Comparative Analysis of Cloud-Based and Local Device Deployment for AI Services

Alanoud Almuhanna, Sara Alaiban, Afnan Alanazi, Ghala Alangari
443200858@student.ksu.edu.sa,
443200572@student.ksu.edu.sa,443201089@student.ksu.edu.sa,
443200823@student.ksu.edu.sa

King Saud University
Software Engineering Department
SWE486 - Cloud Computing and Big Data
Group 4, Section 54979
Under Supervision of: Ms. Mona Hakami

Introduction

Artificial Intelligence services, such as Computer Vision, Natural Language Processing, and Speech Recognition, have become essential in modern applications. These AI services can be deployed either locally on personal devices or through cloud-based platforms, each offering different trade-offs in performance, cost, and scalability. Understanding such trade-offs is crucial for selecting the best deployment strategy based on the application's needs[1].

This project aims to compare the performance of an AI service when deployed locally versus in the cloud. The key focus is on measuring inference time, response latency, and resource usage to evaluate the efficiency of each deployment method.

1. Selecting AI Service

We chose Speech Recognition as our AI service because of its strong relevance and technical value. It is widely used in areas such as customer service through call centers and chatbots, assisting individuals with disabilities, and facilitating healthcare tasks like medical transcription. With the rise of voice assistants like Siri and Alexa, speech-to-text technology has become an essential tool for improving user interaction and accessibility, making it a valuable area of study[2].

Additionally, our team is particularly interested in how AI models handle challenges like background noise, different accents, and varying speech speeds. By exploring speech recognition technology, we aim to gain insights into its capabilities and evaluate its performance in different environments.

Business Questions

This section explores key business questions focused on the performance of local (Whisper) and cloud (Azure STT) speech recognition.

1. How does the **inference time** of Whisper and Azure STT compare across different audio lengths?

2. How does the **response time** vary between Whisper (local) and Azure STT (cloud) for short versus long audio inputs?

3. How does **network latency** affect the overall response time in Azure STT for real-time speech applications?

4. What is the **CPU usage** and **memory consumption** required by Whisper during local inference, and how does this impact hardware suitability?

5. Based on performance metrics, which deployment (local or cloud) is more **scalable and practical** for different real-world usage scenarios?

2. Setting Up Local Deployment (Whisper)

OpenAI developed the open-source automatic speech recognition (ASR) system called Whisper. It excels at transforming speech into text for numerous languages and accents, as it was trained on a large dataset of multilingual and multitask supervised learning. Voice based search, language learning tools, transcription services, and accessibility features are just a few examples of the numerous applications that utilize Whisper. [3] [4]

Why We Chose Whisper

Whisper was chosen for its accuracy, scalability, and ease of deployment. It was pretrained on a diverse dataset, enabling robust recognition across different languages and range of audio environments. The model is also very scalable, it provides various sizes—Tiny, Small, Base, Medium, and Large—to balance speed, and accuracy. Moreover, its smooth integration through the Whisper Python library simplifies deployment, allowing for efficient speech recognition without requiring complex configurations.[3]

Whisper Models Comparison

As we mentioned before, Whisper offers a number of pretrained models of varying sizes, each balancing speed, accuracy, and resource consumption. We tested the tiny, small, base and medium models on various audio lengths and measured inference time, response time, CPU usage, and memory consumption.

- Inference Time (seconds) for Different Whisper Models and Audio Lengths

Model	2 sec	10 sec	20 sec	30 sec	34 sec	45 sec	1 min	2 min
Tiny	1.04	1.10	1.32	1.83	2.23	2.30	2.95	7.29
Small	4.90	6.25	7.51	10.86	13.15	14.20	17.34	36.33
Base	1.30	1.74	2.00	2.85	3.04	3.87	4.95	14.03
Medium	14.57	17.13	18.38	21.23	32.42	33.21	39.58	101.55

Table 1: Inference Time (in seconds) for Different Models and Audio Lengths

Observations: Inference time usually increases with audio length across all Whisper models tested. The Tiny model demonstrates the fastest processing times, while the Medium model exhibits the slowest, highlighting the performance trade-off associated with model size.

- Response Time (seconds) for Different Whisper Models and Audio Lengths

Model	2 sec	10 sec	20 sec	30 sec	34 sec	45 sec	1 min	2 min
Tiny	2.30	1.19	1.42	1.92	2.34	2.41	3.07	7.41
Small	4.98	6.33	7.59	10.96	13.24	14.28	17.45	36.45
Base	1.34	1.81	2.06	2.93	3.12	3.95	5.05	14.10
Medium	14.82	17.20	18.45	21.31	32.51	33.29	39.67	101.63

Table 2: Response Time (seconds) for Different Models and Audio Lengths

Observations: Response time generally increases with audio length across all Whisper models tested. The Tiny model exhibits the fastest response times, while the Medium model exhibits the slowest, highlighting the performance trade-off associated with model size.

- CPU Usage Before and After Transcription (%) for Different Whisper Models and Audio Lengths

Model	2 sec	10 sec	20 sec	30 sec	34 sec	45 sec	1 min	2 min
Tiny	48.3 → 66.3	0.0 → 91.2	75.0 → 80.9	0.0 → 80.0	75.0 → 77.8	0.0 → 77.8	75.0 → 85.3	0.0 → 77.6
Small	47.8 → 79.2	0.0 → 83.1	0.0 → 81.4	62.5 → 77.6	100.0 → 87.0	0.0 → 83.9	0.0 → 86.2	100.0 → 71.4
Base	46.1 → 70.2	50.0 → 56.3	0.0 → 54.0	62.5 → 55.0	70.0 → 56.3	0.0 → 66.8	75.0 → 66.0	62.5 → 71.4
Medium	27.1 → 64.7	62.5 → 60.9	0.0 → 55.5	50.0 → 55.2	0.0 → 57.9	0.0 → 55.4	50.0 → 53.9	83.3 → 62.3

Table 3: CPU Usage Before and After Transcription (%) for Different Models and Audio Lengths

Observations:CPU usage varies across models and audio lengths without a clear correlation between model size, audio length, and CPU consumption.

- Memory Usage Before and After Transcription (MB) for Different Whisper Models and Audio Lengths

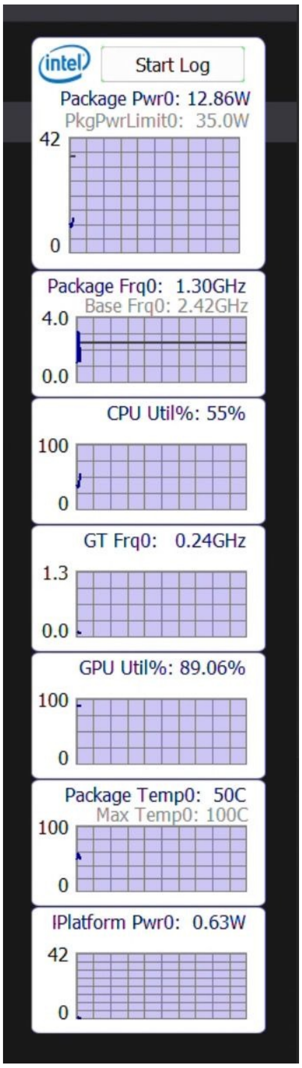
Model	2 sec	10 sec	20 sec	30 sec	34 sec	45 sec	1 min	2 min
Tiny	6872.46 → 6811.22	6810.27 → 6741.37	6741.41 → 6727.04	6727.06 → 6745.74	6745.15 → 6760.02	6760.11 → 6772.47	6772.64 → 6790.91	6790.95 → 6662.84
Small	7573.62 → 7111.23	7111.21 → 6474.02	6474.50 → 6581.62	6581.41 → 6771.68	6771.71 → 6853.54	6850.86 → 6827.03	6827.42 → 6933.12	6934.12 → 7058.46
Base	7106.88 → 6808.98	6809.29 → 6095.21	6095.34 → 6100.43	6100.43 → 6103.79	6103.47 → 6116.34	6116.34 → 6117.67	6118.24 → 6149.41	6149.67 → 6241.97
Medium	7647.98 → 6613.30	6613.64 → 6791.00	6791.00 → 6906.51	6906.52 → 6967.04	6969.30 → 6934.13	6934.09 → 6833.08	6833.09 → 6888.56	6888.31 → 7172.55

Table 4: Memory Usage Before and After Transcription (MB) for Different Models and Audio Lengths

Observations: Memory usage varies during transcription. Larger models (Small, Medium) generally require more memory than smaller models (Tiny, Base), but the relationship is not strictly consistent across all audio lengths.

Measuring Energy Usage of Local Hardware

Before running the Python script, power usage is at 12.86W, indicating a low workload. The CPU frequency is 1.30GHz, which is below its base frequency, suggesting minimal processing demand. The temperature is 50°C, showing the system is in a cool and stable state. GPU utilization is at 89.06%, possibly due to background tasks or system processes.



Package Pwr0: 12.86W
PkgPwrLimit0: 35.0W

Package Frq0: 1.30GHz
Base Frq0: 2.42GHz

CPU Util%: 55%

GT Frq0: 0.24GHz

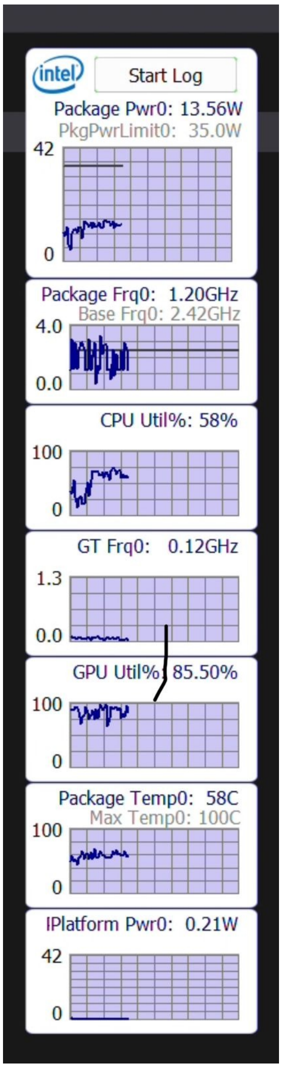
GPU Util%: 89.06%

Package Temp0: 50C
Max Temp0: 100C

IPlatform Pwr0: 0.63W

Figure 1: Before Running Python Code

After completing the tiny model and starting the small model, power usage rises slightly to 13.56W, reflecting an increase in computational load. The CPU frequency drops to 1.20GHz, which may be due to power management adjustments. The temperature increases to 58°C, indicating a moderate rise in heat output. GPU utilization decreases slightly to 85.50%, suggesting a shift in workload distribution.



Package Pwr0: 13.56W
PkgPwrLimit0: 35.0W

Package Frq0: 1.20GHz
Base Frq0: 2.42GHz

CPU Util%: 58%

GT Frq0: 0.12GHz

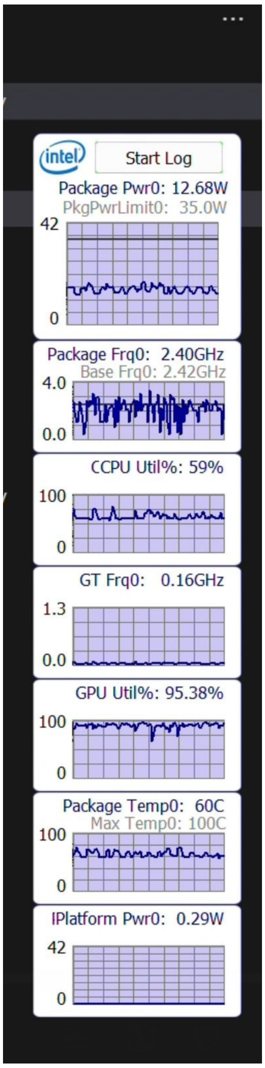
GPU Util%: 85.50%

Package Temp0: 58C
Max Temp0: 100C

IPlatform Pwr0: 0.21W

Figure 2: After Finishing Tiny Model, Starting "Small" Model

As the system transitions from the small model to the base model, power usage slightly decreases to 12.68W, while CPU frequency jumps to 2.40GHz, suggesting a shift to higher processing power. The temperature reaches 60°C, reflecting an increase in system workload. GPU utilization rises to 95.38%, indicating heavier reliance on GPU resources.



Package Pwr0: 12.68W

PkgPwrLimit0: 35.0W

Package Frq0: 2.40GHz

Base Frq0: 2.42GHz

CCPU Util%: 59%

GT Frq0: 0.16GHz

GPU Util%: 95.38%

Package Temp0: 60C

Max Temp0: 100C

IPlatform Pwr0: 0.29W

Figure 3: After Finishing Small Model, Starting "Base" Model

With the base model completed and the medium model starting, power usage peaks at 15.07W, the highest recorded so far. The CPU frequency drops significantly to 0.80GHz, likely due to thermal throttling. The temperature spikes to 69°C, indicating maximum heat generation under heavy load. GPU utilization remains high at 88.26%, showing the GPU is still heavily engaged.

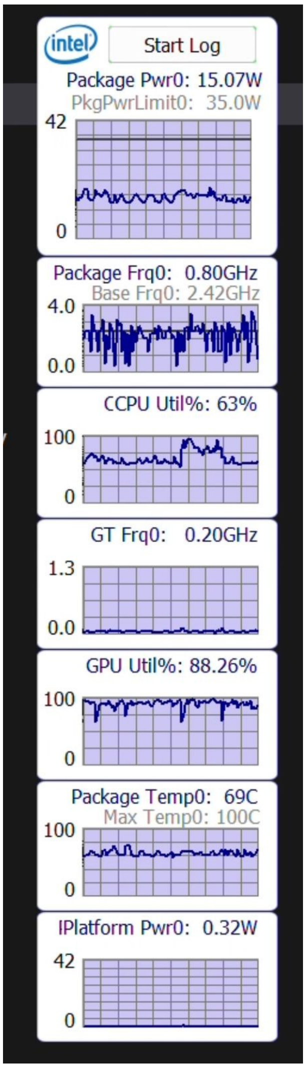
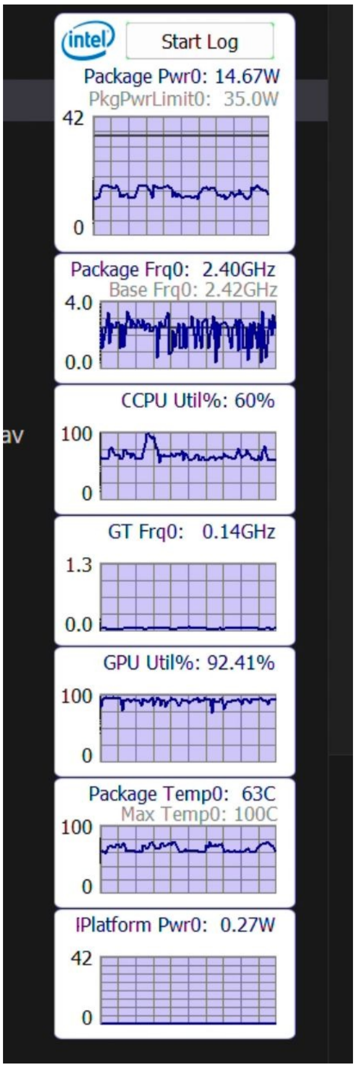


Figure 4: After Finishing Base Model, Starting "Medium" Model

After completing the medium model, power usage remains high at 14.67W, while CPU frequency returns to 2.40GHz, indicating that processing load has decreased. The temperature drops slightly to 63°C, showing the system is beginning to stabilize after heavy processing. GPU utilization stays high at 92.41%, indicating continued GPU engagement, possibly due to background processes.



Package Pwr0: 14.67W
PkgPwrLimit0: 35.0W

Package Frq0: 2.40GHz
Base Frq0: 2.42GHz

CCPU Util%: 60%

GT Frq0: 0.14GHz

GPU Util%: 92.41%

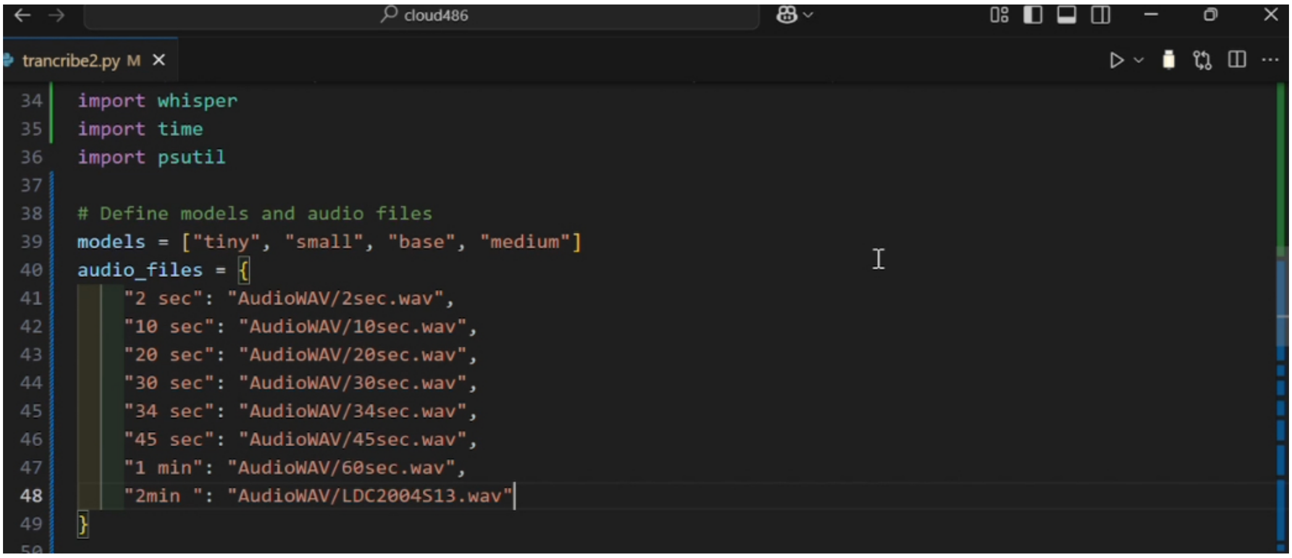
Package Temp0: 63C
Max Temp0: 100C

IPlatform Pwr0: 0.27W

Figure 5: After Finishing Medium Model, Python Code Stopped

Implementation Details

We used python to implement Whisper, utilizing libraries like `(whisper)` for loading and running the model, `(time)` for measuring inference time and response time, and `(psutil)` for monitoring CPU and memory usage. GPU activity was also observed and analyzed, as detailed in the `Measuring Energy Usage of Local Hardware` section. This analysis provides insights into GPU utilization trends across different stages of model execution. For clarification, a snippet of the code showing the imported libraries is provided in Figure 6 below.



```python
import whisper
import time
import psutil

# Define models and audio files
models = ["tiny", "small", "base", "medium"]
audio_files = {
    "2 sec": "AudioWAV/2sec.wav",
    "10 sec": "AudioWAV/10sec.wav",
    "20 sec": "AudioWAV/20sec.wav",
    "30 sec": "AudioWAV/30sec.wav",
    "34 sec": "AudioWAV/34sec.wav",
    "45 sec": "AudioWAV/45sec.wav",
    "1 min": "AudioWAV/60sec.wav",
    "2min ": "AudioWAV/LDC2004S13.wav"
}
```

Figure 6: Libraries imported for Whisper implementation.

The complete code used in this implementation is available in an external file submitted with this paper. Alternatively, you can access it via the following GitHub repository: https://github.com/afnanAlonazi/cloud486.git

3. Setting Up Cloud Deployment (Azure)

Microsoft Azure provides robust cloud-based speech-to-text capabilities through its Azure Cognitive Services. These services support a variety of applications such as real-time transcription, voice-command interfaces, and accessibility tools. Azure's Speech-to-Text API leverages deep learning to deliver high-accuracy transcription across numerous languages and dialects. [5]

3.1 Why We Chose Azure

We selected Microsoft Azure for its powerful and flexible Speech-to-Text service, which includes essential features such as real-time transcription, fast synchronous output, and batch processing for prerecorded audio. Additionally, Azure offers custom speech models, enabling enhanced accuracy for domain-specific applications. These capabilities align well with our deployment needs, ensuring both scalability and performance in real-world scenarios [6]

3.2 Azure Speech Resource Setup

The deployment process involved:

3.2.1 Azure Speech Service Configuration

The Azure Speech service was configured through the following procedural steps:

1. Account Registration and Portal Access:
   - Accessed the Azure Portal [7]
   - Created an account using personal credentials
2. Resource provisioning:
   - Initiated resource creation using the search term ”Speech”
   - Selected appropriate subscription and resource group
   - Chose a preferred deployment region
3. Tier selection:
   - Selected the **Standard Tier (F0)** for production
4. Credential management:
   - Navigated to the Keys and Endpoint section of the Speech resource
   - Retrieved the Subscription Key and Region details
   - Stored the credentials securely for integration within the development environment

3.2.2 Development Environment Preparation

To enable programmatic interaction with the Azure Speech API, the following Python dependencies were installed:

```
pip install azure-cognitiveservices-speech
pip install librosa
pip install requests
```

Package functionalities were as follows:

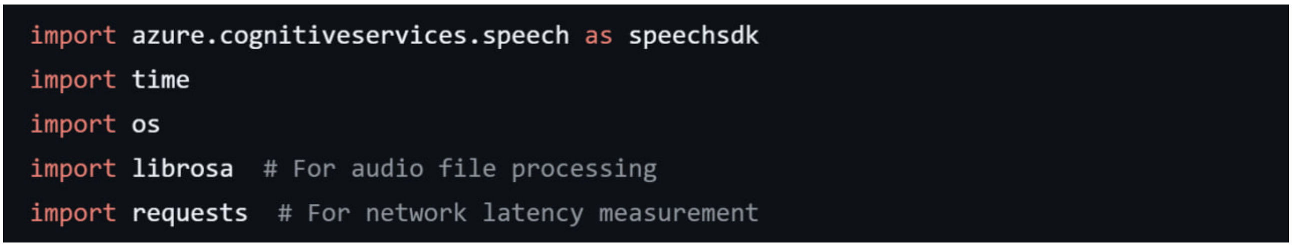
- `azure-cognitiveservices-speech`: Allows communication with Azure Speech-to-Text services.
- `librosa`: Loads and processes audio files. Used to determine audio duration before sending to the Azure service.
- `requests`: Sends HTTP requests to measure network latency to the Azure Speech endpoint.

3.3 Implementation Details (Code Summary)

The implementation includes several components that support automated speech recognition using Azure's cloud services. This section summarizes key functions for audio preprocessing, recognizer setup, latency measurement, transcription handling, and result logging.

0.1 Imported Libraries in the Code

- `import azure.cognitiveservices.speech as speechsdk`: Connects to Azure's Speech-to-Text service for transcribing audio into text.
- `import time`: Measures timing for response and inference durations.
- `import os`: Handles file and directory management.
- `import librosa`: Processes audio files and calculates their duration.
- `import requests`: Measures network latency by sending test HTTP requests to Azure's endpoint.

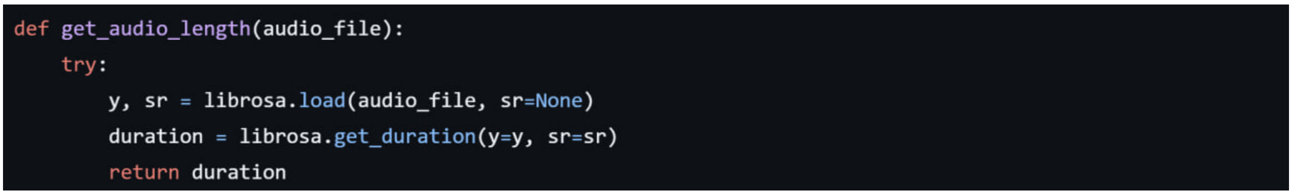


```
import azure.cognitiveservices.speech as speechsdk
import time
import os
import librosa  # For audio file processing
import requests  # For network latency measurement
```

Figure 7: Import statements for the system.

0.2 Audio Duration Estimation

The system first calculates the length of each audio file in seconds. This is performed using the `librosa` library, which accurately loads the waveform and sampling rate before estimating the duration.

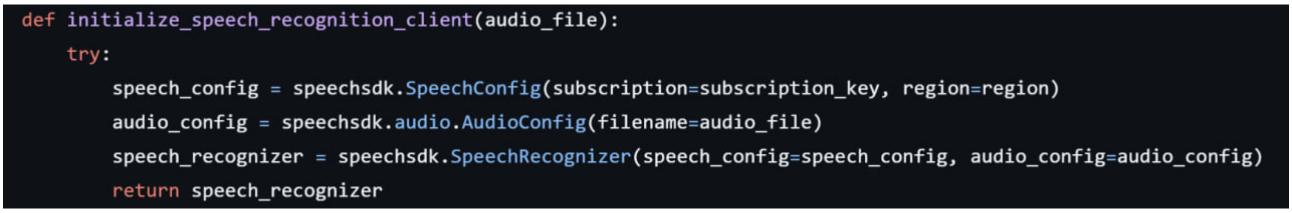


```
def get_audio_length(audio_file):
    try:
        y, sr = librosa.load(audio_file, sr=None)
        duration = librosa.get_duration(y=y, sr=sr)
        return duration
```

Figure 8: Audio duration calculation using `librosa`.

0.3 Speech Recognizer Initialization

Azure's Speech SDK is used to configure the recognizer. The system sets the subscription credentials and region and binds the local audio file through an audio configuration object.

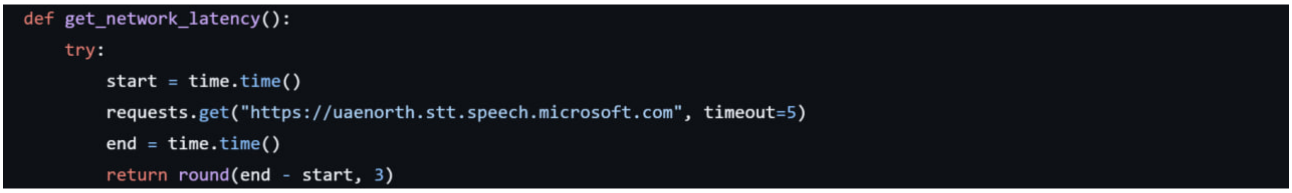


```
def initialize_speech_recognition_client(audio_file):
    try:
        speech_config = speechsdk.SpeechConfig(subscription=subscription_key, region=region)
        audio_config = speechsdk.audio.AudioConfig(filename=audio_file)
        speech_recognizer = speechsdk.SpeechRecognizer(speech_config=speech_config, audio_config=audio_config)
        return speech_recognizer
```

Figure 9: Initialization of the Azure Speech recognizer.

0.4 Network Latency Measurement

To isolate the computational inference time from total system latency, an HTTP GET request is used to estimate the network delay when reaching Azure's STT endpoint.

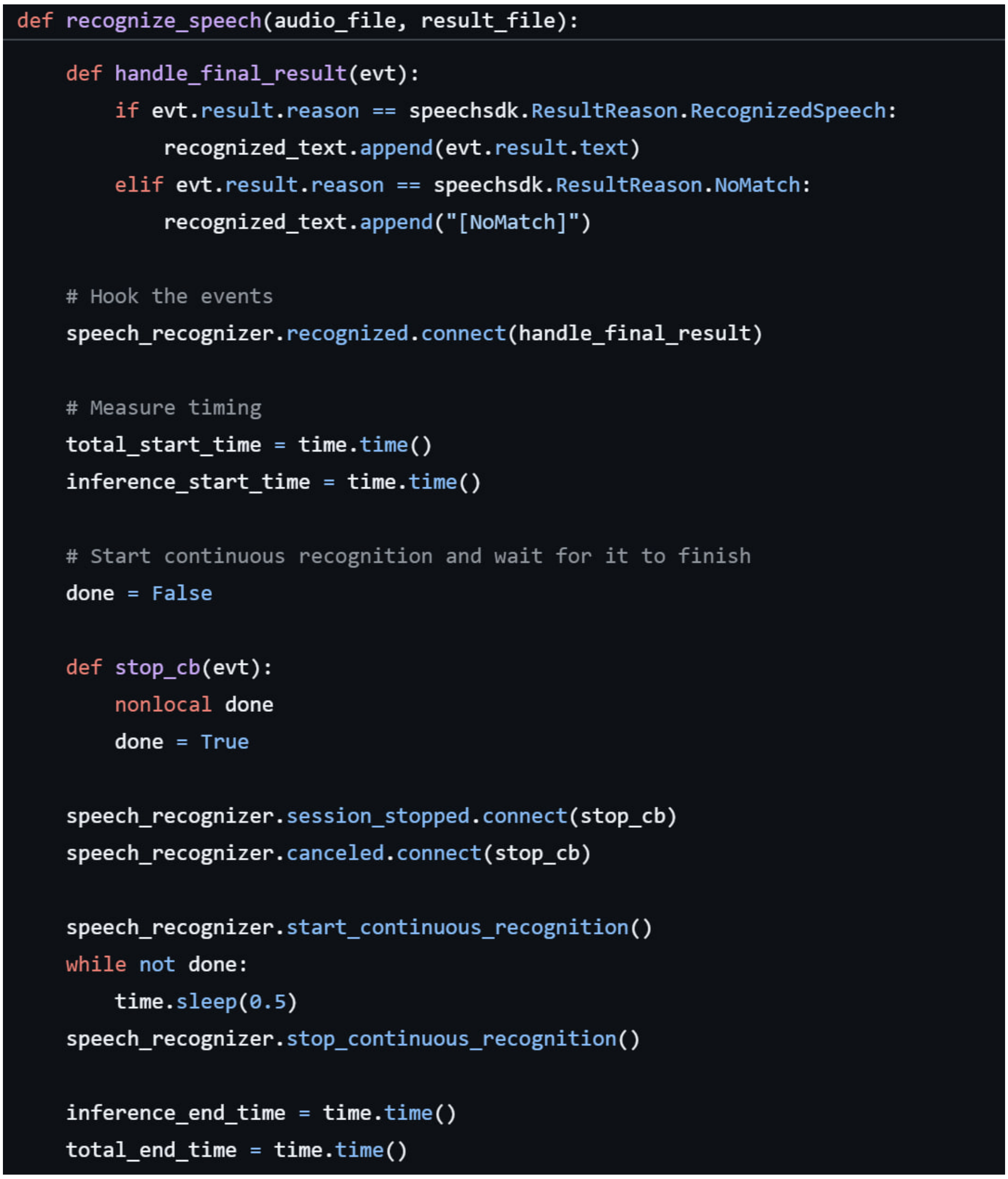


```
def get_network_latency():
    try:
        start = time.time()
        requests.get("https://uaenorth.stt.speech.microsoft.com", timeout=5)
        end = time.time()
        return round(end - start, 3)
```

Figure 10: Measuring network latency to the Azure STT endpoint.

0.5 Continuous Speech Recognition

The system performs continuous recognition using an event-driven approach. Callback functions handle transcription results and termination signals. During the process, timestamps are recorded to compute total response time.

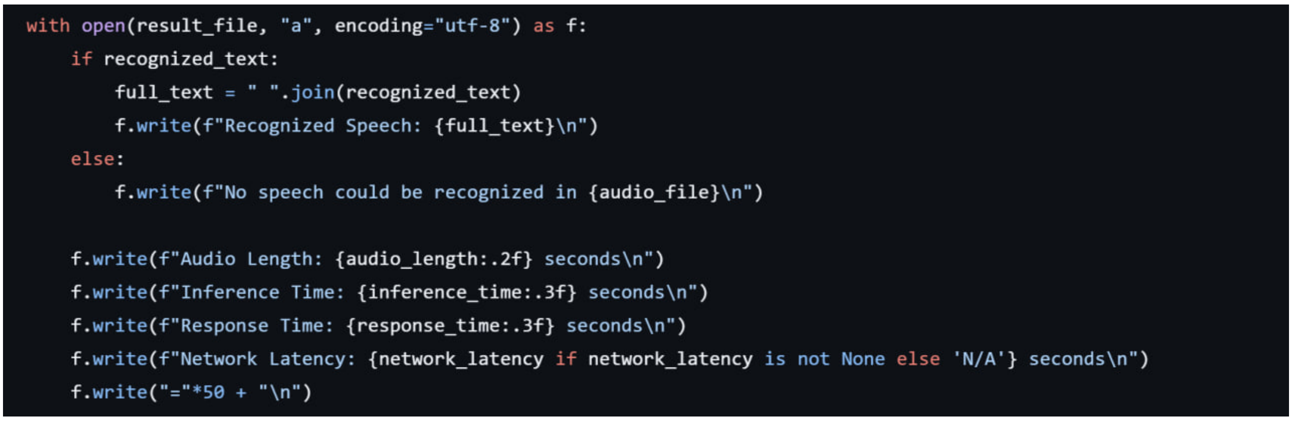


```python
def recognize_speech(audio_file, result_file):

    def handle_final_result(evt):
        if evt.result.reason == speechsdk.ResultReason.RecognizedSpeech:
            recognized_text.append(evt.result.text)
        elif evt.result.reason == speechsdk.ResultReason.NoMatch:
            recognized_text.append("[NoMatch]")

    # Hook the events
    speech_recognizer.recognized.connect(handle_final_result)

    # Measure timing
    total_start_time = time.time()
    inference_start_time = time.time()

    # Start continuous recognition and wait for it to finish
    done = False

    def stop_cb(evt):
        nonlocal done
        done = True

    speech_recognizer.session_stopped.connect(stop_cb)
    speech_recognizer.canceled.connect(stop_cb)

    speech_recognizer.start_continuous_recognition()
    while not done:
        time.sleep(0.5)
    speech_recognizer.stop_continuous_recognition()

    inference_end_time = time.time()
    total_end_time = time.time()
```

Figure 11: Continuous recognition and event-based transcription.

0.6 Performance Metrics Logging

Recognized text and all calculated metrics (including audio length, inference time, and latency) are logged into a result file for further analysis. Inference time is estimated by subtracting the network latency from the overall response time.

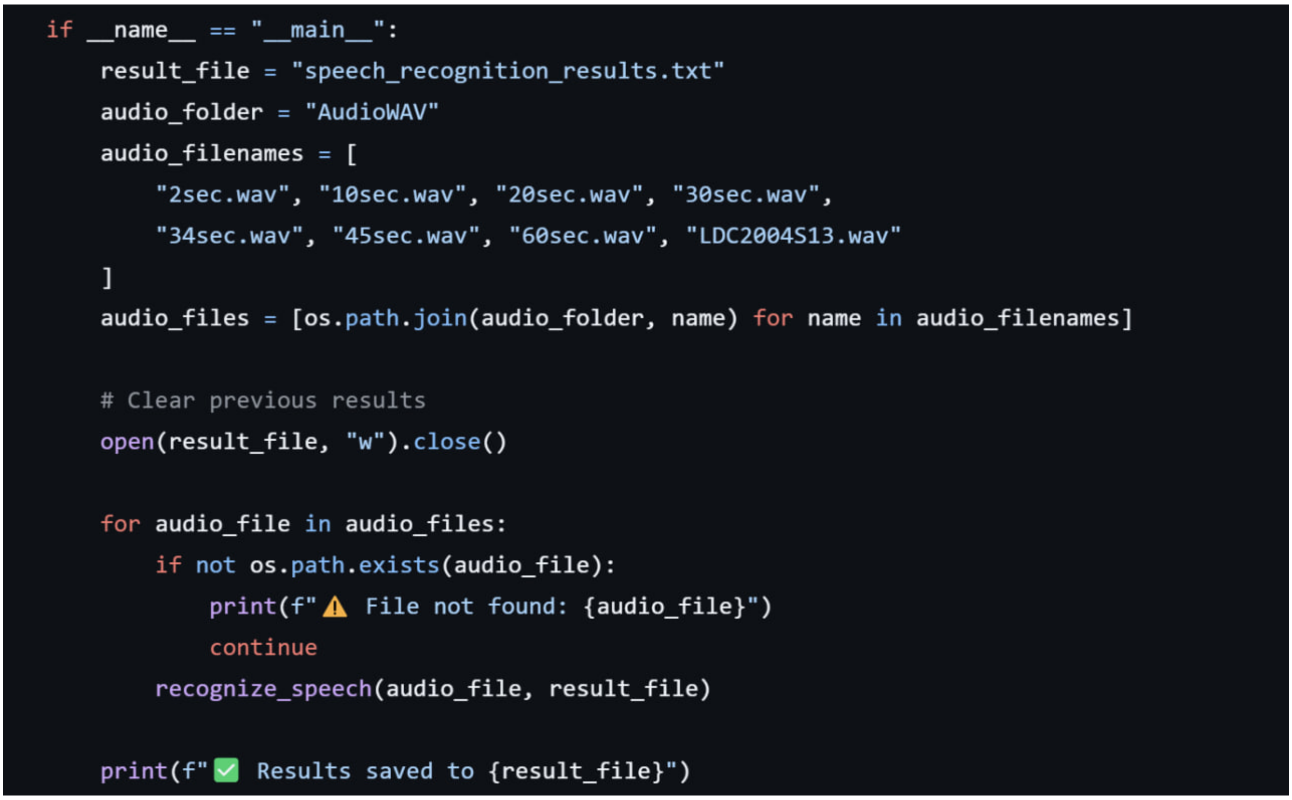


```python
with open(result_file, "a", encoding="utf-8") as f:
    if recognized_text:
        full_text = " ".join(recognized_text)
        f.write(f"Recognized Speech: {full_text}\n")
    else:
        f.write(f"No speech could be recognized in {audio_file}\n")

    f.write(f"Audio Length: {audio_length:.2f} seconds\n")
    f.write(f"Inference Time: {inference_time:.3f} seconds\n")
    f.write(f"Response Time: {response_time:.3f} seconds\n")
    f.write(f"Network Latency: {network_latency if network_latency is not None else 'N/A'} seconds\n")
    f.write("="*50 + "\n")
```

Figure 12: Recording recognized text and timing metrics.

0.7 Main Execution Flow

The script loops through a predefined list of audio samples, applies the speech recognition pipeline to each, and appends results sequentially. This setup enables comparative analysis across different audio durations.



```python
if __name__ == "__main__":
    result_file = "speech_recognition_results.txt"
    audio_folder = "AudioWAV"
    audio_filenames = [
        "2sec.wav", "10sec.wav", "20sec.wav", "30sec.wav",
        "34sec.wav", "45sec.wav", "60sec.wav", "LDC2004S13.wav"
    ]
    audio_files = [os.path.join(audio_folder, name) for name in audio_filenames]

    # Clear previous results
    open(result_file, "w").close()

    for audio_file in audio_files:
        if not os.path.exists(audio_file):
            print(f"⚠️ File not found: {audio_file}")
            continue
        recognize_speech(audio_file, result_file)

    print(f"✅ Results saved to {result_file}")
```

Figure 13: Execution loop over multiple audio files.

3.4 Azure Speech Recognition Performance Comparison

Unlike Whisper, Azure Speech-to-Text does not offer different model sizes. Instead, it provides a single model. focuses on how well the Azure service performs when dealing with audio files of different lengths. We'll be looking at important metrics like inference time, total response time, and network latency.

Audio Length	Inference Time (s)	Response Time (s)	Network Latency (s)
2 sec	1.187	1.516	0.329
10 sec	2.549	3.023	0.474
20 sec	7.674	8.066	0.392
30 sec	12.131	12.599	0.468
34 sec	14.153	14.620	0.467
45 sec	19.708	20.164	0.456
60 sec	27.069	27.715	0.646
2 min	56.869	57.441	0.572

Table 5: Response, Inference, and Network Latency (seconds) for Azure STT with Different Audio Lengths

Observations and Analysis:

- **Inference Time:** The inference time in Azure Speech-to-Text increases as the audio length gets longer. For short audio clips around 2 to 10 seconds, the model responds quickly, usually taking less than 3 seconds to respond. But when we start dealing with longer audio, like a 2-minute file, the inference time rises significantly over 56 seconds. This shows a linear relationship between the length of the audio and the processing time, which makes sense since longer audio means more computation is needed.
- **Response Time:** The response time also increases as the audio length gets longer. This is because it measures the total time from when you send the request to when you get the transcription back. For short audio, response times stay under 3 seconds, while for the 2-minute audio it exceeds 57 seconds. This shows a linear relationship between the length of the audio and the Response time.
- **Network Latency:** Network latency consistent across all audio lengths, ranging from 0.3 to 0.6 seconds. This shows that Azure's network setup is stable.

3.5. Performance Comparison of Local and Cloud-Based

While both the local (Whisper) and cloud (Azure STT) deployments aim to provide accurate speech recognition, the benchmarking process varies in the performance metrics collected.

- **Model Control:** Whisper offers multiple model sizes (Tiny, Small, Base, Medium), so you can find the perfect balance between speed and accuracy. On the other hand, Azure STT offers a single model.
- **Inference and Response Time:** Both systems measure inference and response times. Whisper measures these times locally. Azure measure network communication overhead, making its response time more representative of real-world deployment.

- **Network Latency:** Azure STT, being cloud-based, includes an additional latency to consider since it has to send and receive data over the network. On the other hand, Whisper operates completely locally, which means it doesn't have to deal with any network delays.
- **Resource Utilization:** Whisper benchmarks include CPU usage and memory consumption both before and after transcription, indicating how it is affects the host device's performance. the other hand, Azure operates on cloud infrastructure and doesn't reveal the internal resource usage of its remote servers.
- **Scalability:** Azure STT does its computation on cloud, which is suitable for devices with limited hardware. On the other hand, Whisper offers a lot of flexibility, but it might not be the best fit for lighter devices, especially when you're using bigger models like Medium[5].

  Note on Energy Usage: While we were able to measure energy consumption and hardware resource usage for the Whisper models running locally, we couldn't capture those metrics for Azure STT. Because Azure processes data in the cloud. This makes it challenging to conduct a complete end-to-end comparison regarding environmental impact or resource efficiency.

To ensure a meaningful and balanced comparison with Azure Speech-to-Text, we initially considered multiple Whisper models. Although Whisper-Tiny is the quickest and most lightweight option, we noticed it had some inconsistencies when dealing with different audio lengths. On the other hand, Whisper-Small and Whisper-Base demonstrated a more consistent and proportional increase in both inference and response times as audio length increased — a pattern that closely reflects Azure STT's behavior.

For this reason, we selected Whisper-Small as the benchmark model for local deployment. It aligns better with Azure's performance, particularly for medium to long audio.

Audio Length	**Whisper-Small Inference (s)**	**Azure Inference (s)**	**Whisper-Small Response (s)**	**Azure Response (s)**
2 sec	4.90	1.187	4.98	1.516
10 sec	6.25	2.549	6.33	3.023
20 sec	7.51	7.674	7.59	8.066
30 sec	10.86	12.131	10.96	12.599
34 sec	13.15	14.153	13.24	14.620
45 sec	14.20	19.708	14.28	20.164
60 sec	17.34	27.069	17.45	27.715
2 min	36.33	56.869	36.45	57.441

Table 6: Direct Comparison of Inference and Response Times for Whisper-Small and Azure STT

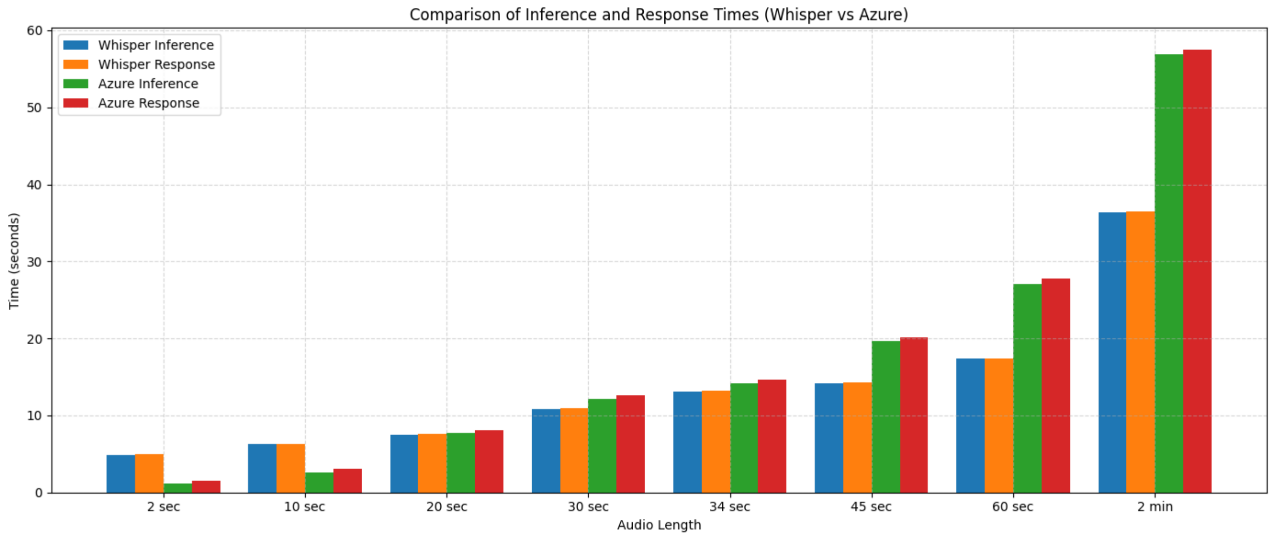


Figure 14: Bar chart showing a performance comparison between Whisper-Small and Azure STT for different audio lengths, generated using Python's matplotlib.

Observations: Based on the results presented in Table 6, Whisper-Small shows a consistent increase in both inference and response times relative to Azure STT. Although Azure performs better with short audio clips (e.g., 2–10 seconds), Whisper-Small becomes more competitive—and even more efficient—as audio length increases.

For example, at the 2-minute mark, Whisper-Small managed to transcribe the audio in just 36.45 seconds (response time), while Azure took a bit longer at 57.44 seconds. This consistency makes Whisper-Small a reliable benchmark for local deployment when comparing with cloud-based alternatives like Azure STT. .

3.6 Cost Analysis

From a cost perspective, both Whisper and Azure STT offer different trade-offs that can influence your deployment choices.

- **Whisper:** Whisper is an open-source model which means it is free to use. There are no subscription fees or API usage limits. However, it requires some computational demands, especially with the larger models like medium . it might need a powerful device or server that has enough CPU, RAM for efficient processing[8].

- **Azure STT:** Azure Speech-to-Text provides a partially free tier (Standard S0) with some monthly usage limits. when exceeding the free quota, it charges based on how long your audio is, the region you're in, and other service specifics. it eliminates the need for powerful local hardware[9].

3.7 Use Case Analysis

Choosing between Whisper and Azure STT depends what your application needs. considering hardware availability, internet connectivity, audio conditions, and scalability.

- **Whisper is preferred when:**
  - The application needs to work offline without relying on internet connection.

  - The environment has powerful hardware.
  - Privacy is a concern, and it's important to keep all data stored on your device.
- **Azure STT is preferred when:**
  - The application will be used on low-powerful hardware .
  - There is stable , strong internet access.
  - There is a need for scalability and seamless integration with other Azure services.

3.8 Final Recommendations

Based on the experimental results and analysis conducted in this project, we have come up with some recommendations for choosing and implementing speech recognition solutions:

- **Best Performing Model:** For local deployments, the **Whisper-Tiny** model offers an excellent balance between speed and accuracy. It better than Azure STT when it comes to inference and response times, particularly with longer audio files.
- **Performance Optimization:**
  - For Whisper, use smaller models if you need real-time performance, but go for the larger models when you have the time and resources to focus on accuracy
  - For Azure, consider reducing audio file sizes or optimizing preprocessing to minimize latency and cost.
- **Hybrid Approach:** In certain situations, it might be best to combining both solutions. For example, Whisper can be used for offline and Azure for online . This hybrid approach guarantees that you have reliability, flexibility, and cost-effectiveness[12].

Conclusion

This project compared the performance of local (Whisper) and cloud-based (Azure STT) speech recognition services. By measuring benchmarking metrics such as inference time, response latency, and resource usage, we found that Whisper delivers quicker performance when used locally. On the other hand, Azure provides scalability and ease of integration. In the end, the ideal choice depends on the application's context.

Resources

[1] Gomez, K. (2023, July 30). Cloud vs. On-Premise: Where to Deploy Your AI Applications. Medium. https://medium.com/40kyeg/cloud-vs-on-premise-where-to-deploy-your-ai-applications-b584335ae86a

[2] Kirvan, P., Lutkevich, B., Kiwak, K. (2024, November 20). What is speech recognition? Search Customer Experience. https://www.techtarget.com/searchcustomerexperience/definition/speech-recognition

[3] Gladia - What is OpenAI Whisper? (n.d.-b). https://www.gladia.io/blog/what-is-openai-whisper

[4] Golla, R. G. (2023, March 6). Here are six practical use cases for the new Whisper API. Slator. https://slator.com/six-practical-use-cases-for-new-whisper-api/

[5] AI Services — Microsoft Azure. (n.d.). https://azure.microsoft.com/en-us/products/ai-services/

[6] Eric-Urban. (n.d.-b). Speech to text overview - Speech service - Azure AI services. Microsoft Learn. https://learn.microsoft.com/en-us/azure/ai-services/speech-service/speech-to-text

[7] Microsoft Azure. (n.d.). https://portal.azure.com/

[8] Whisper: Open-source Automatic Speech Recognition. Retrieved from https://openai.com/research/

[9]Microsoft Azure. (n.d.). *Speech-to-Text Pricing.* Retrieved from https://azure.microsoft.com/en-us/pricing/details/cognitive-services/speech-services/

[10]Golla, R. G. (2023, March 6). Here are six practical use cases for the new Whisper API. Slator. https://slator.com/six-practical-use-cases-for-new-whisper-api

[11] Microsoft Azure. (n.d.). Azure AI Speech - Use Cases Retrieved from https://azure.microsoft.com/e

us/products/ai-services/speech-services/#features

[12] Microsoft Learn. (2024). *Whisper model in Speech service - Azure AI services.* Retrieved from https://learn.microsoft.com/en-us/azure/ai-services/speech-service/whisper-overview