Resource Profiling for Smart Home Machine Learning Applications

Paul Florian
University of Twente, Enschede, Netherlands

ABSTRACT

As smart home devices increasingly incorporate machine learning applications, their resource constraints pose significant challenges for deployment. This research investigates resource profiling to better understand the requirements of machine learning applications in smart home devices under varying conditions. By analyzing the effects of input size, model complexity, workload conditions, and hardware platforms, we aim to understand how these factors influence resource utilization and system performance. Controlled experiments will focus on face recognition and speech recognition. These applications were chosen for their relevance as representative machine learning tasks in smart homes. They cover key areas like security and accessibility, making them ideal for studying resource demands. To capture performance across a range of hardware capabilities, we will conduct testing on two hardware platforms, namely the Raspberry Pi and a Desktop PC. The results will identify critical resource requirements and offer insights for more efficient implementation of machine learning in smart home devices, enhancing their capabilities and reliability.

ACM Reference Format:
Paul Florian, University of Twente, Enschede, Netherlands. 2025. Resource Profiling for Smart Home Machine Learning Applications. In *Proceedings of the 42nd Twente Student Conference on IT*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

As technology advances, smart home devices continue to evolve, to the point where many are now incorporating machine learning applications that significantly enhance user experiences. These applications are diverse and impactful, addressing key areas such as security and accessibility. Face recognition, for example, provides a cheap and effective home security solution by identifying people entering or leaving the home with high accuracy. This can reduce the need for costly traditional security systems, making advanced protection more accessible [5]. Similarly, researchers have achieved highly accurate speech recognition through deep learning techniques, enabling effortless voice control of lights, speakers, and other devices. This is especially beneficial for users with accessibility needs, who can now interact with their smart devices in a more natural and intuitive way [4]. By fundamentally changing how people interact with their homes, these machine learning applications are shaping the future of smart devices.

However, smart home devices are often constrained by limited computational resources due to their small size and energy efficiency requirements. On the other hand, machine learning applications often have high computational demands, making it challenging to run them on smart home devices. To be able to run these applications, it is necessary to either scale up the devices to provide greater processing power, or to offload the tasks to a more computationally powerful device, e.g., local smart hubs, edge servers, or cloud servers [10].

Scaling up devices allows processing with minimal latency but is often impractical due to higher energy consumption, costs, and size limitations. Offloading tasks to cloud servers offers scalability but can introduce latency [8] and privacy concerns. Edge computing strikes a balance by offloading tasks to nearby hubs or servers, reducing latency and bandwidth use while offering more power than individual devices [9]. However, it also faces resource constraints and potential bottlenecks in high-demand scenarios.

Strategically allocating resources is essential in all scenarios to ensure that the specific needs of the machine learning applications being implemented are met. Properly allocating resources where needed has been shown to be very effective in optimizing resource utilization and system performance [7].

In this research we aim to determine what resources are most important to run these machine learning applications and to see how these requirements change under different conditions. These conditions include factors such as system load, hardware type, and varying input sizes. Therefore, in this research we aim at addressing the following research questions:

- **RQ1:** How does input size affect resource utilization of machine learning applications?
- **RQ2:** How does model complexity affect resource utilization of machine learning applications?
- **RQ3:** How do workload conditions impact the performance and resource utilization of machine learning applications?
- **RQ4:** How does application performance vary across hardware platforms?

2 RELATED WORK

Edge computing involves processing data closer to where it is generated, such as on smart devices or sensors, rather than sending it all to a central cloud server. It has become essential for deploying machine learning applications in smart homes, where resources are limited. This approach helps reduce latency and saves on bandwidth, allowing for quicker responses. Studies have highlighted the role of hardware accelerators, input characteristics, and system conditions in influencing performance.

TScIT42, 31 January 2025, Enschede, Netherlands

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Zhou et al. [2] demonstrated that resource demands in machine learning tasks vary by hardware type and workload, emphasizing the need for profiling under different conditions. Liang et al. [3] showed that while edge accelerators like the EdgeTPU improve latency and efficiency, their performance declines with larger models or under high-concurrency scenarios. The PAIGE framework by Liang, O'Keeffe, and Sastry [1] highlighted how input size and workload stress impact hybrid edge-cloud architectures, stressing the importance of understanding these dynamics for resource optimization.

While previous studies have looked at hardware accelerators, input characteristics, and workload stress in edge and hybrid systems, our work takes a broader approach by examining how input size, model complexity, and hardware type together influence resource utilization. Unlike Zhou et al. [2] and Liang et al. [3], which focus on specific scenarios or architectures, we aim to identify the most critical resources across diverse conditions. This will provide a deeper understanding of how to optimize performance for machine learning applications in smart home environments, especially under varying workloads.

3 METHODOLOGY

The research relies on controlled experiments to systematically test machine learning applications under varying conditions. The experiments will measure how input size, model complexity, and system load impact performance and resource utilization on different hardware platforms.

3.1 Applications

We focus on two machine learning applications, representative of smart home environments. Both applications are implemented in JavaScript using Node.js.

Face recognition enhances smart home security by providing secure, seamless access control. Its widespread adoption in home security systems highlights its relevance as a representative application. The speech recognition application processes .wav and .flac audio files, comparing model outputs to ground truth transcripts.

Speech recognition enables hands-free control of smart home devices, enhancing accessibility for users with disabilities. This application was selected due to its widespread utility in user interaction and specific resource demands in real-time processing. The application first generates a JSON file containing known face embeddings from a subset of our dataset. Afterward, facial embeddings are extracted and matched against known faces using cosine similarity.

3.2 Software and Tools

This study employs multiple machine learning frameworks and models optimized for edge computing. TensorFlow is utilized for small operations such as image tensor conversion, while Vosk and ONNX facilitate efficient model loading and processing.

For speech recognition, we selected Vosk en-us 0.22-lgraph and Vosk small en-us 0.15. The former provides higher accuracy and quality, whereas the latter is optimized for lightweight deployment in constrained environments, as described in the official Vosk model documentation [11]. Since both models process full audio files, input size variations were tested by adjusting audio length rather than using additional models.

For face recognition, MobileFaceNet [13], FaceNet [12], and ArcResFaceNet [14] were chosen based on efficiency and accuracy trade-offs. MobileFaceNet, with around 1 million parameters, is ideal for resource-limited devices due to its compact size and efficiency. FaceNet, at approximately 22 million parameters, was included to assess performance at a larger 160×160 input resolution. ArcResFaceNet, the most computationally demanding with around 65 million parameters, maintains the 112×112 input size but features a more advanced architecture.

4 EXPERIMENTAL SETUP

In this part, we elaborate on our analysis. We conducted a comprehensive analysis taking into account various factors:

- **Input Variability:** Testing with datasets of varying input sizes (e.g., different image resolutions, audio lengths).
- **Model Complexity:** Comparing lightweight (e.g., MobileFaceNet) vs. complex models (e.g., ArcResFaceNet).
- **System Load:** Simulating multi-tasking by running a script putting load on the CPU to simulate real-world usage.

4.1 Hardware

To analyze how hardware platforms affect resource utilization and performance in machine learning applications, we will conduct experiments on the following platforms:

- **Raspberry Pi 4 model B:** Represents low-power edge devices typical of smart home systems. It is equipped with a quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz and 4GB RAM.
- **PC:** Simulates a more powerful edge server environment. It is equipped with a Ryzen 5 3600, 16GB 3200MHz RAM, and an Nvidia RTX 3070.

4.2 Applications

4.2.1 Speech Recognition. Each test run under perf is conducted using 1,000 audio files to ensure consistency in performance evaluation. The datasets used for testing are:

- **Google Speech Commands:** Contains short, one-word voice commands, representing small input sizes.
- **LibriSpeech:** Composed of full-sentence recordings from audiobooks, representing larger input sizes.

4.2.2 Face Recognition. To evaluate performance, we selected the Labeled Faces in the Wild (LFW) dataset, a widely used benchmark for face recognition. LFW contains over 13,000 images collected from the web, featuring diverse lighting conditions, poses, and image qualities, making it a robust choice for real-world applications. Performance measurements exclude the generation of the knownFaces.json file, focusing only on the embedding matching process while running the application under perf. The entire dataset is processed during profiling, with images resized to match each model's expected input dimensions.

4.3 Performance metrics

Performance is measured through CPU cycles, execution time, and memory usage. CPU cycles indicate processor efficiency, execution time ensures real-time feasibility, and memory usage helps identify potential bottlenecks. A Python script runs the machine learning applications under the perf utility, capturing relevant metrics such as CPU cycles, execution time, and context switches.

To simulate system load, we use a Python script that utilizes the multiprocessing module to generate a specified CPU load across all available cores. Each core runs a busy-wait loop for a proportion of time corresponding to the target load percentage. This approach allows for testing various levels of system load (e.g., 0%, 50%, 100%) to understand their effects on CPU cycles, execution time, and memory usage. By doing so, we can analyze performance trade-offs and identify potential bottlenecks in resource-constrained environments.

The Raspberry Pi runs Linux natively, while the PC uses WSL 2 to access profiling tools, ensuring compatibility in metric collection.

5 RESULTS

The effect of input size on resource utilization: To evaluate how varying input sizes affect resource usage, we analyzed CPU cycles and execution times in speech recognition using the Google and Libri datasets, as well as face recognition models operating at different input resolutions.

The results in this section are based on the measurements from the PC platform, as it provides a more stable reference for evaluating computational demands without the hardware constraints of the Raspberry Pi.

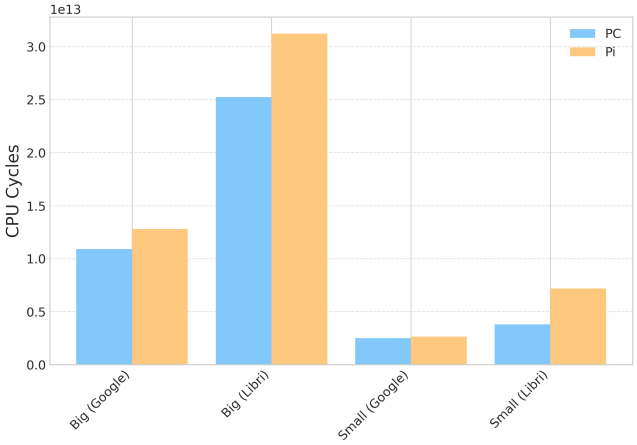

Figure 1: CPU cycles for speech recognition.

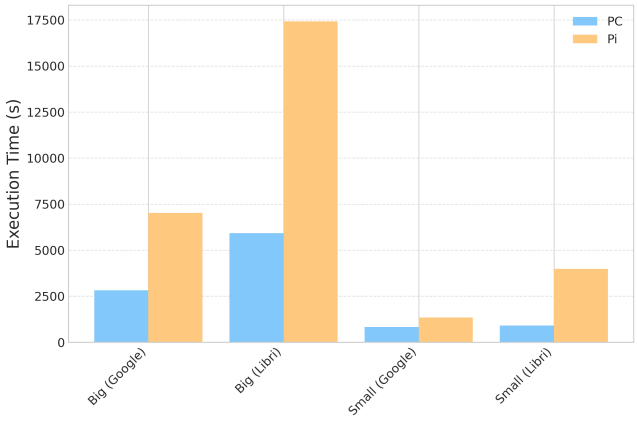

Figure 2: Execution time for speech recognition.

5.0.1 Speech Recognition. Figures 1 and 2 show the effect of varying input sizes on CPU cycles and execution time in speech recognition tasks. The Google dataset features shorter audio samples and thus requires fewer computational resources compared to the Libri dataset. For example, when using the small model, the Google dataset consumed about 2.5×10^{12} CPU cycles with an execution time of about 800 seconds, whereas the Libri dataset consumed about 3.8×10^{12} cycles (about a 50% increase) with an execution time of about 900 seconds (about a 10% increase). A similar pattern emerges with the big model, where Google required about 1.1×10^{13} CPU cycles and roughly 2800 seconds, while Libri used about 2.5×10^{13} cycles (about a 130% increase) and just over 5200 seconds (about an 80% increase in cycles). Execution time grew notably with the longer Libri samples, indicating that larger or more complex inputs significantly elevate resource consumption.

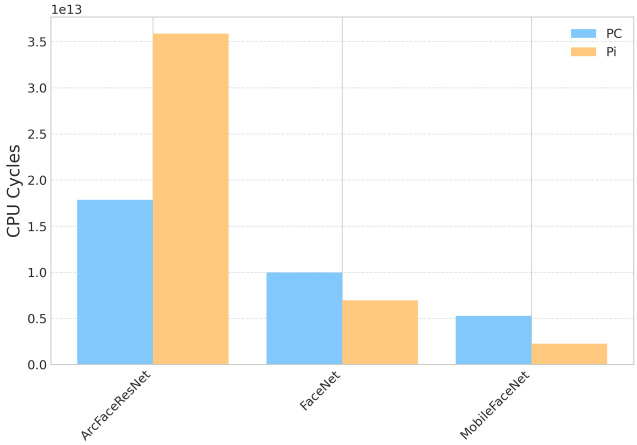

Figure 3: CPU cycles for face recognition.

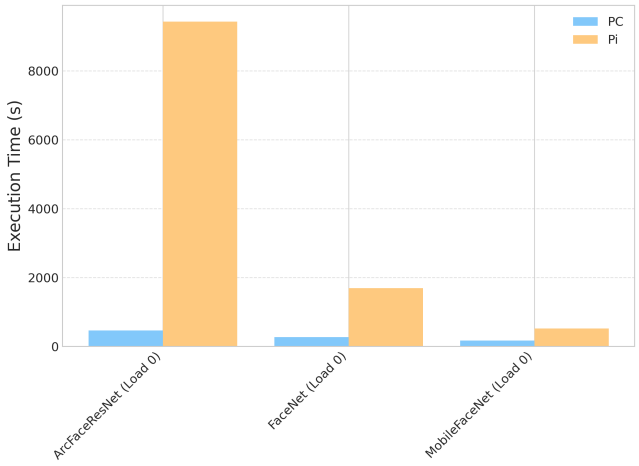

Figure 4: Execution time for face recognition.

5.0.2 Face Recognition. A similar trend is observed in face recognition, where differences in input resolution and model size both influence resource usage. Figures 3 and 4 show how varying input sizes affect CPU cycles and execution time in face recognition tasks. MobileFaceNet consumed about 5.3×10^{12} CPU cycles and took about 170 seconds, whereas FaceNet required about 1.0×10^{13} cycles and roughly 270 seconds, reflecting an increase of nearly 90% in cycles and about 60% in execution time.

Even at the same 112×112 input size, ArcFaceResNet used about 1.8×10^{13} cycles with a runtime of about 460 seconds, consuming over three times more cycles than MobileFaceNet. These results suggest that while larger input dimensions increase resource demands, model complexity and parameter count can have an even greater impact, regardless of input size.

The effect of model complexity on resource utilization: Next, we isolate the effect of model complexity by comparing smaller and larger architectures under similar conditions. For this section we also used the measurements from the PC platform, to not be constrained by hardware limitations.

5.0.3 Speech recognition. On the Google Speech Commands dataset, the small model used about 2.5×10^{12} CPU cycles and took roughly 800 seconds, while the big model required about 1.1×10^{13} cycles and about 2800 seconds, indicating around 4.3× more cycles and about 3.4× longer execution time. On the LibriSpeech dataset, the disparity grew even larger, with the big model consuming roughly 6.6× more CPU cycles and taking about 6.6× longer to run than the small model. These results demonstrate that bigger, more accurate speech recognition models incur substantially higher computational costs.

5.0.4 Face Recognition. Similarly, face recognition models with the same 112×112 input can vary widely in complexity. MobileFaceNet used about 5.3×10^{12} CPU cycles (about 170 seconds), whereas ArcFaceResNet reached about 1.8×10^{13} cycles (about 460 seconds), approximately 3.4× and nearly 3× higher, respectively. Comparing FaceNet to ArcFaceResNet shows that ArcFaceResNet still consumes about 1.8× more CPU cycles and takes about 1.7× longer, underscoring that deeper architectures typically place greater demands on resources than higher-resolution inputs alone.

The effect of workload conditions on resource utilization: This section examines how increasing system load affects execution time and CPU cycles for both voice and face recognition applications. We compare performance at different load levels (0%, 50%, and 100%).

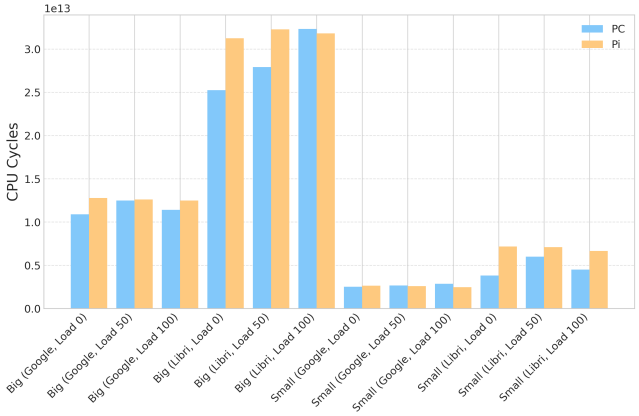

Figure 5: CPU cycles comparison under load for speech recognition.

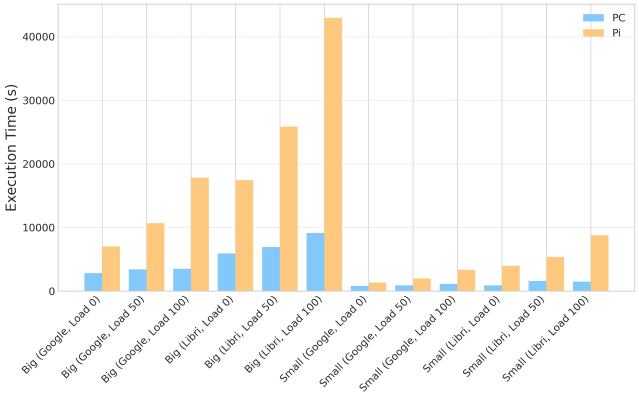

Figure 6: Execution time comparison under load for speech recognition.

5.0.5 Speech recognition. Figures 5 and 6 show the effects of system load on CPU cycles and execution time in speech recognition tasks. Under increasing system load, both small and big speech recognition models showed significant increases in execution time. On the Raspberry Pi, using the Google dataset, the small model's runtime rose from about 1300 seconds at 0% load to approximately 3300 seconds at 100% load, while CPU cycles remained relatively stable. This suggests that the additional overhead was primarily due to scheduling and context switching, which increased from about 1.1×10^6 at 0% load to about 3.4×10^6 at 100% load, meaning around a 3× increase. A similar but more severe pattern was

observed with the Libri dataset, where the small model's execution time increased from about 700 to just over 3400 seconds under full load. The big model also exhibited substantial slowdowns, with higher scheduling overhead and context switching as workload increased.

Despite the system load being applied equally across all CPU threads, the PC experienced a much smaller relative increase in execution time compared to the Raspberry Pi. However, on both platforms, execution time increased significantly with higher model complexity and larger input sizes, growing by about 1.5× on the PC and roughly 2.5× on the Raspberry Pi under full load.

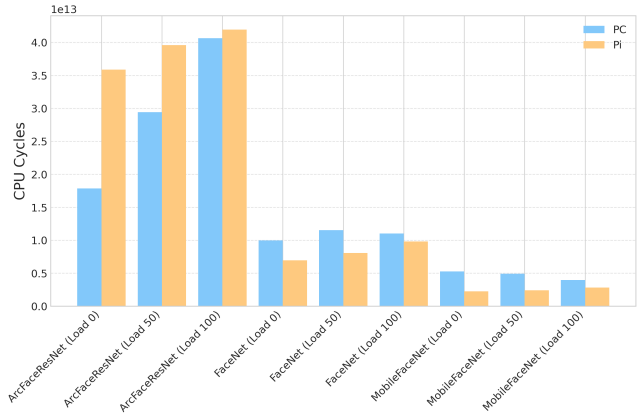

Figure 7: CPU cycles comparison under load for face recognition.

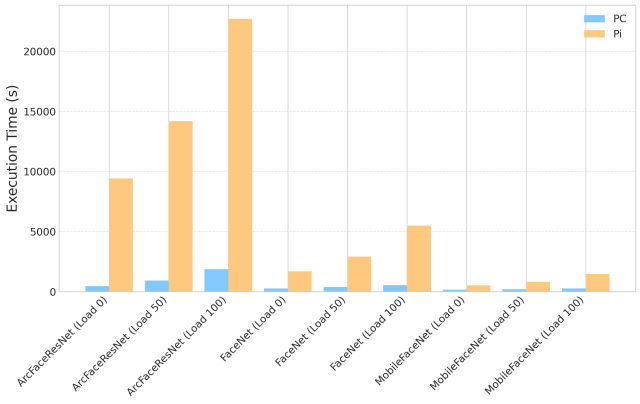

Figure 8: Execution time comparison under load for face recognition.

5.0.6 Face Recognition. Figures 7 and 8 show the effects of system load on CPU cycles and execution time in speech recognition tasks. By analyzing them, we observe that face recognition follows the same trend, with progressively longer runtimes under heavier system loads. MobileFaceNet, being comparatively lightweight, experienced smaller relative increases than FaceNet or ArcFaceResNet. In the more complex models, execution times escalated sharply at higher loads, reflecting the compound effect of greater model complexity and additional scheduling overhead.

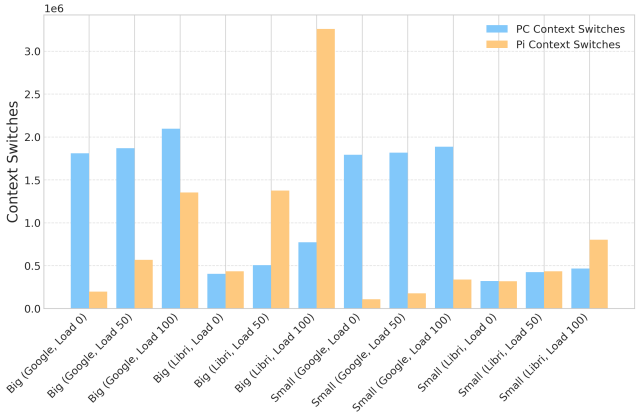

Figure 9: Context switches comparison under load for speech recognition.

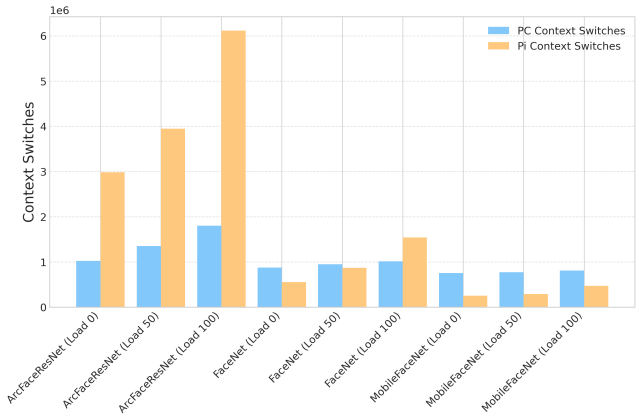

Figure 10: Context switches comparison under load for face recognition.

Performance across hardware platforms: System load had a significantly greater impact on the Raspberry Pi compared to the PC, particularly in terms of execution time, as shown in Figures 2 and 4. Across all tested conditions, execution times were consistently lower on the PC, despite CPU cycle counts fluctuating between the two platforms.

A notable difference between the two platforms was the behavior of context switching under load. On the Raspberry Pi, context switch counts increased sharply as system load increased, while on the PC, context switches remained relatively stable across different load conditions (Figures 9 and 10).

Additionally, CPU cycle measurements were more consistent on the Raspberry Pi, whereas the PC exhibited greater variation across runs, especially in face recognition tasks. These findings indicate that execution time, context switching, and CPU cycle

stability differ notably between the Raspberry Pi and PC, with the Raspberry Pi showing greater sensitivity to system load.

6 CONCLUSION

This study explored how input size, model complexity, and system load affect resource usage and performance in smart home machine learning applications. We analyzed speech and face recognition on a Raspberry Pi and a PC, testing different input sizes, model complexities, and workload conditions.

Our results show that larger inputs and more complex models significantly increase computational demands, leading to higher CPU cycles and longer execution times. System load further amplified these effects, with the Raspberry Pi experiencing greater slowdowns under heavy workloads, while the PC handled load more efficiently, maintaining stable context switches and lower execution times.

Interestingly, CPU cycle measurements were more consistent on the Raspberry Pi but varied on the PC, particularly in face recognition tasks. This is likely due to dynamic frequency scaling (DVFS) and cache behavior on the PC, where the CPU adjusts its clock speed based on workload and thermal conditions. The Raspberry Pi's fixed clock speed and simpler architecture contributed to more stable performance.

One limitation of this study is that each experiment was only run once per condition. On the PC, factors like background processes, cache effects, and dynamic scheduling can impact execution, meaning repeated runs and averaging results could provide more stable and reliable CPU cycle measurements. Future work should account for this variability by incorporating multiple trials.

While WSL 2's virtualization and Windows' resource management may introduce some additional variation, our findings suggest that hardware-level factors, rather than OS scheduling alone, are the primary cause of CPU cycle fluctuations. Future research could explore alternative schedulers, hardware accelerators, and task offloading strategies to improve performance consistency. Additionally, profiling power consumption, thermal behavior, and real-time execution across a wider range of edge devices would help optimize machine learning workloads for resource-constrained smart home environments.

REFERENCES

[1] Y. Liang, D. O'Keeffe, and N. Sastry, *PAIGE*, in *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*, EuroSys '20: Fifteenth EuroSys Conference 2020, ACM, pp. 55–60, 2020, doi: 10.1145/3378679.3394536.

[2] X. Zhou, R. E. Canady, S. Bao, and A. S. Gokhale, "Cost-effective Hardware Accelerator Recommendation for Edge Computing," USENIX Workshop on Hot Topics in Edge Computing, 2020.

[3] Q. Liang, P. Shenoy, and D. Irwin, "AI on the Edge: Characterizing AI-based IoT Applications Using Specialized Edge Architectures," in *2020 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 145–156, IEEE, 2020, doi: 10.1109/iiswc50251.2020.00023.

[4] P. Wang, "Research and Design of Smart Home Speech Recognition System Based on Deep Learning," in *2020 International Conference on Computer Vision, Image and Deep Learning (CVIDL)*, IEEE, 2020, doi: 10.1109/cvidl51233.2020.00-98.

[5] S. Khunchai and C. Thongchaisuratkrul, "Development of Application and Face Recognition for Smart Home," in *2020 International Conference on Power, Energy and Innovations (ICPEI)*, pp. 105–108, IEEE, 2020, doi: 10.1109/icpei49860.2020.9431473.

[6] Y. Xi, P. Chen, and Z. Fu, "Research on Fall Detection Method of Empty-nesters Based on Computer Vision," in *2022 IEEE 4th Eurasia Conference on Biomedical Engineering, Healthcare and Sustainability (ECBIOS)*, pp. 232–235, IEEE, 2022, doi: 10.1109/ecbios54627.2022.9945007.

[7] A. Murali, N. N. Das, S. S. Sukumaran, K. Chandrasekaran, C. Joseph, and J. P. Martin, "Machine Learning Approaches for Resource Allocation in the Cloud: Critical Reflections," in *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 2073–2079, IEEE, 2018, doi: 10.1109/icacci.2018.8554703.

[8] B. Charyyev, E. Arslan, and M. H. Gunes, "Latency Comparison of Cloud Datacenters and Edge Servers," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, pp. 1–6, IEEE, 2020, doi: 10.1109/globecom42002.2020.9322406.

[9] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos, "Challenges and Opportunities in Edge Computing," in *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, pp. 20–26, IEEE, 2016, doi: 10.1109/smartcloud.2016.18.

[10] S. Taheri-abed, A. M. Eftekhari Moghadam, and M. H. Rezvani, "Machine learning-based computation offloading in edge and fog: a systematic review," in *Cluster Computing*, vol. 26, issue 5, pp. 3113–3144, Springer Science and Business Media LLC, 2023, doi: 10.1007/s10586-023-04100-z.

[11] AlphaCephei, "Vosk Models," https://alphacephei.com/vosk/models, accessed Dec. 2024.

[12] FaceONNX, "FaceONNX: Face Recognition using FaceNet," https://github.com/FaceONNX/FaceONNX, accessed Dec. 2024.

[13] OpenVINO, "Open Model Zoo: MobileFaceNet," https://github.com/openvinotoolkit/open_model_zoo/tree/master, accessed Dec. 2024.

[14] ONNX Models, "ArcResFaceNet in ONNX Model Zoo," https://github.com/onnx/models, accessed Dec. 2024.