work with Russian, you will likely have to fine-tune the model first on Russian speech data (for example, CommonVoice)

LFM2-Audio-1.5B is a smaller and a bit less robust model. If you'd like to invest more into model fine-tuning, you can use it. It does not support Russian language. However, this model natively supports speech generation along with text generation.

> 💡 Feel free to propose and use another smallish (up to 4B) multi-modal LLM

It is recommended to use Unsloth fine-tuning setup, especially if you work on T4 machine in Google Colab. Unsloth team implemented custom optimized kernels and quantization utilities for fine-tuning on small GPUs.

Tool(s)

You must implement **at least one tool** and may add more if you have time.
Example tool:

- `weather_forecast(city, date?)`

You can start with a mocking a tool, i.e. return some random (placeholder) responses from your tool. But if you want, you can use a real service with API call. Better avoid tools that require too many arguments or complex parsing. You can also invent fun tools as long as the interface is simple.

Data & Training Strategy

You will teach the model when and how to call a tool. 📝 The key skill is:

> Given a user request, decide whether to call a tool, which tool, with which arguments, and in what exact format.

Recommended workflow

👨‍💻 1. Define your tool API

- Choose a concise interface (`weather_forecast(city, date?)`)
- Decide whether your tool call format will be JSON or XML
- Write a clear schema in your system prompt

```
You are a tool-using assistant. You must respond with exactly one XML tool ca
ll if the tool usage is required, otherwise respond to user's request in plain tex
t.
Template:
<tool_call>
   <name>weather_forecast</name>
   <arguments>
      <city>CITY</city>
   </arguments>
</tool_call>
```

📰 2. Generate synthetic text dataset

Create 50-100 templates per tool, weather examples: `"What's the weather in {city}?", "Do I need an umbrella in {city} now?"`

- Slot-fill cities, dates, etc. and generate ±200–300 text user messages.
- For each user message:
  - Define the expected tool call (correct arguments)
  - Optionally execute your tool to get a tool result JSON/XML
  - Autogenerate a final answer that is consistent with the tool output
- Add 10–20% no-tool examples (questions that should be answered directly without a tool).

📍This is your textual **SFT dataset** for testing!

Now you can test:

🟠 **(A)** Text query → model → Tool call (text-only) — [text instruction following]

Test model's instruction following capabilities on text inputs only. **How to test?** Run inference on your test examples and check if the output of the model follows the schema can be parsed into a valid function call and if the arguments are as expected.

If the model is not capable of following the instruction from text input after a thorough prompt tuning, then we need to fine-tune. Generate a training dataset (~3k examples) with pairs (`text_user_query`, `expected_tool_call`) and fine-tune the model. If the model can already follow tool call instruction reasonably well (after SYSTEM PROMPT tuning), then do not bother with text fine-tuning and go to the next step

📣 3. Generate audio

For each textual user query, synthesize audio (you can use Coqui TTS as you are already familiar with it). For speaker reference you can use any audio dataset, we recommend using CommonVoice, just randomly pick audios from there and use them for reference. In your output dataset store the following: `user_text`, `user_audio`, `assistant_text` (it's recommended to store slots and generate toll call on-the-fly)

With the generated audio you can now explore:

- How well the model can transcribe your generated audios into text, measure WER on test partition of your dataset
- Test model instruction following capability on audio inputs only. Just try to instruct the model to understand user's speech and directly generate a function call (see reference notebooks)

Test the following pipelines:

🟠 **(A)** Text query → model → Tool call (text-only) — [text instruction following]

🟠 **(B)** Audio query → model → ASR Transcript — [ASR capability]

🟠 **(C)** Audio query → model → Transcript + Tool call in a single run — [joint pipeline]

🟠 **(D)** Audio query → model → Transcript → model → Tool call — [cascaded pipeline]

In case you observe that the model fails on recognizing speech or on instruction following after a dedicated system prompt tuning, you can additionally fine-tune the model. Compare how well each of the pipelines (A) - (D) work and how much of fine-tuning is required.

💪 4. Fine-tune

Use LoRA/QLoRA on selected modules only (attention/MLP and possibly audio front-end). If you have enough compute, full model fine-tuning is also an option.

- LFM fine-tuning docs: https://docs.liquid.ai/lfm/fine-tuning/unsloth
- Gemma-3n unsloth fine-tuning docs: https://docs.unsloth.ai/models/gemma-3-how-to-run-and-fine-tune/gemma-3n-how-to-run-and-fine-tune

📊 5. Evaluation & Metrics

You will synthesize an eval set that covers both typical and edge cases. We care about:

1. **Tool call correctness (on text):** precision, recall, false alarm rate, parsable tool invocation rate

2. **Modality gap (text vs audio):** run the same prompts as pure text and as synthetic audio and measure the difference in metrics
3. **ASR quality**
    - Measure WER for the audio part on your test partition
    - Pay attention to the correctness of recognizing arguments that matter for tools (like city name in the example)

Feel free to add extra metrics (latency, robustness).

📦 Deliverables

1. **Colab notebook**
    - End-to-end demo: microphone / audio file → your model → tool call → tool result → your model → final answer (→ optional TTS)
    - Experiments for different configurations (system prompts, pipelines, fine-tuned vs. base).
2. **Short report**
    - Data design: how you generated synthetic data + audio
    - Training setup: model choice, LoRA config, learning rate, masking strategy
    - Evaluation metrics, results, failure modes
    - You can also include takeaways, like what worked, what didn't, and what you'd try if you had more time

📚 References

- https://developers.googleblog.com/en/introducing-gemma-3n-developer-guide/
- https://docs.unsloth.ai/models/gemma-3-how-to-run-and-fine-tune/gemma-3n-how-to-run-and-fine-tune
- https://ai.google.dev/gemma/docs/capabilities/function-calling
- https://ai.google.dev/gemma/docs/gemma-3n

- https://ai.google.dev/gemma/docs/core/huggingface_text_finetune_qlora
- MatFormer: Nested Transformer for Elastic Inference — https://arxiv.org/pdf/2310.07707
- https://www.liquid.ai/blog/lfm2-audio-an-end-to-end-audio-foundation-model
- https://github.com/Liquid4All/liquid-audio
- https://docs.liquid.ai/lfm/fine-tuning/unsloth

Project evaluation

> 💡 **Important:** You are not graded on having the best numbers (WER, accuracy, etc.). Different languages, tools, and data setups have different difficulty. We grade based on experimental design, correctness of evaluation, and clarity of reasoning, not on reaching specific metric values

- 🎁 **2 bonus pts — For creativity in tool selection**

  The very first task is to select a tool that your model will call on user's request. Feel free to come up with whatever tool you find more useful, fun or interesting to work with. It can be any tool , different from a boring weather tool in the reference (for example, currency conversion)

- **2 pts – System prompt & base text instruction following**

  A clear system prompt that defines the tool and output schema (JSON/XML), plus a text-only evaluation of base model instruction following on your test set (including tool vs non-tool requests, example outputs, analysis).

- **3 pts – Text tool-usage metrics & interpretation**

  Reported metrics for text-only tool use (accuracy, precision, recall, false alarm rate, parsable tool invocation rate).

- **4 pts – Synthetic dataset (text + audio) design & description**

A synthetic test dataset with `user_text`, `user_audio`, and ground-truth tool calls; multiple voices and/or languages if applicable. Short but clear description of how prompts, slots (cities/dates/etc.), no-tool cases, and TTS audio were generated.

- **3 pts – ASR baseline: WER & error analysis**

  Evaluation of the model as an ASR on your synthetic audio test set with WER reported (per language if used) and a short error analysis (e.g. typical mistakes: names, dates, numbers).

- **3 pts – Workflows A-D evaluations**

  Evaluation of the (A) - (D) workflows on the same test set. For each tested workflow, report tool-usage metrics (precision, recall, false alarm rate, parsable tool invocation rate) and compare text vs audio where applicable.

- **2 pts – Choice of best workflow & justification**

  A clear statement of which workflow you consider most optimal for this project and why, based on your metrics, robustness, and simplicity

- **3 pts – Final report & discussion of experiments**

🏋️ Extra points for fine-tuning

- **+5 pts – Text fine-tuning**

  You fine-tuned the model on a text-only dataset for better tool calling. If done:

  - report tool usage metrics before and after
  - report basic description of dataset, objective, and training setup (model variant, LR, steps, LoRA or not)

- **+5 pts – ASR or audio-conditioned fine-tuning**

  You fine-tuned the model to improve ASR or joint audio→(transcript + tool) behavior. If done:

  - report WER before and after, and WERR (relative improvement), tool usage metrics before and after

- report basic description of dataset, objective, and training setup (model variant, LR, steps, LoRA or not)