Step-by-Step: Build an Open-Source Calling Agent with Web Interface

---

Overview of Components

Component	Recommended Open Source Tool
Telephony Server	**Asterisk** or **FreeSWITCH**
Speech-to-Text (STT)	**Whisper (OpenAI Whisper or Faster Whisper)**
Language Model (LLM)	**LLaMA 2** (local via llama.cpp) or **GPT4All**
Text-to-Speech (TTS)	**Coqui TTS** or **Mozilla TTS**
Orchestration API	**FastAPI (Python)**
Web Dashboard	**React.js** + **FastAPI backend**

Step 1: Set Up Your Telephony Server (Asterisk)

Asterisk is an open-source PBX system that will accept calls from customers and let you control call audio.

Install Asterisk (Ubuntu example)

```
sudo apt update
sudo apt install -y asterisk
sudo systemctl start asterisk
sudo systemctl enable asterisk
```

Basic Dialplan (extensions.conf)

Configure `/etc/asterisk/extensions.conf` to route incoming calls to your app:

```
 [default]
exten => 1234,1,Answer()
 same => n,AGI(agi://127.0.0.1:5000)
 same => n,Hangup()
```

This tells Asterisk to answer calls to extension 1234 and hand off audio to your AGI server (which we'll build in Step 3).

Step 2: Speech-to-Text with Whisper

Use OpenAI Whisper or the faster faster-whisper for transcribing audio.

Install Whisper:

```
pip install faster-whisper
```

Sample code to transcribe audio:

```
from faster_whisper import WhisperModel

model = WhisperModel("base")

def transcribe_audio(audio_path: str) -> str:
    segments, info = model.transcribe(audio_path)
    text = " ".join(segment.text for segment in segments)
    return text
```

Step 3: Build the Orchestrator API with FastAPI

This service will:

- Accept audio from Asterisk via AGI or WebSocket,
- Transcribe it,
- Send the text to an LLM,
- Generate a spoken reply via TTS,
- Send audio back to Asterisk.

Install FastAPI and dependencies:

```
pip install fastapi uvicorn pydub
```

Sample FastAPI app `orchestrator.py`

```
from fastapi import FastAPI, UploadFile, File
from faster_whisper import WhisperModel
from llm import query_llm
from tts import text_to_speech
import tempfile
from pydub import AudioSegment

app = FastAPI()
model = WhisperModel("base")

@app.post("/call/audio")
async def process_audio(file: UploadFile = File(...)):
    # Save incoming audio file temporarily
    temp_audio = tempfile.NamedTemporaryFile(delete=False, suffix=".wav")
    content = await file.read()
```

```
    temp_audio.write(content)
    temp_audio.close()

    # Transcribe
    segments, _ = model.transcribe(temp_audio.name)
    text = " ".join(segment.text for segment in segments)

    # Query LLM
    response_text = query_llm(text)

    # Convert to speech
    speech_path = text_to_speech(response_text)

    # Return TTS audio file to caller
    speech_audio = AudioSegment.from_file(speech_path)
    output_audio_path = tempfile.NamedTemporaryFile(delete=False,
suffix=".wav").name
    speech_audio.export(output_audio_path, format="wav")

    return {"response_text": response_text, "audio_file": output_audio_path}
```

Step 4: LLM Query Module (Local LLaMA or GPT4All)

You can run LLaMA locally via `llama.cpp` and expose it via HTTP API or use GPT4All Python API.

Example `llm.py`:

```
def query_llm(prompt: str) -> str:
    # Placeholder: replace with your llama.cpp or GPT4All integration
    # Here, we just echo input for demo purposes
    return f"Agent reply to: {prompt}"
```

Step 5: Text-to-Speech with Coqui TTS

Install Coqui TTS:

```
pip install TTS
```

Example `tts.py`:

```
from TTS.api import TTS
import tempfile

tts = TTS(model_name="tts_models/en/ljspeech/tacotron2-DDC",
progress_bar=False, gpu=False)
```

```
def text_to_speech(text: str) -> str:
    out_path = tempfile.NamedTemporaryFile(delete=False, suffix=".wav").name
    tts.tts_to_file(text=text, file_path=out_path)
    return out_path
```

Step 6: Connect Asterisk to your FastAPI

You can build an **AGI script** (Asterisk Gateway Interface) that streams or sends call audio to your FastAPI endpoints, and returns TTS audio for playback.

Example AGI Python script (simplified):

```
#!/usr/bin/env python3
import sys
import requests

def agi_read():
    while True:
        line = sys.stdin.readline().strip()
        if line == '':
            break

def main():
    agi_read()
    # For demonstration: ask caller to record a message
    print("STREAM FILE hello-world \"\" 0")
    print("RECORD FILE /tmp/call-recording  wav  5  0  s")
    sys.stdout.flush()

    # Send recorded file to API (implement this part with proper file
transfer)
    # Receive TTS audio and play back

if __name__ == "__main__":
    main()
```

You can find complete AGI examples in Asterisk AGI docs.

Step 7: Simple Web Dashboard with FastAPI + React

You can add call logs and transcripts by saving calls to a database (like SQLite or Redis) in your FastAPI backend, and expose APIs like `/calls` to fetch recent calls.

Example API to list calls (save calls in DB as they come in):

```
from fastapi import FastAPI
```

```
app = FastAPI()

calls = []

@app.post("/log_call")
def log_call(call: dict):
    calls.append(call)
    return {"status": "ok"}

@app.get("/calls")
def get_calls():
    return calls
```

Then, a React app fetches `/calls` and displays them.

Summary & Tips:

- **Start with local testing**: test STT + LLM + TTS locally before integrating telephony.
- Use **ngrok** or other tunneling to expose your FastAPI if testing with Twilio or Asterisk on local network.
- Optimize for **low latency**: quantize your LLM, use smaller models, and low-latency TTS.
- Use **Redis** or **SQLite** for call/session context.
- For production, consider **FreeSWITCH** if you need more scalable SIP capabilities.
- You can replace FastAPI with any backend that supports streaming audio and HTTP APIs.