A Multi-Layered Voice Authentication System: Architectural Design and Implementation Guide

Section 1: Architectural Blueprint for a Multi-Layered Voice Authentication System

This section establishes the high-level design for a secure, multi-layered voice authentication system. It defines the core components, their interactions, and the logical flow of data, serving as the foundational map for the entire implementation. The architecture is predicated on a defense-in-depth strategy, ensuring that multiple, independent security checks are performed before granting access to sensitive data.

1.1. System Overview and Component Diagram

The proposed system is a distributed architecture composed of specialized services, each responsible for a distinct part of the authentication workflow. This separation of concerns is a key architectural principle that enhances modularity, scalability, and maintainability. The orchestration of these services is managed by n8n, a workflow automation platform, which acts as the central controller for the business logic.

The end-to-end data flow can be visualized as follows:

User Phone -> Twilio (Telephony Gateway) -> VAPI (Conversational AI Agent) -> n8n (Orchestration Layer via Webhook) -> Custom Security Backend (AI/ML Inference via HTTP Request) -> MFA Provider (Secondary Authentication via HTTP Request) -> Data Store (Resource Access) -> n8n (Response Aggregation) -> VAPI (User Feedback) -> User Phone

The roles of each component are defined as:

- **Twilio:** Serves as the Public Switched Telephone Network (PSTN) gateway. It acquires and manages the phone number that users call, receives inbound calls, and forwards them to the VAPI platform for handling.[1]
- **VAPI (Voice AI Platform):** Provides the conversational AI agent powered by an OpenAI model. Its primary role is to answer the call, deliver a prompt to the user (e.g., "Please state your passphrase"), and capture the user's spoken response.[3] Upon capturing the audio, it triggers the orchestration workflow by making a call to an n8n webhook.
- **n8n:** Acts as the central orchestrator or "nervous system" of the application. It is a low-code platform that defines the sequence of operations, manages conditional logic (e.g., if verification passes, proceed to the next step), and calls the various external services via their APIs.[5] It does not perform heavy computation itself but rather directs the flow of data between the specialized components.
- **Custom Security Backend:** This is a bespoke microservice, likely developed in Python, that houses the computationally intensive AI/ML models. Its sole responsibility is to receive an audio sample and perform a cascade of security analyses: voice biometric verification and anti-spoofing detection. It exposes a simple REST API for n8n to call.
- **MFA Provider (e.g., Duo Security):** An external service that provides the final, out-of-band authentication factor. After the voice-based checks are passed, n8n will call this service's API to send a push notification to the user's registered device.[7]
- **Data Store:** A database or secure file storage that holds the sensitive medical or financial data the user wishes to access. Access is granted only after all preceding security gates have been successfully passed.

This decoupled architecture ensures that the core security logic within the Custom Security Backend can be updated, scaled, or replaced (e.g., swapping in a new anti-spoofing model) without requiring any changes to the high-level business logic defined in the n8n workflow. This modularity is critical for long-term maintenance and adaptation to new security threats.

1.2. The Security Gateway Sequence: A Defense-in-Depth Approach

Authentication is not treated as a single event but as a sequence of independent security "gates." A user must pass through each gate in order before the next is attempted. This sequential model provides multiple layers of defense and creates clear, auditable checkpoints for security monitoring and incident analysis. If an authentication attempt fails, the system can log precisely which gate was not passed, providing invaluable data for understanding attack vectors and user issues.

- **Gate 1: Call Ingress & Conversational Prompting (Twilio & VAPI):** The process begins with the secure reception of the user's call and the capture of their spoken passphrase. VAPI's AI agent handles the initial interaction, ensuring a consistent and clear prompt is delivered to the user.[4]
- **Gate 2: Identity Verification (Backend - Speaker Verification):** The first computational check answers the question, "Is this the correct person?" The captured audio is analyzed to extract a voice embedding (a mathematical representation of the voice), which is then compared against a pre-enrolled voice signature for that user.[10]
- **Gate 3: Liveness & Authenticity Check (Backend - Anti-Spoofing):** This gate answers the critical question, "Is this a live, human voice, or a fake?" This check is a composite of two sub-gates:
  - **Deepfake Detection:** Identifies synthetically generated speech, such as that from advanced Text-to-Speech (TTS) or Voice Conversion (VC) models.[12]
  - **Replay Attack Detection:** Identifies recorded speech that is being played back to the system.[14]
- **Gate 4: Active Confirmation (MFA Provider):** This final gate answers, "Can the user confirm this login attempt on a separate, trusted device?" After all voice-based checks are successful, an out-of-band challenge, such as a push notification, is sent to the user's smartphone via the MFA provider's API.[7]
- **Gate 5: Authorization & Data Access:** Only if all four preceding gates are passed is an access token generated and used to retrieve the sensitive data from the data store.

1.3. API Contracts and Data Payloads

Precise API contracts are essential for the seamless integration of these distributed components. The following JSON payloads define the data exchanged at each key interface.

- **VAPI to n8n (via Webhook):** VAPI will be configured to use a "Tool Call" to trigger the n8n workflow. When the user speaks their passphrase, VAPI will send a POST request to the n8n Webhook URL with a payload containing details about the

function call event.[16] The audio itself is typically retrieved via a recording URL provided in a subsequent event or by querying the VAPI API.
Payload Example (function-call event):
JSON
```json
{
  "message": {
    "type": "function-call",
    "call": {
      "id": "call_xxxxxxxxxxxxxxxx",
      "phoneCall": {
        "from": "+15551234567",
        "to": "+15557654321"
      },
     ...
    },
    "functionCall": {
      "name": "verifyPassword",
      "parameters": "{ \"passphrase\": \"open sesame\" }"
    }
  }
}
```

Note: The end-of-call-report event will contain the recordingUrl which is crucial for processing.[17] The workflow should be designed to wait for this event or use the
call.id to query the VAPI API for the recording.

- **n8n to Custom Security Backend:** n8n's HTTP Request node will make a POST request to the backend microservice, providing the user's identifier and the URL to the audio recording.
  Payload Example:
  JSON
```json
{
  "userId": "user-123",
  "audioUrl": "https://vapi-public.s3.amazonaws.com/recordings/xxxxxxxx.wav",
  "claimedPhoneNumber": "+15551234567"
}
```

- **Custom Security Backend to n8n:** The backend will respond with a detailed JSON object that clearly states the outcome of each security check. This rich

response allows for granular logic and logging within n8n.
Success Response Example:
```json
{
  "status": "success",
  "checks": {
    "speakerVerification": {
      "passed": true,
      "confidence": 0.98
    },
    "antiSpoofing": {
      "passed": true,
      "deepfake": {
        "isSpoof": false,
        "confidence": 0.95
      },
      "replayAttack": {
        "isReplay": false,
        "confidence": 0.99
      }
    }
  }
}
```

Failure Response Example:
```json
{
  "status": "failure",
  "reason": "Speaker verification failed.",
  "checks": {
    "speakerVerification": {
      "passed": false,
      "confidence": 0.45
    },
    ...
  }
}
```

- **n8n to MFA Provider (Duo):** n8n will make a POST request to the Duo Auth API's

/auth endpoint to trigger the push notification.[8]
Request Parameters (Form-encoded):
username=user-123&factor=push&device=auto&async=1
- **n8n to Data Store:** After successful MFA, a final GET request is made to a secure endpoint to retrieve the user's data, using an authorization token if required.

Section 2: The Conversational Front-End: Integrating Twilio and VAPI

This section provides a practical guide to configuring the user-facing components of the system. It details the setup of Twilio for telephony ingress and the design of the VAPI assistant that will interact with the user and capture their voice passphrase for verification.

2.1. Twilio Configuration for VAPI Integration

The first step is to establish a pathway for phone calls to reach the VAPI AI agent. Twilio offers two primary methods for this integration: direct webhook configuration and the more robust Elastic SIP Trunking.

- **Acquiring a Phone Number:** The process begins in the Twilio Console by purchasing a phone number that will serve as the entry point for users.[21] This number must have voice capabilities enabled.
- Method 1: Webhook Configuration (for Development and Simplicity):
  This is the quickest way to get started. In the Twilio phone number's configuration settings, under "A CALL COMES IN," you can set the webhook to point to a server endpoint that you control. This endpoint's responsibility is to receive the call metadata from Twilio and then programmatically initiate an outbound call from VAPI to the user, effectively bridging the two.1 Alternatively, if VAPI provides a direct webhook URL, it can be placed here. This method is excellent for rapid prototyping but may be less flexible for complex call-handling logic like placing users on hold with music, which is better managed with TwiML and conferences.1
- Method 2: Elastic SIP Trunking (for Production and Scalability):
  For a production-grade system, Elastic SIP Trunking is the recommended

approach. It provides greater control over call routing, can be more cost-effective at scale, and is the standard for interconnecting telephony platforms.2 The setup involves several steps within the Twilio Console:

1. **Create an Elastic SIP Trunk:** Navigate to the Elastic SIP Trunking section and create a new trunk.
2. **Configure Termination:** Termination settings define how calls *from* your Twilio trunk are sent *to* an external system. Here, you will set the Termination SIP URI to VAPI's provided SIP endpoint. To secure this connection, you must add VAPI's static IP addresses (e.g., 44.229.228.186, 44.238.177.138) to the IP Access Control List (ACL) to whitelist them.[2]
3. **Configure Origination:** Origination settings define how calls *from* an external system are sent *to* your Twilio numbers. You will configure the Origination SIP URI in the format sip:YOUR_PHONE_NUMBER@sip.vapi.ai, which tells Twilio how to route incoming calls from VAPI.[2]
4. **Assign Number to Trunk:** Finally, associate the phone number you purchased with the newly created SIP trunk.

The choice between these methods has significant implications. Webhooks are straightforward for a proof-of-concept, but SIP trunking offers the performance, cost, and control benefits required for a high-stakes production environment handling sensitive financial or medical data.

2.2. Designing the VAPI Assistant

The VAPI assistant is the AI-powered agent that interacts with the user. Its design is critical for a smooth and secure user experience. Configuration can be done through the VAPI Dashboard or programmatically via its API.[9]

- **Creating the Assistant:** An assistant is created with a name, a designated language model, and a voice.[4]
- **Model Selection:** VAPI supports various providers, including OpenAI. For this use case, a powerful and responsive model like gpt-4o is recommended. Vapi's integration with OpenAI allows for the use of their flagship models, which are capable of nuanced, human-like conversation and understanding.[4]
- **System Prompt Engineering:** The system prompt is the core instruction set that governs the AI's behavior. It must be crafted with precision to constrain the agent to its specific security task and prevent conversational drift.

Example System Prompt:
You are a secure voice authentication agent. Your sole purpose is to greet the user and ask them to state their secret passphrase. Do not engage in any other conversation. Do not answer questions. If the user says anything other than their passphrase, politely repeat the instruction: "Please state your secret passphrase now." Once the user provides a passphrase, you must use the 'verifyPassword' tool.

- **First Message:** This is the initial greeting the user hears upon the call connecting. It should be clear, concise, and immediately set the user's expectation.
  Example First Message:
  "Welcome to the secure access system. Please state your secret passphrase now."

2.3. Capturing the Passphrase: VAPI Function and Tool Calling

The mechanism for transferring the user's spoken passphrase to the backend for analysis is VAPI's Tool Calling feature (previously known as Function Calling).[18] This allows the AI assistant to trigger an external API call in response to the conversation.

- **Defining the Tool:** In the VAPI Dashboard, a new tool of type "Function" is created.
  - **Tool Name:** A descriptive name, such as verifyPassword.
  - **Description:** A clear description for the language model to understand its purpose, e.g., "Call this function after the user has spoken their passphrase to begin the verification process."
  - **Parameters:** For this use case, no explicit parameters are needed in the tool definition itself, as the primary input is the audio of the user's last utterance, which is implicitly handled by the VAPI platform.
  - **Server URL:** This is the most critical setting. It must be set to the unique Webhook URL generated by the n8n workflow's trigger node.[16]
- **The Tool Call Process:**
  1. The user calls the Twilio number.
  2. The call is routed to the VAPI assistant.
  3. The assistant plays the "First Message."
  4. The user speaks their passphrase.
  5. Based on the system prompt, the language model determines that the

verifyPassword tool should be called.

6. VAPI sends a POST request to the configured Server URL (the n8n webhook) with a tool-calls payload.[18] This payload contains metadata about the call and the tool that was invoked.

- **Handling the Response and User Experience:** The interaction's user experience is determined by whether the tool call is synchronous or asynchronous. VAPI's function configuration allows an async: true parameter.[17]
  - **Synchronous (Default):** If async is false, the VAPI assistant will pause and wait for a response from the n8n workflow. This entire backend process (biometrics, anti-spoofing, MFA) could take several seconds. During this time, the user hears silence, which can be a poor experience. To mitigate this, the VAPI assistant could be programmed to say, "Thank you. Verifying your identity, please hold," before the tool call is made. An even better approach involves using Twilio's <Conference> TwiML to place the user in a hold state with music while the backend processing occurs.[1]
  - **Asynchronous (async: true):** If async is true, VAPI does not wait for a response. It can immediately continue the conversation (e.g., "Thank you, I have received your passphrase. You will receive a notification shortly to complete your login."). The n8n workflow would then proceed independently. This model is less suitable for this specific use case, as the final result of the authentication needs to be communicated back to the user on the same call. Therefore, the synchronous approach, combined with good user experience management for the waiting period, is superior.

Section 3: Core Identity Verification: Implementing Voice Biometrics

This section addresses the first and most fundamental security check performed by the custom backend: voice biometrics. The goal is to verify that the speaker is the legitimate owner of the account by comparing their voice to a previously enrolled "voice signature."

3.1. Foundational Concepts: Speaker Verification vs. Identification

It is critical to distinguish between two related but distinct tasks in speaker recognition [10]:

- **Speaker Verification:** This is a 1:1 matching process. The system is given a voice sample and a claimed identity, and it must answer the question, "Is this voice sample from the person it claims to be?" This results in a binary decision (yes/no) or a similarity score. This is the correct approach for an authentication workflow.
- **Speaker Identification:** This is a 1:N matching process. The system is given a voice sample and must determine which speaker from a known set of N speakers it belongs to. This is used in scenarios like identifying speakers in a meeting recording.

This workflow exclusively requires **speaker verification**.

3.2. Framework Selection: SpeechBrain

For implementing speaker verification, the **SpeechBrain** toolkit is the recommended choice. SpeechBrain is an open-source, PyTorch-based toolkit designed to be flexible, user-friendly, and comprehensive for a wide range of conversational AI tasks.[25] Its key advantages for this project include:

- **State-of-the-Art Models:** It provides implementations of cutting-edge architectures for speaker recognition.[27]
- **Hugging Face Integration:** SpeechBrain hosts over 100 pre-trained models on the Hugging Face Hub, which can be loaded and used for inference with just a few lines of code.[26]
- **Task-Specific Interfaces:** It offers high-level inference classes like EncoderClassifier and SpeakerRecognition that abstract away much of the underlying complexity, making it ideal for rapid development.[11]

3.3. Model Selection and Analysis

Within the SpeechBrain ecosystem, several pre-trained models are suitable for speaker verification. The choice of model impacts the accuracy and robustness of the system.

Table 3.1: Comparison of Pre-trained Speaker Verification Models

Model Name	Architecture	Training Dataset	Embedding Size	Key Features	Hugging Face Link
speechbrain/spkrec-ecapa-voxceleb	ECAPA-TDNN	VoxCeleb 1 & 2	192	State-of-the-art performance. Emphasized Channel Attention, Propagation, and Aggregation. Includes a high-level SpeakerRecognition interface for easy verification.[11]	Link
speechbrain/spkrec-xvect-voxceleb	TDNN (x-vector)	VoxCeleb 1 & 2	512	Well-established, industry-standard architecture. Robust and widely studied. Provides a solid baseline.[30]	Link
pyannote/speaker-diarization-3.1	PyanNet	Multiple	512	Primarily designed for speaker diarization, but its speaker embedding model can	Link

				be used for verification. Part of the pyannote.audio toolkit.[31]	

Primary Recommendation: The speechbrain/spkrec-ecapa-voxceleb model is the superior choice for this high-security application. The ECAPA-TDNN (Emphasized Channel Attention, Propagation and Aggregation in Time Delay Neural Network) architecture represents the state-of-the-art in speaker verification, generally outperforming older architectures like x-vectors on benchmark tasks.[32] Furthermore, its availability through a simple, high-level

SpeakerRecognition API in SpeechBrain makes it both powerful and easy to implement.[11]

3.4. Implementation Guide: The Backend Service (Python/Flask)

The backend service will have two primary functions related to voice biometrics: enrolling a new user and verifying an existing user.

Step 1: User Enrollment

Enrollment is a one-time process where a user's voice signature is created and stored. This should be done in a secure context, separate from the main authentication flow.

Python

```python
import torchaudio
from speechbrain.inference.speaker import EncoderClassifier
import numpy as np

# Load the pre-trained encoder model once during application startup
# Use run_opts={"device":"cuda"} for GPU acceleration
classifier = EncoderClassifier.from_hparams(
    source="speechbrain/spkrec-ecapa-voxceleb",
    savedir="pretrained_models/spkrec-ecapa-voxceleb"
)

def enroll_user(user_id, audio_file_path):
    """
    Generates and stores a voice embedding for a new user.
    """
    try:
        # Load and preprocess the audio file
        signal, fs = torchaudio.load(audio_file_path)

        # Generate the embedding (voice signature)
        # The model expects a batch, so we add a dimension
        embedding = classifier.encode_batch(signal)

        # Squeeze to remove batch dimension for storage
        embedding_numpy = embedding.squeeze().cpu().numpy()

        # --- DATABASE LOGIC ---
        # Store the 'embedding_numpy' array in a database,
        # associated with the 'user_id'.
        # For example, in a PostgreSQL DB with a bytea or vector column.
        print(f"Successfully generated embedding for user {user_id}.")
        return embedding_numpy

    except Exception as e:
        print(f"Error during enrollment for user {user_id}: {e}")
        return None

# Example usage:
# enroll_user("user-123", "/path/to/enrollment_audio.wav")
```

Step 2: Verification

This function is called by the API endpoint that n8n communicates with. It takes the incoming audio, generates an embedding, and compares it to the stored one.

Python

```python
from speechbrain.inference.speaker import SpeakerRecognition
import os

# Initialize the full SpeakerRecognition interface
# This class includes both the encoder and the comparison logic
verification = SpeakerRecognition.from_hparams(
    source="speechbrain/spkrec-ecapa-voxceleb",
    savedir="pretrained_models/spkrec-ecapa-voxceleb"
)

# The verification threshold is a critical parameter.
# The default is tuned on a specific dataset. This value MUST be
# adjusted and tested based on your specific use case and risk tolerance.
# A higher threshold is more secure but may reject more valid users.
verification.classifier.hparams.similarity_threshold = 0.65 # Example value

def verify_user_voice(user_id, incoming_audio_path):
    """
   Verifies an incoming voice sample against a pre-enrolled signature.
   """
    try:
        # --- DATABASE LOGIC ---
        # Retrieve the stored embedding for 'user_id' from the database.
        # For this example, we assume it's saved to a file.
        stored_embedding_path = f"./embeddings/{user_id}.npy"
        if not os.path.exists(stored_embedding_path):
            raise FileNotFoundError("User not enrolled.")

        # The verify_files method is a high-level API that handles
        # loading audio, creating embeddings, and comparing them.
        # It's more efficient to create the embedding for the stored audio
        # once and save it, then compare embeddings directly.
        # However, for simplicity, verify_files is shown here.
        # A more optimized approach would use verify_batch with embeddings.

        # For this example, we'll simulate the enrolled audio being a file
        enrolled_audio_path = f"./enrollment_audio/{user_id}.wav" # Placeholder

        score, prediction = verification.verify_files(
            enrolled_audio_path,
            incoming_audio_path
        )

        # prediction is a tensor, convert to boolean
        is_verified = bool(prediction)

        return {
            "passed": is_verified,
            "confidence": float(score)
        }

    except Exception as e:
        print(f"Error during verification for user {user_id}: {e}")
        return {"passed": False, "confidence": 0.0}

# Example usage:
# result = verify_user_voice("user-123", "/path/to/auth_attempt.wav")
# print(result)
```

A critical operational detail is the similarity_threshold. This value determines the cutoff point for accepting or rejecting a match. It directly controls the trade-off between the False Acceptance Rate (FAR), where an impostor is incorrectly accepted, and the False Rejection Rate (FRR), where a legitimate user is incorrectly rejected. This threshold should not be left at its default value but must be carefully tuned based on the application's security requirements and validated against a test set of genuine and impostor audio samples.

Section 4: Advanced Security Countermeasures: Detecting Spoofing and Replay Attacks

Passing voice biometric verification confirms that a voice *sounds like* the enrolled user, but it does not confirm that the voice is *genuine*. A sophisticated attacker could use a high-quality recording (a replay attack) or a synthetic deepfake voice (a presentation attack) to fool the verification system. This section details the implementation of a second, crucial security layer: anti-spoofing. This directly addresses the user's requirement for a "GAN like model" to detect fakes, providing a more nuanced and robust solution.

4.1. The Modern Threat Landscape: Presentation Attacks

A Presentation Attack (PA) is an attempt to subvert a biometric system by presenting a fake or artificial biometric sample.[14] In the context of voice authentication, PAs fall into two main categories:

- **Logical Access (LA) Attacks:** These are attacks using synthetically generated or modified speech. This category includes:
  - **Text-to-Speech (TTS):** An AI model generates speech directly from text.
  - Voice Conversion (VC): An AI model converts the voice of one speaker to sound like a target speaker.
    These are the "deepfake" attacks that have become increasingly sophisticated. The ASVspoof challenge series has dedicated tracks for LA countermeasures, providing standardized datasets and benchmarks for evaluating detection models.13
- **Physical Access (PA) Attacks:** This category primarily consists of **replay attacks**. An attacker obtains a recording of a legitimate user's voice and simply plays it back to the system's microphone during an authentication attempt.[14] These are considered "low-effort" but can be surprisingly effective if not specifically defended against.[15]

A robust system must defend against both LA and PA attacks, as they exploit different vulnerabilities and leave different acoustic artifacts.

4.2. Deepfake and Synthetic Voice Detection (LA Countermeasure)

The detection of synthetic voices is a rapidly evolving field. Early deepfakes created with vocoder-based methods left predictable artifacts that detectors could easily identify. However, modern deepfakes, particularly those generated by neural audio codecs, are far more realistic and challenging to detect.

The emergence of the **CodecFake** dataset and associated research highlights a critical vulnerability: anti-spoofing models trained on older datasets (like ASVspoof 2019) often fail to detect these modern, codec-based fakes.[34] The underlying generative technology has changed, and so the defense must change as well. Therefore, the choice of training data for an anti-spoofing model is as important, if not more so, than the model's architecture.

Table 4.1: Analysis of Voice Anti-Spoofing Datasets

Dataset Name	Year(s)	Focus (LA/PA/Both)	Key Attack Types Included	Significance
ASVspoof 2019	2019	Both	LA: WaveNet, WaveRNN-based VC/TTS. PA: Simulated replay attacks.[33]	A foundational benchmark, but its LA attacks are now considered somewhat dated. Models trained only on this may be vulnerable to newer fakes.
ASVspoof 2021	2021	Both + Deepfake (DF)	Extended 2019 attacks with more variability and channel degradation. DF track focused solely on deepfake detection.[38]	Increased complexity and realism over the 2019 version. A better source for training robust models.
CodecFake	2024+	LA	Deepfakes from	The most

		(Codec-based)	modern neural audio codecs (e.g., EnCodec, SoundStream).[39]	relevant dataset for defending against state-of-the-art synthetic voice attacks. Essential for a high-security system.
ReMASC	2021	PA (Replay)	Replay attacks recorded in realistic environments (indoor, outdoor, in-vehicle) with various microphones.[15]	A high-quality, realistic dataset specifically for training and evaluating replay attack countermeasures.

With this understanding of the threat landscape, an appropriate model architecture can be selected.

Table 4.2: Comparison of Anti-Spoofing Model Architectures

Architecture	Input Type	Key Principle	Pros	Cons	Relevant Repositories/Models	
RawNet2	Raw Waveform	End-to-end learning directly from raw audio samples using sinc-convolutional layers to learn filterbanks.[41]	Captures subtle phase and fine-grained temporal information missed by spectrogram-based methods. Proven high performance in	Can be more computationally intensive. Requires careful implementation.	eurecom-asp/rawnet2-antispoofing [41],	Jungjee/RawNet [42]

			ASVspoof challenges.[38]			
Audio Spectrogram Transformer (AST)	Spectrogram	Applies the Transformer architecture (self-attention) to patches of an audio spectrogram, treating it like an image.[43]	Excellent at capturing long-range dependencies in the spectro-temporal domain. Leverages powerful, pre-trained vision transformer backbones.	May lose some fine-grained phase information during the spectrogram conversion.	MIT/ast-finetuned-audioset-10-10-0.4593 (base model) [44],	WpythonW/ast-fakeaudio-detector (fine-tuned) [44]
AASIST	Raw Waveform + Graph	An advanced architecture that models speech as a graph of spectro-temporal features and uses graph attention networks.[33]	State-of-the-art performance by explicitly modeling relationships between frequency bands and time frames.	Highly complex architecture, steeper learning curve to implement and train from scratch.	KORALLLL/AASIST_SCALING [33]	

Model Recommendation and Implementation (LA):

For a balance of high performance and ease of implementation, using a pre-trained **Audio Spectrogram Transformer (AST)** is the recommended starting point. The WpythonW/ast-fakeaudio-detector model on Hugging Face is fine-tuned specifically for this binary classification task and can be used directly with the transformers library pipeline.[44]

Python

```python
from transformers import pipeline
import torch

# Initialize the audio classification pipeline with the fine-tuned AST model
# Use device=0 for GPU acceleration
device = 0 if torch.cuda.is_available() else -1
spoof_detector = pipeline(
    "audio-classification",
    model="WpythonW/ast-fakeaudio-detector",
    device=device
)

def detect_logical_spoof(audio_file_path):
    """
   Detects if an audio file is a synthetic deepfake (LA attack).
   Returns a dictionary with the prediction and confidence score.
   """
    try:
        predictions = spoof_detector(audio_file_path)

        # The model outputs probabilities for 'fake' and 'real'
        is_spoof = False
        confidence = 0.0

        for p in predictions:
            if p['label'] == 'fake':
                is_spoof = True
                confidence = p['score']
                break # Assuming 'fake' label exists

        # If the highest score was for 'real', adjust the values
        if not is_spoof:
            for p in predictions:
                if p['label'] == 'real':
```

```
                confidence = p['score']
                break

        return {
            "isSpoof": is_spoof,
            "confidence": confidence
        }

    except Exception as e:
        print(f"Error during deepfake detection: {e}")
        return {"isSpoof": True, "confidence": 0.0} # Fail safe

# Example usage:
# result = detect_logical_spoof("/path/to/auth_attempt.wav")
# print(result)
```

For maximum security, a more advanced approach would involve fine-tuning a RawNet2 model using a combination of the ASVspoof 2021 and CodecFake datasets. This would provide the highest level of protection against the most modern attacks but represents a significantly larger engineering effort.

4.3. Replay Attack Detection (PA Countermeasure)

Replay attacks introduce distinct acoustic artifacts that are different from those of synthetic speech. These include [14]:

- **Device Distortion:** Imperfections from the playback device's speaker and the recording device's microphone.
- **Environmental Acoustics:** Reverberation and background noise from the room where the replay occurs.
- **Digital-to-Analog-to-Digital Conversion:** The process of playing and re-recording can introduce subtle electronic noise and frequency filtering.

While a general-purpose anti-spoofing model might detect some replay attacks, a specialized model trained on a PA-focused dataset like ASVspoof or ReMASC will be far more effective. The eurecom-asp/rawnet2-antispoofing repository provides the code necessary to train a RawNet2 model on the ASVspoof 2019 PA dataset, which would serve as a robust PA countermeasure.[41]

4.4. Integrated Anti-Spoofing Strategy

A single anti-spoofing model is a single point of failure. A more resilient strategy is to use **two separate, specialized models** in sequence within the custom backend.

1. **Run the LA Countermeasure:** First, process the audio with the deepfake detection model (e.g., the AST model). If it is flagged as synthetic, the entire authentication process fails immediately.
2. **Run the PA Countermeasure:** If the audio passes the LA check, then process it with the replay attack detection model (e.g., a trained RawNet2 PA model). If it is flagged as a replay, the process fails.
3. **Pass:** Only if the audio passes *both* checks is it considered "live" and authentic.

This defense-in-depth approach is highly effective. Even if an attacker creates a perfect deepfake that fools the LA model, they still must play it over a speaker to inject it into the phone call. This act of physical playback introduces the very artifacts that the PA model is trained to detect. This layering significantly raises the bar for a successful attack.

Section 5: The Final Checkpoint: Multi-Factor Authentication (MFA) Integration

After the system has successfully verified the user's identity and the authenticity of their voice, a final security gate is required to confirm the user's intent actively. This is achieved through Multi-Factor Authentication (MFA), typically via a push notification to a trusted, pre-registered device. This step protects against scenarios where an attacker might coerce a legitimate user into speaking their passphrase.

5.1. Choosing an MFA Provider: Commercial vs. Open Source

The choice of MFA provider is a significant architectural decision, balancing cost, control, and development complexity.

- **Commercial (SaaS):** Services like Duo Security provide a fully managed, highly reliable MFA platform with excellent developer documentation and support. While they incur a per-user, per-month cost, they abstract away the complexity of infrastructure management, message delivery, and security maintenance.[46]
- **Open Source:** Solutions like Keycloak and Authelia offer a free-to-use, self-hosted alternative. They provide immense flexibility and control but come with the significant operational overhead of deploying, maintaining, scaling, and securing the IAM infrastructure yourself.[48]

A critical distinction for this project is the primary use case of these tools. Open-source IAM platforms like Keycloak and Authelia are primarily designed to act as centralized identity providers for web applications, managing user-facing login flows (SSO, etc.).[50] Their APIs are structured around this paradigm. In contrast, Duo's Auth API is specifically designed for the exact use case required here: programmatically triggering an out-of-band authentication from a backend service.[7] Attempting to force an open-source IAM tool into this role would be working against its intended design, likely resulting in a more complex and brittle integration.

Table 5.1: Comparison of MFA Solutions (Duo vs. Open Source)

Feature	Duo Security	Keycloak	Authelia
Hosting Model	Cloud (SaaS)	Self-Hosted	Self-Hosted
Cost	Per user/month [47]	Free (Infrastructure/Ops cost)	Free (Infrastructure/Ops cost)
Primary Use Case	Enterprise MFA for apps, VPNs, servers	Full-featured IAM/SSO provider	SSO and 2FA gateway for reverse proxies
Programmatic Trigger API	Yes (Auth API) [20]	No direct endpoint; requires custom SPI [52]	No; designed for proxy integration [53]

Ease of Integration	High (Purpose-built API)	Low (Requires extensive custom dev)	Low (Not designed for this use case)
Maintenance Overhead	Low (Managed by Duo)	High (Self-managed)	High (Self-managed)

5.2. Recommended Approach: Duo Security

For this project, **Duo Security is the strongly recommended MFA provider.** The Duo Auth API is purpose-built for this server-initiated workflow, making the integration significantly simpler, faster, and more reliable than attempting to build a custom solution with an open-source IAM platform.

Setup Process:

1. **Create a Duo Account:** Sign up for a Duo account. A free trial is available which includes API access.[20]
2. **Create an Application:** In the Duo Admin Panel, navigate to "Applications" and protect a new application. Select the "Auth API" type.[7]
3. **Obtain Credentials:** Upon creation, Duo will provide three crucial pieces of information: an **Integration Key (ikey)**, a **Secret Key (skey)**, and an **API Hostname**. These must be stored securely as credentials in the n8n workflow or the backend service.[55]

5.3. Implementation Guide: Triggering a Duo Push

Triggering a Duo push notification is an asynchronous, multi-step process that requires careful handling in the backend service. It is not a single fire-and-forget API call.

The API Flow:

1. **/auth/v2/preauth (POST):** Before attempting to authenticate, the backend must first call the preauth endpoint. This call verifies that the user exists in Duo, is permitted to use the application, and returns a list of their enrolled authentication

devices and their capabilities (e.g., push, sms, phone).[8] This step is vital to ensure the user can actually receive a push notification before one is sent.

2. **/auth/v2/auth (POST):** If the preauth call is successful and confirms the user has a push-capable device, the backend then calls the /auth endpoint.
   - **Parameters:** username, factor=push, and device=auto are sent.
   - **Asynchronous Flag:** Crucially, the parameter async=1 must be included. This tells the Duo API to return a response immediately without waiting for the user to approve the push. The response will contain a txid (transaction ID).[8]
3. **/auth/v2/auth_status (GET):** The backend must now use the txid to poll the auth_status endpoint to determine the outcome of the push notification.
   - The backend should poll this endpoint periodically (e.g., every 1-2 seconds).
   - The initial responses will have a result of "waiting".
   - Once the user interacts with the push notification, the result will change to "allow" (if approved) or "deny" (if denied).
   - The backend must implement a timeout (e.g., 60 seconds) to handle cases where the user does not respond at all, treating it as a denial.[8]

Code Example (Python using duo_client_python):

Python

```python
import duo_client
import time
import os

# Load credentials from environment variables or a secure store
IKEY = os.getenv("DUO_IKEY")
SKEY = os.getenv("DUO_SKEY")
HOST = os.getenv("DUO_HOST")

# Create the Auth API client
auth_api = duo_client.Auth(ikey=IKEY, skey=SKEY, host=HOST)

def trigger_duo_push_and_wait(username):
    """
    Triggers a Duo push notification and polls for the result.
    """
    try:
```

```
    # Step 1: Pre-authentication
    preauth_response = auth_api.preauth(username=username)
    if preauth_response['result']!= 'auth':
        print(f"Duo preauth failed: {preauth_response['status_msg']}")
        return {"status": "failure", "reason": "User not permitted or enrolled."}

    # Check for push-capable devices
    has_push_device = any(
        'push' in device.get('capabilities',)
        for device in preauth_response.get('devices',)
    )
    if not has_push_device:
        return {"status": "failure", "reason": "No push-capable device found."}

    # Step 2: Trigger the asynchronous push
    auth_response = auth_api.auth(
        username=username,
        factor='push',
        device='auto',
        async_txn=True  # Equivalent to async=1
    )

    if auth_response['result']!= 'allow':
        return {"status": "failure", "reason": "Failed to send Duo push."}

    txid = auth_response['txid']

    # Step 3: Poll for the result
    start_time = time.time()
    timeout_seconds = 60

    while time.time() - start_time < timeout_seconds:
        status_response = auth_api.auth_status(txid=txid)

        if status_response['result'] == 'allow':
            return {"status": "success", "reason": "MFA approved."}
        elif status_response['result'] == 'deny':
            return {"status": "failure", "reason": "MFA denied."}
            # Wait before polling again
            time.sleep(2)

        return {"status": "failure", "reason": "MFA timed out."}

    except Exception as e:
        print(f"An error occurred with Duo API: {e}")
        return {"status": "failure", "reason": "Duo API error."}

# Example usage:
# result = trigger_duo_push_and_wait("user-123")
# print(result)
```

This entire polling loop should be encapsulated within the custom backend service. The service's API endpoint will only return a final success or failure response to the n8n workflow once the Duo transaction is complete.

Section 6: Orchestration and Data Management with n8n

With the core security components designed, n8n serves as the high-level orchestrator, connecting these services and executing the business logic of the authentication flow. This section details the construction of the n8n workflow and the process for generating the dummy data that users will access.

6.1. Designing the n8n Workflow

The n8n workflow visually represents the state machine of the authentication process. Its node-based interface makes the complex, multi-service interaction transparent, manageable, and easy to debug.[56] Each node represents a state or action, and the connections represent the transitions based on the outcomes of those actions.

The workflow will be structured as follows:

1. **Webhook Trigger:** Receives the initial tool-call from VAPI.
2. **HTTP Request (to Security Backend):** Sends the audio for analysis.

3. **IF Node:** Checks if voice verification and anti-spoofing passed.
   - **True Path:** Continues to MFA.
   - **False Path:** Terminates and sends a failure response.
4. **HTTP Request (to MFA Backend Logic):** Calls a backend endpoint that encapsulates the Duo polling logic.
5. **IF Node:** Checks if MFA was successful.
   - **True Path:** Continues to data retrieval.
   - **False Path:** Terminates and sends a failure response.
6. **HTTP Request (to Data Store):** Fetches the user's data.
7. **Respond to Webhook:** Sends the final success or failure message back to VAPI.

6.2. Node-by-Node Configuration

Here is a detailed configuration guide for each key node in the n8n canvas.

- **1. Webhook Trigger Node:**
  - **Authentication:** Set to None.
  - **HTTP Method:** Set to POST.
  - **Path:** A unique path will be generated (e.g., webhook/voice-auth). This full URL is what must be configured as the "Server URL" for the verifyPassword tool in VAPI.[58]
  - When listening for a test event, this node will capture the JSON payload sent by VAPI, making its data available to subsequent nodes.
- **2. HTTP Request Node (Call Security Backend):**
  - **URL:** The endpoint of the custom Python/Flask backend service (e.g., http://localhost:5000/verify).
  - **Method:** POST.
  - **Send Body:** true.
  - **Body Content Type:** JSON.
  - **Body:** Use expressions to construct the JSON payload from the webhook data.

    JSON
    ```json
    {
      "userId": "{{ $json.body.message.call.metadata.userId }}",
      "audioUrl": "{{ $json.body.message.recordingUrl }}",
      "claimedPhoneNumber": "{{ $json.body.message.call.phoneCall.from }}"
    }
    ```

  (Note: The exact paths depend on the final VAPI payload structure. The userId might need to be passed as metadata in the initial VAPI call.)
  - **Authentication:** Header Auth. Configure an API key credential that matches what the backend service expects for authorization.[59]
- **3. IF Node (Security Check):**
  - This node branches the workflow based on the backend's response.
  - **Condition:** Add a condition to check for a successful response.
    - **Value 1:** {{ $json.status }}
    - **Operation:** Equal
    - **Value 2:** success
  - The true output will connect to the MFA node. The false output will connect to a Respond to Webhook node configured with a failure message.
- **4. HTTP Request Node (Trigger MFA):**
  - **URL:** The backend endpoint that handles the Duo logic (e.g., http://localhost:5000/trigger-mfa).
  - **Method:** POST.
  - **Body:**
    ```JSON
    {
      "userId": "{{ $('Webhook').item.json.body.message.call.metadata.userId }}"
    }
    ```
- **5. IF Node (MFA Check):**
  - Checks the response from the MFA trigger endpoint.
  - **Condition:**
    - **Value 1:** {{ $json.status }}
    - **Operation:** Equal
    - **Value 2:** success
  - The true output connects to the data retrieval node. The false output connects to a different failure response node.
- **6. HTTP Request Node (Data Store):**
  - **URL:** The endpoint that serves the dummy data (e.g., http://localhost:5001/data/{{ $('Webhook').item.json.body.message.call.metadata.userId }}).
  - **Method:** GET.
- **7. Respond to Webhook Node:**
  - This node sends a response back to VAPI's synchronous tool call, which VAPI then converts to speech for the user.

- There should be multiple Respond to Webhook nodes, one for each terminal state of the workflow.
- **Success Response (after Data Store):**
  - **Response Body (JSON):** {"result": "Authentication successful. I am now retrieving your records."}
- **Failure Response (after Security Check):**
  - **Response Body (JSON):** {"result": "Authentication failed. Your voice could not be verified. Please try again."}
- **Failure Response (after MFA Check):**
  - **Response Body (JSON):** {"result": "Authentication was not approved on your device. Please try again."}

6.3. Generating Dummy Data

To test the final stage of the workflow, a simple data store with mock sensitive data is required. The Python library **Faker** is an excellent tool for generating large amounts of realistic-looking test data.[61]

Generating Dummy Medical Data

This script uses Faker and its community providers to generate a JSON file of patient records.[63]

Python

```python
import json
import random
from faker import Faker

# Initialize Faker
fake = Faker()
```

```
def generate_diagnoses():
    diagnoses =
    return random.sample(diagnoses, random.randint(1, 3))

def generate_prescriptions():
    medicines = [
        {"medicine": "Lisinopril", "dosage": "10mg"},
        {"medicine": "Metformin", "dosage": "500mg"},
        {"medicine": "Albuterol Inhaler", "dosage": "90mcg"},
    ]
    return random.sample(medicines, random.randint(1, 2))

def generate_patient_record(patient_id):
    return {
        "patient_id": f"user-{patient_id}",
        "name": fake.name(),
        "dob": fake.date_of_birth(minimum_age=18, maximum_age=90).isoformat(),
        "address": fake.address().replace("\n", ", "),
        "recent_visit": {
            "date": fake.date_between(start_date="-1y", end_date="today").isoformat(),
            "doctor": f"Dr. {fake.last_name()}",
            "diagnoses": generate_diagnoses(),
            "prescriptions": generate_prescriptions()
        }
    }

# Generate 100 patient records
patients_data = [generate_patient_record(i) for i in range(100)]

# Save to a JSON file
with open("dummy_medical_data.json", "w") as f:
    json.dump(patients_data, f, indent=2)

print("Generated dummy_medical_data.json")
```

Generating Dummy Financial Data

This script uses Faker to generate a list of financial transactions.[65]

Python

```python
import json
import random
from faker import Faker
from datetime import datetime, timedelta

fake = Faker()

def generate_financial_record(user_id):
    transactions =
    for _ in range(random.randint(5, 20)):
        transaction = {
            "transaction_id": fake.uuid4(),
            "date": fake.date_time_between(start_date="-90d", end_date="now").isoformat(),
            "merchant": fake.company(),
            "amount": round(random.uniform(-500.0, 2000.0), 2),
            "currency": "USD",
            "category": fake.bs().split(' ') # Use a business buzzword as category
        }
        transactions.append(transaction)

    return {
        "user_id": f"user-{user_id}",
        "account_number": fake.bban(),
        "balance": round(random.uniform(1000.0, 50000.0), 2),
        "transactions": sorted(transactions, key=lambda x: x['date'], reverse=True)
    }

# Generate 100 financial records
financial_data = [generate_financial_record(i) for i in range(100)]

# Save to a JSON file
with open("dummy_financial_data.json", "w") as f:
    json.dump(financial_data, f, indent=2)
```

```
print("Generated dummy_financial_data.json")
```

These generated JSON files can be served by a simple Python Flask or FastAPI server, which will act as the "Data Store" that n8n queries in the final step of the workflow.

Section 7: Conclusion and Strategic Recommendations

This report has outlined a comprehensive architectural design for a multi-layered voice authentication system. By integrating best-in-class services for telephony, conversational AI, security, and orchestration, the proposed system achieves a robust defense-in-depth posture suitable for protecting sensitive medical or financial data.

7.1. Recommended Technology Stack Summary

The analysis concludes with the following technology stack recommendation, balancing state-of-the-art performance with practical implementation and maintainability:

- **Telephony Gateway: Twilio**, configured with Elastic SIP Trunking for production environments to ensure scalability and cost-efficiency.[2]
- **Conversational AI: VAPI** with an **OpenAI GPT-4o** model, using Tool Calling to integrate with the backend orchestration layer.[18]
- **Orchestration: n8n**, serving as the central workflow engine to manage the sequence of API calls and conditional logic.[56]
- **Custom Security Backend:** A **Python** microservice (using Flask or FastAPI) to host the AI/ML models.
    - **Speaker Verification: SpeechBrain** with the speechbrain/spkrec-ecapa-voxceleb model for its state-of-the-art accuracy.[11]
    - **Anti-Spoofing:** A dual-model strategy is critical.
        - **Deepfake Detection (LA):** An **Audio Spectrogram Transformer (AST)** model fine-tuned for fake audio detection, such as WpythonW/ast-fakeaudio-detector.[44]
        - **Replay Detection (PA):** A **RawNet2** model trained on a physical access

dataset like ASVspoof 2019 PA.[41]

- **Multi-Factor Authentication: Duo Security**, utilizing its purpose-built Auth API for programmatic, server-side MFA triggers.[20]
- **Data Generation:** The **Faker** library in Python for creating realistic dummy datasets for testing and demonstration.[61]

7.2. Security Posture and Risk Mitigation

The strength of this architecture lies in its layered, sequential gating model. An attacker must successfully bypass four distinct and technologically diverse security checks: voice biometrics, synthetic voice detection, replay attack detection, and out-of-band MFA.

However, the threat landscape for voice AI is not static. The continuous improvement of voice synthesis and conversion models means that anti-spoofing countermeasures must be considered a moving target. To maintain a strong security posture over time, it is essential to:

- **Monitor the Research Community:** Keep abreast of the findings from initiatives like the ASVspoof challenge series and new datasets like CodecFake. These are the primary sources of information on emerging attack vectors.[13]
- **Periodically Update Models:** The anti-spoofing models, in particular, should be periodically re-evaluated and potentially retrained or replaced with newer models that have been trained on more recent and sophisticated spoofing datasets.

7.3. Scalability and Performance Considerations

The primary performance bottleneck in this system will be the Custom Security Backend, as AI model inference is computationally intensive. The end-to-end latency of the verification process directly impacts the user experience, as the user will be on hold waiting for a result.

To ensure low latency and high concurrency, the following strategies should be employed:

- **GPU Acceleration:** Running the PyTorch models for speaker verification and

anti-spoofing on a GPU-enabled server will dramatically reduce inference time compared to a CPU.
- **Containerization and Scalable Deployment:** The backend service should be containerized (e.g., using Docker) and deployed on a scalable platform like Kubernetes or a serverless environment (e.g., AWS Lambda with container support). This allows the system to automatically scale the number of inference workers to handle fluctuating call volumes.

7.4. Future Enhancements

The proposed architecture provides a strong foundation that can be extended with more advanced capabilities in the future.

- **Continual Learning:** Implement a feedback loop for the anti-spoofing models. When a call is flagged as a potential spoof and rejected, the audio can be saved for manual review. Confirmed spoofing attempts can be collected into a proprietary dataset used to periodically fine-tune and improve the detection models.
- **Adaptive Authentication:** The n8n workflow can be enhanced with more sophisticated logic to implement adaptive authentication. For example, the system could analyze the context of the user's request (e.g., retrieved via VAPI's natural language understanding). A low-risk request, like checking an appointment time, might only require passing the voice biometric gate. A high-risk request, like authorizing a financial transaction, would trigger the full cascade of security checks, including MFA.
- **Multi-Lingual Support:** To support users in different languages, the system can be extended by incorporating a language identification model at the beginning of the security backend's process, such as speechbrain/lang-id-voxlingua107-ecapa.[68] Based on the detected language, the workflow could then route the request to language-specific speaker verification and anti-spoofing models.

Works cited

1. Call Handling with Vapi and Twilio, accessed August 2, 2025, https://docs.vapi.ai/calls/call-handling-with-vapi-and-twilio
2. Twilio SIP Integration - Vapi, accessed August 2, 2025, https://docs.vapi.ai/advanced/sip/twilio

3. Vapi Server SDK, accessed August 2, 2025, https://vapi.ai/community/m/1389128510433398795
4. Vapi: Introduction, accessed August 2, 2025, https://docs.vapi.ai/quickstart/introduction
5. Twilio Trigger integrations | Workflow automation with n8n, accessed August 2, 2025, https://n8n.io/integrations/twilio-trigger/
6. No Code Messaging Automation With Twilio & N8N Cloud, accessed August 2, 2025, https://n8n-automation.com/2024/02/24/messaging-automation-with-twilio/
7. Duo Auth API v1 (Legacy), accessed August 2, 2025, https://duo.com/docs/authapi-v1
8. Duo Two-Factor Authentication API for Applications, accessed August 2, 2025, https://duo.com/docs/authapi-guide
9. Phone calls - Vapi, accessed August 2, 2025, https://docs.vapi.ai/quickstart/phone
10. Exploring Speaker Recognition Projects with Python: A Comprehensive Guide - EdgeOps.AI, accessed August 2, 2025, https://www.edgeops.ai/post/exploring-speaker-recognition-projects-with-python-a-comprehensive-guide
11. speechbrain/spkrec-ecapa-voxceleb · Hugging Face, accessed August 2, 2025, https://huggingface.co/speechbrain/spkrec-ecapa-voxceleb
12. YuanGongND/efficient-voice-antispoof - GitHub, accessed August 2, 2025, https://github.com/YuanGongND/efficient-voice-antispoof
13. (PDF) ASVspoof: The Automatic Speaker Verification Spoofing and Countermeasures Challenge - ResearchGate, accessed August 2, 2025, https://www.researchgate.net/publication/313836587_ASVspoof_The_Automatic_Speaker_Verification_Spoofing_and_Countermeasures_Challenge
14. Audio Replay Attacks and Countermeasures Against Them - Antispoofing Wiki, accessed August 2, 2025, https://antispoofing.org/audio-replay-attacks-and-countermeasures-against-them/
15. Detecting Replay Attack on Voice-Controlled Systems using Small Neural Networks - Fadi A. Aloul, accessed August 2, 2025, https://www.aloul.net/Papers/faloul_rtsi2022.pdf
16. Server URLs - Vapi, accessed August 2, 2025, https://docs.vapi.ai/server-url
17. Server events - Vapi, accessed August 2, 2025, https://docs.vapi.ai/server-url/events
18. Custom Tools | Vapi, accessed August 2, 2025, https://docs.vapi.ai/tools/custom-tools
19. Server events | Vapi, accessed August 2, 2025, https://docs.vapi.ai/server-url/events#function-call
20. Duo Auth API, accessed August 2, 2025, https://duo.com/docs/authapi
21. Add Function and Tool Calling to a Twilio Voice OpenAI Integration, accessed August 2, 2025, https://www.twilio.com/en-us/blog/add-function-tool-calling-twilio-voice-openai-integration

22. Web calls | Vapi, accessed August 2, 2025, https://docs.vapi.ai/quickstart/web
23. Build Conversational AI Apps with Twilio and the OpenAI Realtime API, accessed August 2, 2025, https://www.twilio.com/en-us/blog/twilio-openai-realtime-api-launch-integration
24. VAPI Server URLs & Function Calls Guide: Building a Calendly Booking Agent - YouTube, accessed August 2, 2025, https://www.youtube.com/watch?v=-W9PlhA7nnY
25. voice-recognition · GitHub Topics, accessed August 2, 2025, https://github.com/topics/voice-recognition
26. speechbrain/speechbrain: A PyTorch-based Speech Toolkit - GitHub, accessed August 2, 2025, https://github.com/speechbrain/speechbrain
27. SpeechBrain: Open-Source Conversational AI for Everyone, accessed August 2, 2025, https://speechbrain.github.io/
28. Using SpeechBrain at Hugging Face, accessed August 2, 2025, https://huggingface.co/docs/hub/speechbrain
29. How to Build a Speaker Identification System for Recorded Online Meetings - Gladia, accessed August 2, 2025, https://www.gladia.io/blog/build-a-speaker-identification-system-for-online-meetings
30. speechbrain/spkrec-xvect-voxceleb - Hugging Face, accessed August 2, 2025, https://huggingface.co/speechbrain/spkrec-xvect-voxceleb
31. pyannote/pyannote-audio: Neural building blocks for ... - GitHub, accessed August 2, 2025, https://github.com/pyannote/pyannote-audio
32. yangwang825/ecapa-tdnn-vox2 - Hugging Face, accessed August 2, 2025, https://huggingface.co/yangwang825/ecapa-tdnn-vox2
33. Daily Papers - Hugging Face, accessed August 2, 2025, https://huggingface.co/papers?q=ASVspoof%202019
34. Daily Papers - Hugging Face, accessed August 2, 2025, https://huggingface.co/papers?q=So-Fake-Set
35. CodecFake+: A Large-Scale Neural Audio Codec-Based Deepfake Speech Dataset - arXiv, accessed August 2, 2025, https://arxiv.org/html/2501.08238v2
36. CodecFake-Omni: A Large-Scale Codec-based Deepfake Speech Dataset - arXiv, accessed August 2, 2025, https://arxiv.org/html/2501.08238v1
37. [2406.08112] Codecfake: An Initial Dataset for Detecting LLM-based Deepfake Audio - arXiv, accessed August 2, 2025, https://arxiv.org/abs/2406.08112
38. ASVspoof 2021 Baseline Systems - GitHub, accessed August 2, 2025, https://github.com/asvspoof-challenge/2021
39. The Codecfake Dataset and Countermeasures for the Universally Detection of Deepfake Audio | Request PDF - ResearchGate, accessed August 2, 2025, https://www.researchgate.net/publication/387802678_The_Codecfake_Dataset_and_Countermeasures_for_the_Universally_Detection_of_Deepfake_Audio
40. [2406.07237] CodecFake: Enhancing Anti-Spoofing Models Against Deepfake Audios from Codec-Based Speech Synthesis Systems - arXiv, accessed August 2, 2025, https://arxiv.org/abs/2406.07237
41. eurecom-asp/rawnet2-antispoofing: This repository ... - GitHub, accessed August

2, 2025, https://github.com/eurecom-asp/rawnet2-antispoofing
42. Jungjee/RawNet: Official repository for RawNet, RawNet2, and RawNet3 - GitHub, accessed August 2, 2025, https://github.com/Jungjee/RawNet
43. Pre-trained models and datasets for audio classification - Hugging Face Audio Course, accessed August 2, 2025, https://huggingface.co/learn/audio-course/chapter4/classification_models
44. WpythonW/ast-fakeaudio-detector - Hugging Face, accessed August 2, 2025, https://huggingface.co/WpythonW/ast-fakeaudio-detector
45. RawGAT-ST-antispoofing/README.md at main - GitHub, accessed August 2, 2025, https://github.com/eurecom-asp/RawGAT-ST-antispoofing/blob/main/README.md
46. Top 8 Duo Security Alternatives For IT Teams To Try In 2025 - Zluri, accessed August 2, 2025, https://www.zluri.com/blog/duo-security-alternatives
47. Best MFA Alternative to Duo Security in 2024 - Rublon, accessed August 2, 2025, https://rublon.com/blog/duo-security-best-mfa-alternatives/
48. Compare 10 Open Source MFA Tools in 2025 - AIMultiple, accessed August 2, 2025, https://aimultiple.com/open-source-mfa
49. Authelia | Free Open-Source Software Modern IAM Solution, accessed August 2, 2025, https://www.authelia.com/
50. MFA Authentication with Keycloak | Cloud-IAM | DOCS, accessed August 2, 2025, https://documentation.cloud-iam.com/resources/keycloak-authentications/mfa.html
51. authelia/authelia: The Single Sign-On Multi-Factor portal for web apps, now OpenID Certified - GitHub, accessed August 2, 2025, https://github.com/authelia/authelia
52. Server Developer Guide - Keycloak, accessed August 2, 2025, https://www.keycloak.org/docs/latest/server_development/index.html#_restful_api
53. Get started | Integration - Authelia, accessed August 2, 2025, https://www.authelia.com/integration/prologue/get-started/
54. Duo Admin API, accessed August 2, 2025, https://duo.com/docs/adminapi
55. MFA for Application Developers and Administrators, accessed August 2, 2025, https://it.ucsb.edu/mfa/mfa-application-developers-and-administrators
56. Best apps & software integrations | n8n, accessed August 2, 2025, https://n8n.io/integrations/
57. How to Integrate an API into N8N? - Reddit, accessed August 2, 2025, https://www.reddit.com/r/n8n/comments/1klppwb/how_to_integrate_an_api_into_n8n/
58. Webhook and OpenAI: Automate Workflows with n8n, accessed August 2, 2025, https://n8n.io/integrations/webhook/and/openai/
59. n8n Tutorial: How To Use a Webhook or Catch hook - Flexxable, accessed August 2, 2025, https://flexxable.com/how-to-create-a-webhook-catch-hook-with-n8n/
60. HTTP Request node documentation - n8n Docs, accessed August 2, 2025, https://docs.n8n.io/integrations/builtin/core-nodes/n8n-nodes-base.httprequest/

61. joke2k/faker: Faker is a Python package that generates fake data for you. - GitHub, accessed August 2, 2025, https://github.com/joke2k/faker
62. Python in Excel: How to generate fake data with Faker - Stringfest Analytics, accessed August 2, 2025, https://stringfestanalytics.com/python-in-excel-how-to-generate-fake-data-with-faker/
63. faker-healthcare-system - PyPI, accessed August 2, 2025, https://pypi.org/project/faker-healthcare-system/
64. Hands-On Coding — Generating Fake Medical Data Using Python Faker - Medium, accessed August 2, 2025, https://medium.com/@sounder.rahul/hands-on-code-generating-fake-medical-data-using-python-faker-d61fdc9c9fe9
65. Fake Financial Transactions with Faker - Python-Fiddle, accessed August 2, 2025, https://python-fiddle.com/examples/faker-financial-transactions
66. Generate Unlimited Fake Data Using Python - Kaggle, accessed August 2, 2025, https://www.kaggle.com/code/adnananam/generate-unlimited-fake-data-using-python
67. CodecFake+: A Large-Scale Neural Audio Codec-Based Deepfake Speech Dataset - arXiv, accessed August 2, 2025, https://arxiv.org/abs/2501.08238
68. speechbrain/lang-id-voxlingua107-ecapa - Hugging Face, accessed August 2, 2025, https://huggingface.co/speechbrain/lang-id-voxlingua107-ecapa