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 questions regarding the performance and efficiency of local versus cloud-based speech recognition.

1. How does real-time speech recognition performance compare between local and cloud deployments?

2. What is the impact of audio quality on accuracy in local vs. cloud-based speech recognition?

3. How does cloud latency affect real-time applications like voice assistants?

4. Is running speech recognition locally practical in terms of processing power efficiency, or would it drain too many resources compared to using a cloud-based service?

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.

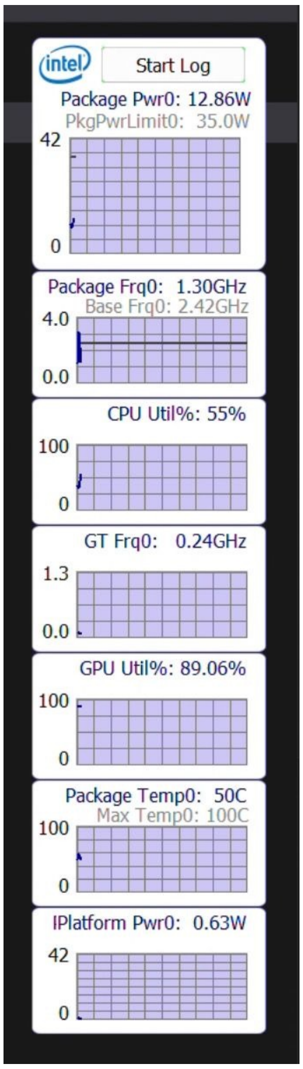


Figure 1: Before Running Python Code

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.

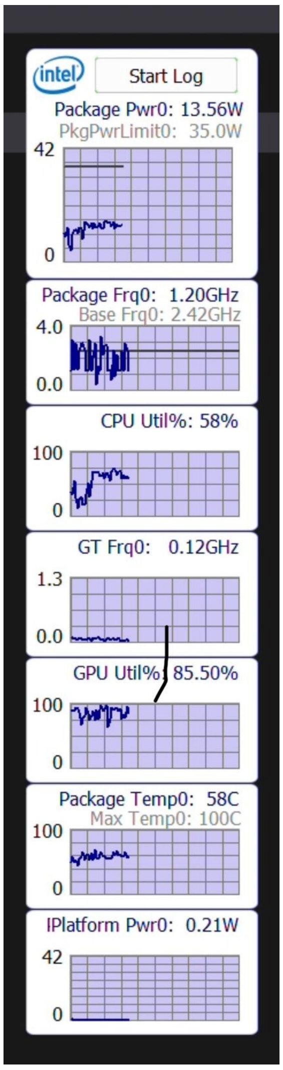
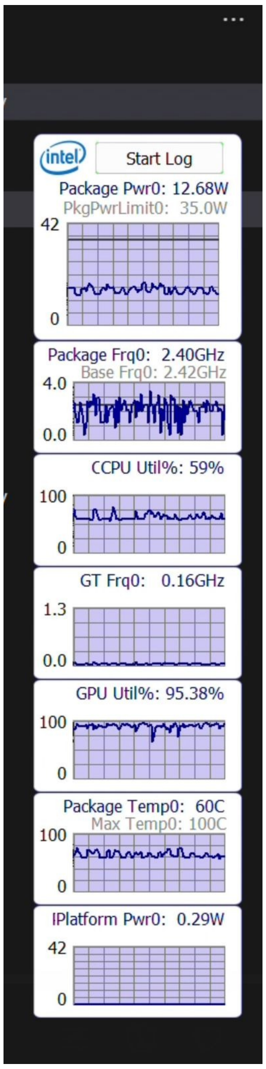


Figure 2: After Finishing Tiny Model, Starting "Small" Model

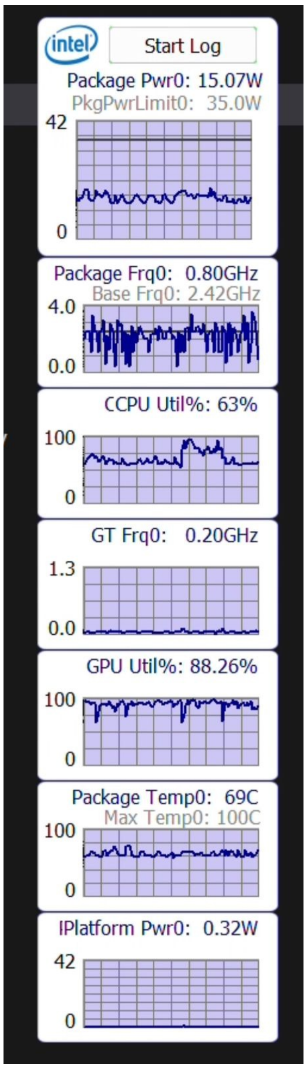
As the system transitions from the small model to the base model, power usage slightly decreases to 12.68W, while CPU frequency jumps to 2.40GHz, suggesting a shift to higher processing power. The temperature reaches 60°C, reflecting an increase in system workload. GPU utilization rises to 95.38%, indicating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



```
Package Pwr0: 15.07W
PkgPwrLimit0: 35.0W
Package Frq0: 0.80GHz
Base Frq0: 2.42GHz
CCPU Util%: 63%
GT Frq0: 0.20GHz
GPU Util%: 88.26%
Package Temp0: 69C
Max Temp0: 100C
IPlatform Pwr0: 0.32W
```

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.

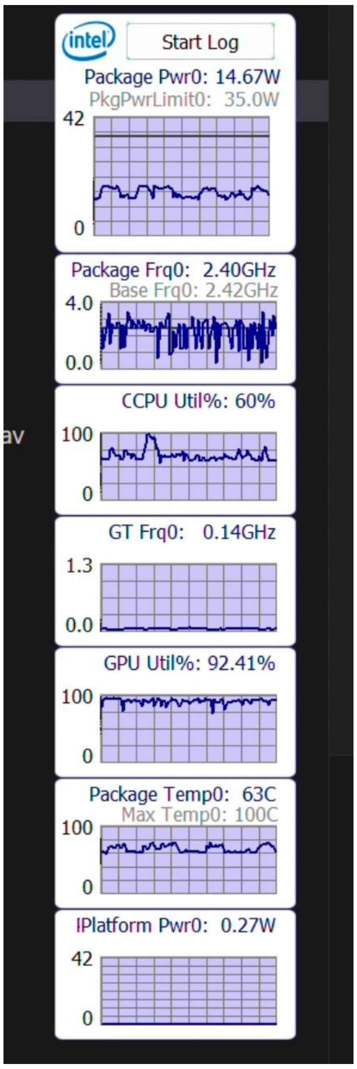
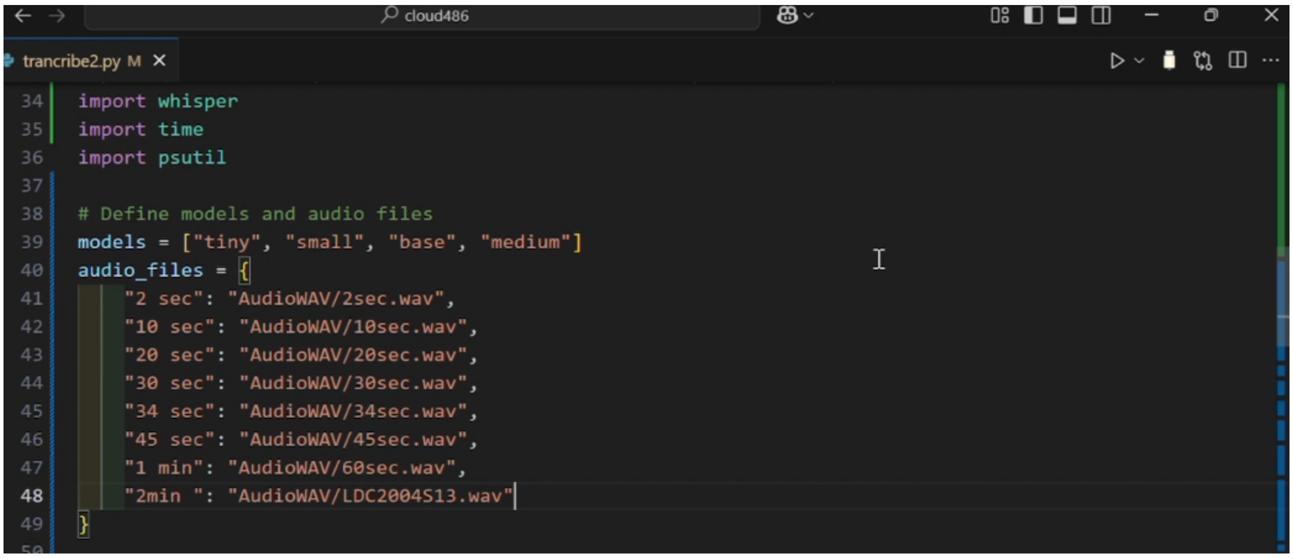


Figure 5: After Finishing Medium Model, Python Code Stopped

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.



```
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

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/