國立臺灣大學工學院土木工程學系

碩士論文

Department of Civil Engineering

College of Engineering

National Taiwan University

Master's Thesis

大型語言模型推論中語意快取的能耗與效能權衡：相似度閾值調整的實證分析

The Energy-Performance Trade-Off of Semantic Caching for LLM Inference: An Empirical Analysis of Similarity Threshold Tuning

陳冠錞

Kuan-Chun Chen

指導教授: 謝尚賢 博士

Advisor: Shang-Hsien Hsieh Ph.D.

中華民國 114 年 8 月

August, 2025

doi:10.6342/NTU202504158

致謝

首先一定要感謝我的指導教授謝尚賢老師，非常支持我做任何我當下想執行的生涯規劃，儘管跟學術研究、土木工程領域較無關，老師還是能給出很多關於職涯上的建議與鼓勵。我想當初被老師錄取，老師應該也不是期待我走向軟體工程師的路途，最終的論文題目也和最初碩班申請時的設定相差甚遠，但老師還是非常鼓勵我完成。另外也一定要感謝 Tracey，在暑期實習階段對我的栽培以及永續課題的經驗傳授（某種程度上也是讓我接觸到「永續」這個領域，後續才會想融合軟體工程所見所聞，做永續軟體工程相關的探索），也幫助我融入 CAE 和 H-Team 這個大家庭；雖然剛進碩一沒多久就因為另個產學合作案的關係而沒有持續合作，但後來的 I3CE 研討會還是受到 Tracey 相當大的協助。謝謝之謙老師給我接觸開發 App 的機會，雖然剛開始我什麼都不會，只會非常基本的前後端觀念，但隨著產學合作案的進展，也慢慢體會到接案開發的各種眉眉角角，並終於把前後端、雲端部署都用到的 App 甚至網頁打造出來，在這個開發案學到很多。感謝湘如每次我有什麼疑難雜症都能迅速幫我解答或處理！

謝謝 CAE 研究室的夥伴們，我雖然不常進研究室，但只要有到學校，永遠都被吵著要出去玩或吃飯。謝謝宏發、沛忻一起在產學合作案裡做開發和研究，到後來成了應該算最好的朋友，可以互相抱怨感情事（大部分都是我在聽，很有趣）；謝謝麒凌、昊天、唯一和元璽常常一起出去喝茶（或酒?），可惜畢旅沒有機會一起去峇里島，之後一定要再一起出去玩。謝謝筠曼推薦不少有趣的電子書，

doi:10.6342/NTU202504158

還有最後幫我解答了很多要跑的文件跟流程，不然我可能會畢不了業。謝謝測量組的詒雯、Andy，陪伴我度過幾乎整整三年的碩班生活，也因為認識你們，我才有那麼不一樣的職涯視野。謝謝大學階段的好友們，得倫、家瑋、祐如、家齊、心蓓、振揚常常聽我講工作或找實習上的事情。謝謝在台北的高中老友們偶爾會把過度工作的我拖出門吃好料，放鬆心情，聊聊最近都在忙些什麼。謝謝 Jubo、國泰、LINE、AWS 以及 Cloud Native Taiwan User Group 所遇到的各位，寫到這裡才發現自己真的花了非常多時間在探索學術研究以外的領域，包括業界的軟體工程實務、雲端產業、開源貢獻文化等，也是感謝自己有勇敢踏出去，才有機會了解人生中真正想做的事情。其中特別想感謝以文學長，是我軟體工程職涯中的第一個 Mentor，把當時非常菜的我拉進 Jubo，從頭接觸、學習業界才有的軟體工程流程，可以說沒有 Jubo 實習期間帶給我的視野、人脈、經驗以及實力的提升，就沒有後來到其他更大規模的軟體公司或社群實習的機會。

最後也要感謝謝依芸老師以及張慰慈老師願意擔任我的口試委員，最初在寄信時很擔心會得不到回覆，因為自己在 CAE 組內應該快要算幽靈人口了；口試之前也很擔心會被問倒，但兩位委員都非常親切地給出不論是文章撰寫上或是內容概念上的建議，甚至在口試正式開始前還有一些有趣的討論和延伸發想，讓我看到這個研究的不足之處。

doi:10.6342/NTU202504158

摘要

隨著大型語言模型（LLM）的突破性發展，其龐大的運算需求已成為全球資料中心能源消耗與碳排放增長的主要驅動力。與傳統的批次處理任務不同，互動式 LLM 應用程式不僅是能源、計算密集型任務，同時也需滿足使用者對低延遲服務品質（QoS）的要求，這對於雲端原生環境下的資源調度帶來一定程度的挑戰。快取機制（Cache Mechanism）在現今的網路世界被大量採用，而在大型語言模型應用程式數量快速增長下，語意快取（Semantic Caching）的出現也被視為一種降低延遲與成本的有效技術；不同於傳統快取機制多半要求完全符合（Exact Match），語意快取透過詞向量（Embeddings）與相似度（Similarity）的距離計算達成；然而，其對於系統總體能耗的真實影響，尤其是在特定參數設定下的權衡關係，目前學術界尚缺乏深入的實證研究。

本研究目的在於量化並分析語意快取對 Kubernetes 環境下 LLM 應用程式能耗的影響。本研究在 GKE（Google Kubernetes Engine）上搭建了一個由 Ollama（Mistral-7B）推論服務、GPTCache 向量快取機制構成的實驗平台，以模擬真實世界的雲端原生環境，並整合 Kepler（Kubernetes-based Efficient Power Level Exporter）與 NVIDIA DCGM（Data Center GPU Manager）監控工具，以實現對 CPU、DRAM 及 GPU 功耗的全面性量測。研究核心在於系統性地評估 GPTCache 的相似度閾值（Similarity Threshold）此關鍵參數，對於系統的快取命中率、回應時間與總體能耗所造成的非線性影響。

doi:10.6342/NTU202504158

實驗結果顯示，語意快取的節能效益與其參數設定高度相關。當相似度閾值設定在一個較寬鬆的區間（0.7），高達 99.99% 的快取命中率能成功規避後端高耗能的 GPU 推論，使系統平均功率從 155.877 W 降低至 88.78 W（下降約 43%），換算在整個實驗週期的總能耗更是從 93,256 焦耳大幅減少至 52,487 焦耳，並將平均回應時間從 30,294 毫秒大幅縮短至約 105 毫秒。然而，隨著閾值變得嚴苛，系統效能開始下降，並在閾值約 0.85 時出現轉折，此時總平均功率（175.22 W）已超過基準組，對應的實驗週期內的總能耗也比基準組多出 11,604 焦耳。當閾值設定為 0.95 時，快取命中率驟降至 29.4%，並且系統平均回應時間達到 29,035 毫秒，僅比基準組的 30,294 毫秒低 4%，導致系統不僅要承擔 GPU 推論的完整成本，還付出因查詢向量化所產生的額外開銷，最終使總能耗增加超過 12%，達到 105,130 焦耳，證實了不當的快取設定反而更耗能。

本研究首次量化了語意快取在 LLM 應用中能源效益的特性，實證對快取參數的適當設定是實現節能目標、避免負面效果的前提之一。本研究成果不僅為 LLM 應用的能耗優化提供初步的探索研究，也為後續在 Kubernetes 平台上進行透過語意相似度、回應時間、能耗等指標調度大型語言模型工作負載的研究奠定基礎。

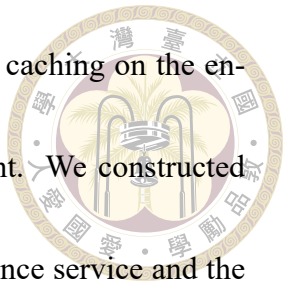
關鍵字：語意向量快取、永續人工智慧、大型語言模型推論、雲端原生架構、能源管理

doi:10.6342/NTU202504158

Abstract

With the breakthrough development of Large Language Models (LLM), their immense computational requirements have become a primary driver of the growth in energy consumption and carbon emissions in global data centers. Unlike traditional batch processing tasks, interactive LLM applications are not only energy- and compute-intensive but must also meet user demands for low-latency Quality of Service (QoS), which presents significant challenges for resource scheduling in cloud-native environments. Caching mechanisms are widely adopted in the modern web, and with the rapid growth in LLM applications, semantic caching has emerged as an effective technique for reducing latency and costs. Different from traditional caching, which mostly requires exact matching, semantic caching operates through the computation of distance and similarity in word embeddings. However, its actual impact on total system energy consumption, especially the trade-offs under specific parameter settings, currently lacks in-depth empirical research in academia.

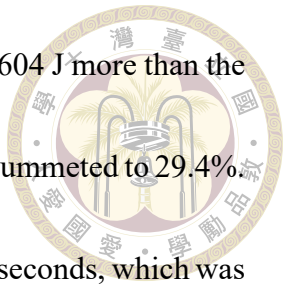
doi:10.6342/NTU202504158



This study aims to quantify and analyze the impact of semantic caching on the energy consumption of LLM applications in a Kubernetes environment. We constructed an experimental platform consisting of an Ollama (Mistral-7B) inference service and the GPTCache vector cache mechanism, integrated with Kepler (Kubernetes-based Efficient Power Level Exporter) and NVIDIA DCGM (Data Center GPU Manager) monitoring tools to achieve comprehensive measurement of CPU, DRAM, and GPU power consumption. The core of the research involves a systematic evaluation of how a key parameter of GPTCache, the similarity threshold, non-linearly affects the system's cache hit rate, response time, and overall energy consumption.

The experimental results demonstrate that the energy efficiency of semantic caching is highly conditional on its parameter configuration. When the similarity threshold was set to a lenient value (0.7), a high cache hit rate of 99.99% successfully circumvented the high-energy GPU inference, reducing average system power from 155.88 W to 88.78 W (a decrease of over 43%), which translates to a reduction in total energy consumption during the experiment from 93,256 J to 52,487 J and drastically improved the average response time from 30,294 ms to approximately 105 ms. However, as the threshold became stricter, performance degraded, reaching an inflection point at a threshold of 0.85, where the total average power consumption (175.22 W) surpassed the baseline, and the total energy

doi:10.6342/NTU202504158



consumption during the corresponding experimental period is also 11,604 J more than the baseline group. When the threshold was set to 0.95, the cache hit rate plummeted to 29.4%. Concurrently, the system's average response time reached 29,035 milliseconds, which was only 4% lower than the baseline group's 30,294 milliseconds. This led to the system not only bearing the full cost of GPU inference but also incurring additional overhead from query vectorization, ultimately increasing total energy consumption by more than 12% to 105,130 J, proving that improper cache configuration is more energy-intensive.

This research provides the first quantitative characterization of the energy efficiency of semantic caching in LLM applications, empirically demonstrating that the appropriate configuration of cache parameters is a prerequisite for achieving energy-saving goals and avoiding negative effects. The findings of this study not only offer a preliminary exploratory investigation into the energy consumption optimization of LLM applications but also lay the groundwork for future research on scheduling LLM workloads on the Kubernetes platform using metrics such as semantic similarity, response time, and energy consumption.

Keywords: Semantic Vector Caching, Sustainable Artificial Intelligence, Large Language Model Inference, Cloud Native Architecture, Energy Management

doi:10.6342/NTU202504158

Contents

doi:10.6342/NTU202504158

doi:10.6342/NTU202504158

List of Figures

doi:10.6342/NTU202504158

List of Tables

doi:10.6342/NTU202504158

Chapter 1 Introduction

1.1 Motivation

The Information and Communication Technology (ICT) sector has experienced exponential growth, becoming deeply integrated into the fabric of modern society. This expansion, however, has been accompanied by a significant and growing environmental footprint. Recent peer-reviewed estimates place the ICT sector's contribution at 1.8-2.8% of global greenhouse gas (GHG) emissions, a figure that could be as high as 2.1-3.9% when accounting for the full supply chain [1]. A significant driver of this energy demand environmental impact is the proliferation of large-scale data centers, which are necessary to power cloud computing services. The global electricity consumption of these facilities has grown by 20-40% annually in recent years, reaching up to 1.3% of total global electricity demand by 2022 [2].

Within this landscape, the rise of Artificial Intelligence (AI) and Machine Learning (ML) has emerged as a particularly energy-intensive domain, driving a super-linear growth in data volume, model complexity, and infrastructure capacity [3]. While the energy costs of model training have been extensively documented, the subsequent inference phase—where trained models are deployed to serve user queries—is now understood to constitute the majority of ML-related energy demand. Industry analyses estimate that in-

doi:10.6342/NTU202504158

ference accounts for 80-90% of ML cloud computing demand [2]. Other reports from major technology companies place this figure between 60-70% of their total ML energy use [4]. As Large Language Models (LLMs) are increasingly deployed in user-facing, latency-sensitive applications that serve billions of queries daily, developing strategies to mitigate the energy consumption of inference has become a critical challenge for sustainable software engineering.

In response to these energy concerns, the field of green software engineering has developed various strategies to create energy-efficient systems. While many strategies are framed as creating "carbon-aware" systems, their effectiveness is fundamentally predicated on managing energy consumption, as energy usage is a primary determinant of operational carbon emissions. These approaches often involve spatial shifting, which relocates computational workloads to data centers in geographical regions with cleaner energy grids [5], and temporal shifting, which adjusts job execution times to coincide with periods of low carbon intensity through methods like suspend-resume or dynamic resource scaling. For the specific domain of LLMs, research has focused on similar energy reduction techniques, such as routing inference requests to data centers with lower power usage effectiveness [6] or using generation directives to produce more concise, and thus less energy-intensive, responses [7]. Another prominent strategy is the use of semantic caching, which aims to reduce the number of expensive, GPU-intensive calls to the LLM by storing and reusing answers to semantically identical queries [8].

doi:10.6342/NTU202504158

1.2 Research Objectives

While strategies such as temporal and spatial workload shifting have shown promise for reducing the environmental impact of delay-tolerant batch processes, these methods are often unsuitable for latency-sensitive, interactive applications like the inference services for LLMs. These applications require immediate responses to maintain Quality of Service (QoS), making it imperative to find solutions that reduce energy consumption without introducing prohibitive delays. To address this challenge, semantic caching has emerged as a viable technique to reduce the operational cost and improve the responsiveness of LLM services by storing and reusing the results of semantically similar queries, thereby avoiding energy-intensive calls to the underlying model.

However, the direct impact of this technology on system-level energy consumption remains under-quantified. The efficacy of a semantic cache, such as GPTCache [8], is critically dependent on its configuration, particularly the `similarity_threshold` that determines a cache hit. An improperly configured threshold risks introducing computational overhead that could negate or even reverse potential energy savings. Furthermore, in this study, the term ”performance” is explicitly defined by **response time (or latency)**. The objective is to move beyond ambiguous terminology and analyze the specific trade-offs between energy consumption and these precise performance indicators. To ensure the academic rigor and clarity of our contributions, it is essential to strictly define the scope of this research. This study's experimental design is specifically tailored for **single-turn, domain-specific question-answering (Q&A) applications**. This scenario is analogous to practical use cases such as Frequently Asked Questions (FAQ) systems or specialized AI agents for a narrow knowledge domain. We acknowledge that the caching mecha-

doi:10.6342/NTU202504158

nisms for more complex applications involving multi-turn dialogue, multi-modal data, or cross-domain reasoning present challenges far beyond the scope of this work. By clearly delineating this boundary, we aim to provide a focused and robust analysis whose findings can serve as a reliable foundation for future, more complex investigations.

To bridge the identified research gaps, this study was structured with the following research objectives:

- To establish an empirical energy consumption baseline for a standard, non-cached LLM inference service to serve as a benchmark for quantifying the net energy savings or costs introduced by the semantic caching layer.
- To quantitatively measure and analyze how different `similarity_threshold` settings in a semantic cache influence the total energy consumption of a LLM inference service deployed in a cloud-native environment.
- To evaluate the performance and power trade-offs associated with different threshold configurations, specifically analyzing the relationship between cache hit rate, average response time, and overall system power consumption.

1.3 Organization of Thesis

This thesis is organized into five chapters to systematically address the research problem. This first chapter provides an introduction to the research, establishing the motivation, defining the core research objectives, and outlining the structure of the document. Chapter 2 presents a comprehensive literature review, discussing the core principles of green software, the significance of cloud-native architectures, existing research on carbon-

doi:10.6342/NTU202504158

aware systems and LLM inference optimization, and the state of observability tools for energy monitoring. Subsequently, Chapter 3 details the methodology employed in this study, describing the system architecture, experimental setup, the selection of the primary independent variable, and the procedures for data collection and energy measurement. Chapter 4 then presents the empirical results and a thorough discussion of the findings, analyzing the measured power consumption and performance metrics across different experimental configurations to reveal the non-linear relationship between the cache's similarity threshold and system efficiency. Finally, Chapter 5 concludes the thesis by summarizing the key findings, highlighting the study's contributions to the field of sustainable software engineering, and proposing promising avenues for future research.

doi:10.6342/NTU202504158

Chapter 2 Literature Review

This study focuses on evaluating and analyzing the energy consumption of a LLM inference service within a cloud-native architecture. Consequently, this section will succinctly delineate the core concepts and key tenets of green software, along with elucidating the significance of cloud-native architectures. Concurrently, it will review existing research concerning software carbon emissions and current studies on energy consumption optimization for LLM inference, and observability tools to monitor energy consumption regarding research are also reviewed at the end of the section.

2.1 Green Software

Green Software is an emergent and interdisciplinary field focused on minimizing the environmental impact of software by considering its entire life-cycle, from design and development to deployment and eventual decommissioning. The Green Software Foundation defines it as software that is responsible for emitting fewer greenhouse gases. This is achieved by adhering to a core set of principles, primarily centered on carbon efficiency, energy efficiency, and hardware efficiency [9]. Carbon efficiency involves running software at times and in locations where the carbon intensity of the electrical grid is lowest, a practice known as carbon awareness. Energy efficiency focuses on consuming the least

doi:10.6342/NTU202504158

amount of electricity possible to perform a given task. Hardware efficiency aims to minimize the embodied carbon, which is the carbon emitted during the manufacturing and disposal of hardware, by extending the life of existing hardware and improving hardware utilization.

These principles are put into practice through the application of Green Software patterns [10], which are reusable, vendor-neutral solutions to common problems in software engineering. Examples of such patterns include demand shaping, where application behavior is adjusted based on the availability of renewable energy, and energy-proportional computing, which ensures that the energy consumed by a system is proportional to the work it is performing. The concrete implementation of these patterns within a specific technological context or vendor's product is referred to as a Green Software practice. These practices can range from optimizing algorithms and data structures to reduce CPU cycles, to designing applications that can run effectively on older, less powerful hardware. The overarching goal of these principles, patterns, and practices is to systematically reduce the carbon footprint of software, contributing to broader sustainability goals.

2.2 Cloud Native Deployment

The significant computational and resource demands of LLMs pose substantial deployment challenges that traditional infrastructures cannot adequately address [11]. In response, cloud-native architecture has become the prevailing paradigm, leveraging principles of containerization, microservices, and orchestration to build scalable and efficient AI systems [12]. This approach decomposes complex applications into independent microservices, which are then packaged into lightweight containers to ensure consistent and

doi:10.6342/NTU202504158

portable deployment across diverse environments [13]. This modularity allows for the independent scaling of components like data processing and model inference, thereby enhancing system resilience [14].

Orchestration platforms such as Kubernetes are essential for managing these containerized services at scale, providing automated deployment, lifecycle management, and declarative resource control [15]. A primary benefit of this architecture for LLM inference is its inherent elasticity. Governed by an orchestrator, cloud-native systems can employ auto-scaling to dynamically adjust the number of service instances in response to real-time workload fluctuations [11, 14]. This ensures resources are provisioned only when needed, which is fundamental to minimizing financial costs and energy consumption from over-provisioning [11, 13].

Furthermore, the cloud-native environment facilitates fine-grained resource configuration, which is critical for optimization. Efficiently deploying an inference service requires navigating a vast configuration space, including CPU cores, GPU memory, and runtime parameters like batch size. Studies show that an optimal configuration can yield over a tenfold increase in performance and significantly improve resource utilization, directly impacting costs and energy use [16]. The convergence of these capabilities is paving the way for an ”AI-native” paradigm, where a deeper integration between ML runtimes and cloud systems enables advanced optimizations. Techniques such as multi-tenancy, which allows multiple tasks to share common infrastructure, can dramatically improve resource usage and throughput, further solidifying the cloud-native approach as a cornerstone for efficient and sustainable AI deployment [11].

doi:10.6342/NTU202504158

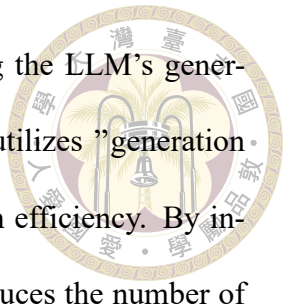
2.3 Carbon Awareness Large Language Model Inference

The escalating computational demand of Generative Artificial Intelligence (GenAI) has brought significant environmental concerns to the forefront, particularly regarding the carbon emissions associated with the inference phase of Large Language Models (LLMs). While model training is an energy-intensive, one-time cost, the operational inference phase, which serves billions of user requests, is poised to become the predominant and continuous source of carbon emissions [6, 7]. Consequently, a substantial body of research has emerged to develop strategies that mitigate the carbon footprint of LLM inference services without compromising performance or the quality of generation. These strategies can be broadly categorized into direct carbon reduction techniques and performance optimizations through semantic caching.

2.3.1 Direct Carbon Reduction Strategies

Direct strategies for carbon reduction in LLM inference aim to modulate the execution of queries based on real-time environmental factors or by altering the generation process itself. One prominent approach is the geographical shifting of computational workloads. Chien et al. [6] propose a workload model for ChatGPT-like services and demonstrate that intelligently directing inference requests to data centers in regions with lower power grid carbon intensity can yield substantial emission reductions. Their CarbonMin algorithm, which routes requests to the greenest available locations, was shown to reduce carbon emissions by 35% under current conditions and up to 56% in a projected 2035 scenario, all while maintaining the quality of service (QoS) for user-facing applications.

doi:10.6342/NTU202504158



An alternative, application-level strategy involves manipulating the LLM's generative process. Li et al. [7] introduce SPROUT, a framework that utilizes "generation directives" to guide the autoregressive process toward greater carbon efficiency. By instructing the model to provide more concise responses, SPROUT reduces the number of generated tokens—a metric that exhibits a strong linear correlation with carbon emissions. This approach circumvents the need to downsize the model, thereby preserving its contextual understanding capabilities while achieving a carbon reduction of over 40%. The framework employs a directive optimizer and an offline quality evaluator to balance sustainability with high-quality outcomes.

2.3.2 Semantic Caching for Cost and Carbon Reduction

Another principal strategy to mitigate the computational and carbon cost of LLM inference is caching. By storing and reusing the results of previous queries, caching systems can significantly reduce the number of expensive calls to the LLM, thereby saving energy, reducing latency, and lowering operational costs. However, traditional key-value caching mechanisms are inadequate for LLM applications due to the unstructured and semantically variant nature of user queries. For instance, a query for 'What is the distance between New York and Los Angeles?' is semantically identical to 'How far is LA from NYC?', yet would be treated as distinct by a traditional cache. This challenge necessitates the use of semantic caching, a concept explored in early web systems where query matching was used to avoid redundant access to data sources [17]. In the context of modern LLMs, this is achieved by transforming queries into high-dimensional vector embeddings and identifying cached entries through similarity searches in a vector space.

doi:10.6342/NTU202504158

An open-source implementation that has become a foundational tool in this domain is GPTCache. It employs embedding algorithms to convert queries into vectors and uses a vector store to conduct similarity searches. When a new query's embedding is sufficiently close to a cached embedding (based on a similarity threshold), the corresponding response is served directly from the cache, leading to significant improvements in response speed and cost savings [8]. Figure 2.1 displays the system architecture of GPTCache.

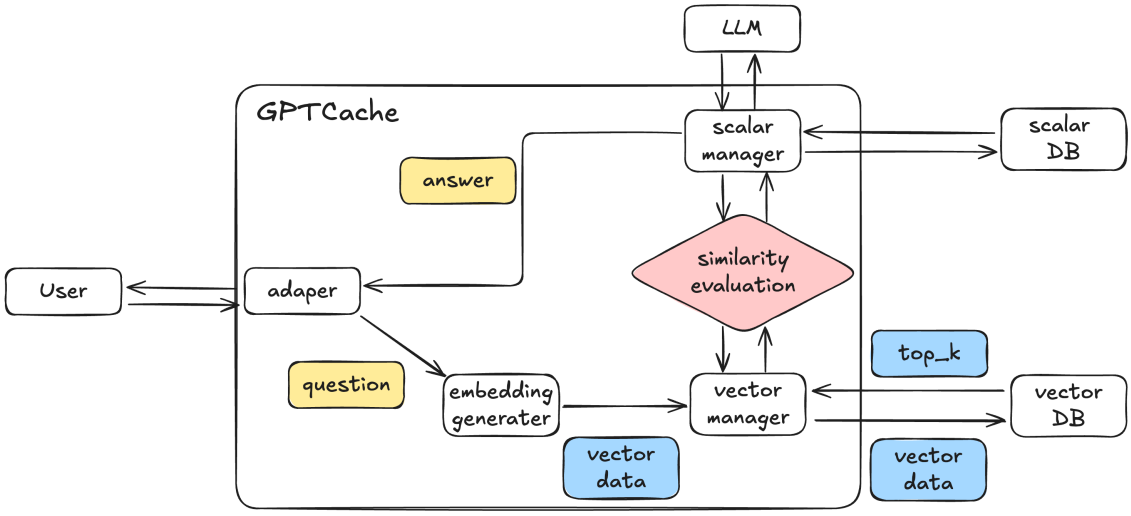


Figure 2.1: Architecture of GPTCache

Under the design of GPTCache, two conditions exist. In the event of a cache miss, user requests are forwarded to the LLM for inference. Following this, the original question is inserted into the vector database, and the LLM-generated answer is stored in the scalar database, as depicted in Figure 2.2. Conversely, upon a cache hit, indicating semantic similarity between the user's query and a previous question, the corresponding answer is directly retrieved from the scalar database and returned to the user, as depicted in Figure 2.3.

doi:10.6342/NTU202504158

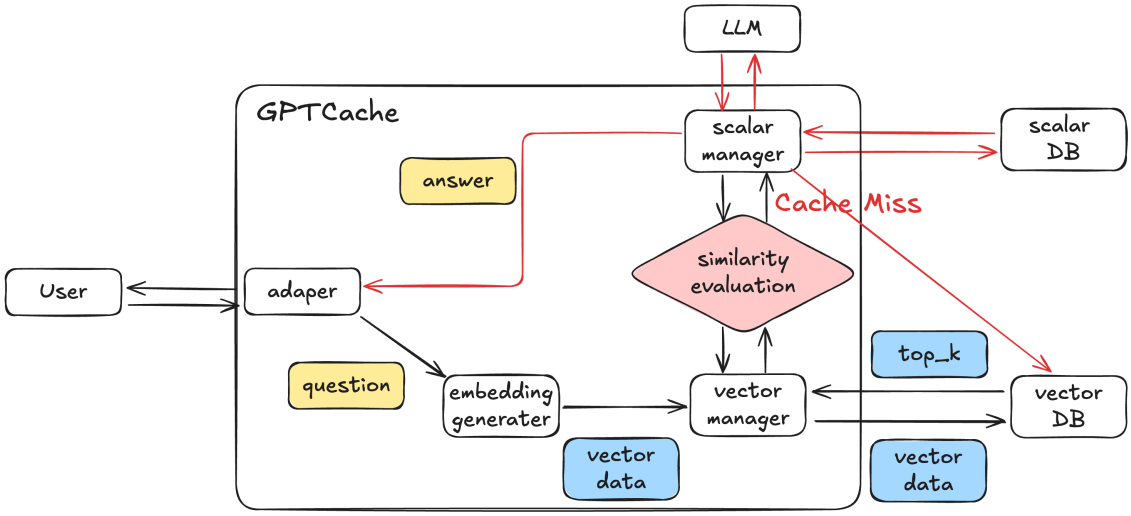


Figure 2.2: Cache Miss Scenario

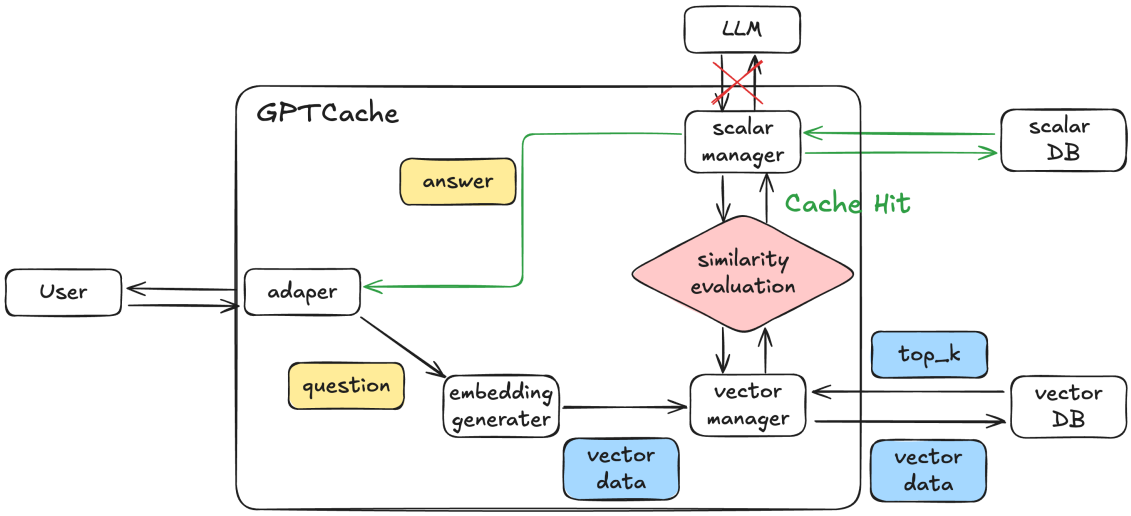


Figure 2.3: Cache Hit Scenario

While foundational, basic semantic caching has its limitations. Researchers have identified that a simple similarity search is often insufficient, leading to the development of more sophisticated, semantics-oriented caching architectures. Li et al. propose SCALM, a framework that moves beyond simple query matching to perform hierarchical semantic clustering on query data. By identifying frequently visited ”semantic patterns,” SCALM can rank queries based on their potential for cost savings and make more intelligent caching and eviction decisions. This approach yielded a relative increase of 63% in cache hit ratio and a 77% improvement in token savings compared to baseline GPTCache

doi:10.6342/NTU202504158

implementations [18]. Further refining this, Mohandoss notes that many user queries are not context-free and depend on factors like user identity, location, or role. His proposed context-based semantic caching design introduces a ”Context Hashkey” alongside the semantic vector, ensuring that cached responses are not only semantically similar but also contextually appropriate for the user, thereby preventing incorrect cache hits for personalized or location-sensitive queries [19].

The optimization of these systems can be further enhanced by integrating caching with other strategies. Zhu et al. [20] investigate the joint optimization of caching and model multiplexing. Their framework introduces a model multiplexer that, in the event of a cache miss, decides whether to route the query to a smaller, less expensive model or a larger, more powerful one based on estimated costs. By combining a Least Expected Cost (LEC) caching policy with this model selection mechanism, their system achieves a more holistic optimization of the cost-performance trade-off, demonstrating up to a 4.3x reduction in FLOPs on real-world datasets.

Above literature presents a clear trajectory from direct carbon reduction techniques to increasingly sophisticated semantic caching systems. These systems have evolved from basic vector similarity lookups to architectures that incorporate semantic pattern analysis, user context, and joint optimization with model selection. While these studies establish the efficacy of various carbon and cost reduction techniques, a detailed analysis of how the similarity threshold within a semantic cache directly influences the energy consumption and carbon footprint of the inference service, particularly within a cloud-native deployment, remains less explored. This thesis aims to address this gap by quantifying the energy impact of varying this critical hyperparameter, providing a new dimension to the understanding of sustainable GenAI operations.

doi:10.6342/NTU202504158

2.4 Observability

In cloud-native systems, standard observability stacks featuring Prometheus and Grafana excel at monitoring general resource metrics but lack native capabilities for energy consumption analysis [21]. Accurately attributing power draw to individual containerized workloads is a significant challenge. While hardware-based power meters provide a high-precision ground truth, their cost, scalability issues, and inability to offer process-level granularity make them impractical for fine-grained analysis in production environments [21, 22]. Consequently, research and industry have focused on software-based power meters that estimate energy usage.

These software tools primarily leverage hardware interfaces such as Intel's Running Average Power Limit (RAPL) for CPU and DRAM domains, and NVIDIA's Management Library (NVML) for GPUs [22]. Open-source solutions built for containerized environments exemplify the state-of-the-art. Kepler (Kubernetes-based Efficient Power Level Exporter), a Kubernetes-native exporter, uses the extended Berkeley Packet Filter (eBPF) to efficiently collect hardware performance counters, applying regression models to estimate power at the pod level [23]. In contrast, tools like Scaphandre employ a simpler, usage-based model that correlates a process's CPU time with the node's total RAPL-reported energy.

However, the current landscape of energy observability is defined by several critical limitations. The most significant is the documented discrepancy between tools; due to differing estimation algorithms, various software meters produce divergent power consumption figures even when analyzing the same workload with the same underlying RAPL data [21, 22]. Furthermore, there is a fundamental trade-off between measurement accuracy

doi:10.6342/NTU202504158

and system overhead, as higher sampling frequencies that capture more detailed power profiles also induce greater CPU load. A final, major constraint is platform compatibility, with many prominent tools, including Kepler and Scaphandre, lacking robust support for ARM architectures. This complex and varied state of energy monitoring underscores the need for empirical research to carefully quantify the power consumption of novel workloads like LLM inference, using a consistent and well-understood measurement methodology.

doi:10.6342/NTU202504158

Chapter 3 Methodology

This chapter details the methodological foundation of the research, outlining the systematic approach employed to investigate the energy consumption and performance trade-offs of semantic caching for LLM inference services. It begins by defining the specific research problem within the context of sustainable software engineering. Subsequently, it describes the system architecture, experimental design, and the selection of the primary independent variable. Finally, it specifies the procedures for conducting the experiments and the precise methods for measuring and calculating energy consumption, ensuring the study's validity and replicability.

3.1 Statement of Problem

The increasing computational demand of LLMs has positioned them as a significant driver of energy consumption in data centers. While much of the research in GreenOps and sustainable software engineering has focused on reducing the carbon footprint of non-real-time workloads, such as batch processing or model training, through temporal or spatial shifting, these strategies are often unsuitable for latency-sensitive, interactive LLM inference tasks. These applications demand immediate responses to maintain Quality of Service (QoS), yet each inference request contributes to the software's overall energy con-

doi:10.6342/NTU202504158

sumption.To address this challenge, semantic caching has emerged as a promising technique. As demonstrated by projects like GPTCache, placing a semantic cache between the user and the LLM can effectively reduce the frequency of expensive LLM API calls, thereby lowering operational costs and improving response times. However, the direct impact of this technology on system-level energy consumption has not been sufficiently quantified or evaluated. The effectiveness of a semantic cache is critically dependent on its configuration, particularly the similarity threshold, which governs its hit-or-miss behavior. An improperly tuned cache could introduce computational overhead that negates any potential energy savings or even increases total consumption.

Therefore, to address this research gap, this study aims to systematically investigate the energy-performance trade-off inherent in semantic caching. The primary objective is to quantitatively measure how tuning the `similarity_threshold` influences the total energy consumption, while analyzing the corresponding effects on cache hit rate and response time. This involves establishing an empirical energy consumption baseline for a standard, non-cached deployment, which serves as a crucial benchmark to accurately quantify the net energy savings or additional costs introduced by the caching layer under different configurations.

To accomplish these objectives, this research will undertake a series of well-defined research tasks:

1. Design and construct a cloud-native experimental platform on Kubernetes, integrating an Ollama inference service with the GPTCache semantic caching layer.

2. Implement a comprehensive energy measurement framework by deploying Kepler and NVIDIA DCGM (Data Center GPU Manager) exporters to capture CPU, DRAM,

doi:10.6342/NTU202504158

and GPU power consumption at the pod level. Prometheus and Grafana are also deployed to store monitoring data and visualize it.

3. Generate a realistic and repeatable workload using the Quora Question Pairs (QQP) dataset and the k6 load testing tool to simulate cache hit and miss scenarios.

4. Conduct systematic experiments by manipulating the `similarity_threshold` as the key independent variable and comparing the results against a non-cached baseline.

5. Collect and analyze the resulting data on cache hit rate, response time, and power consumption to quantify the trade-offs and identify the conditions under which the cache provides a net energy benefit or becomes a parasitic overhead.

3.2 System and Experiment Design

To empirically investigate the research questions, a comprehensive experiment was designed, encompassing a specific system architecture, a representative dataset for workload generation, and a robust framework for energy measurement. All infrastructure components were defined using Terraform to ensure the experiment's replicability.

3.2.1 System Architecture

The experiment was conducted on the Google Kubernetes Engine (GKE) [24], a fully managed Kubernetes service provided by Google Cloud. While Kubernetes is open-source and can be installed and managed by anyone, it can be complex to set up and maintain a production-ready Kubernetes cluster. GKE simplifies the process and lets us utilize the

doi:10.6342/NTU202504158

functionalities of Kubernetes without having to build and maintain all by ourselves, which is suitable for our research. On GKE, a single-node cluster was provisioned, equipped with an NVIDIA T4 GPU to handle the computational demands of LLM inference. The core of the application stack consists of three main components:

- LLM Inference Service: Ollama was used to serve a 4-bit quantized version of the Mistral-7B model, providing a powerful yet resource-efficient open-source LLM for inference tasks.
- Semantic Cache Layer: GPTCache was integrated as the semantic caching layer within FastAPI framework to provide publicly accessible HTTP service.
- Vector and Scalar Database: pgvector was adopted as vector store for similarity searches in this system due to its compatibility with GPTCache. Also, PostgreSQL was utilized to store scalar data that returned to users once cache is hit.

The selection of the above tools primarily addresses the initial requirements for building LLM inference services on cloud-native platforms, alongside considerations for experimental costs. Two primary experimental groups were established to facilitate a comparative analysis, and Figures 3.1 and 3.2 illustrate the infrastructure configurations for the two distinct experimental groups, respectively:

- **Baseline** Group: This configuration represents a standard LLM deployment **without a caching layer**. All user prompts are sent directly to the inference service.
- **Benchmark** Group: This configuration includes the GPTCache semantic cache. **All incoming prompts are first processed by GPTCache**. Only in the event of a cache miss is the request forwarded to Ollama.

doi:10.6342/NTU202504158

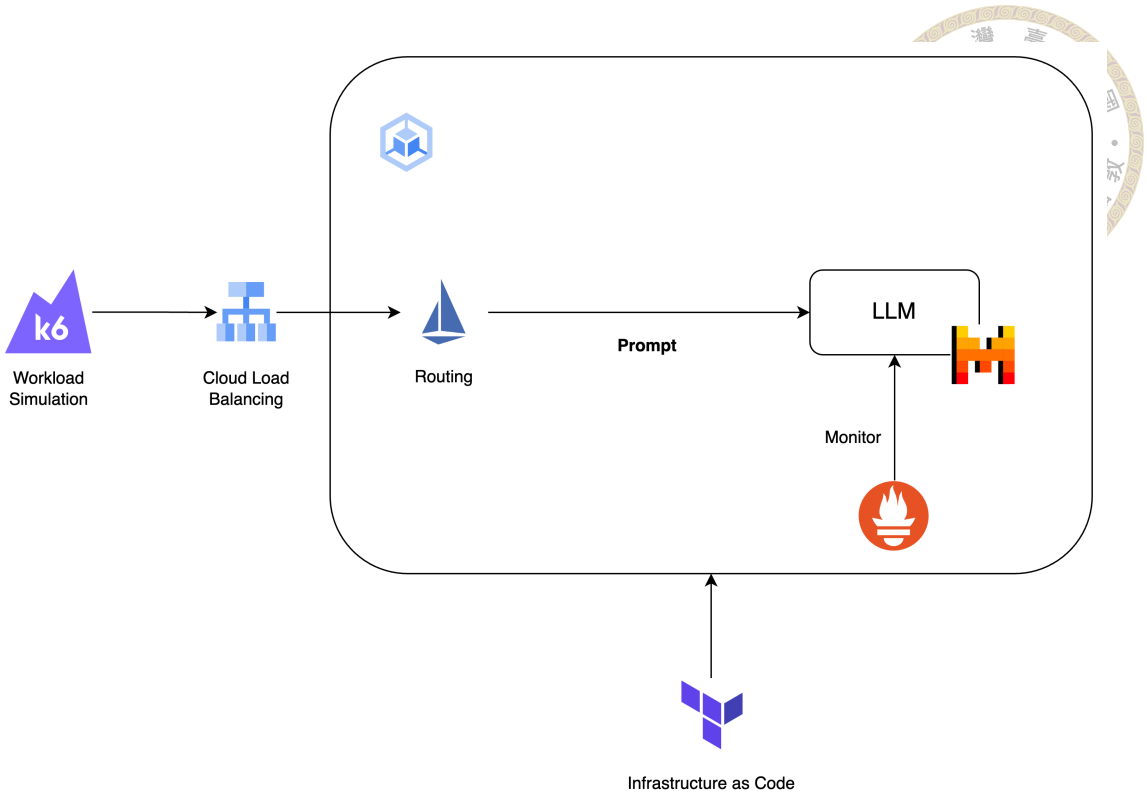


Figure 3.1: Infrastructures Setup of Baseline Group

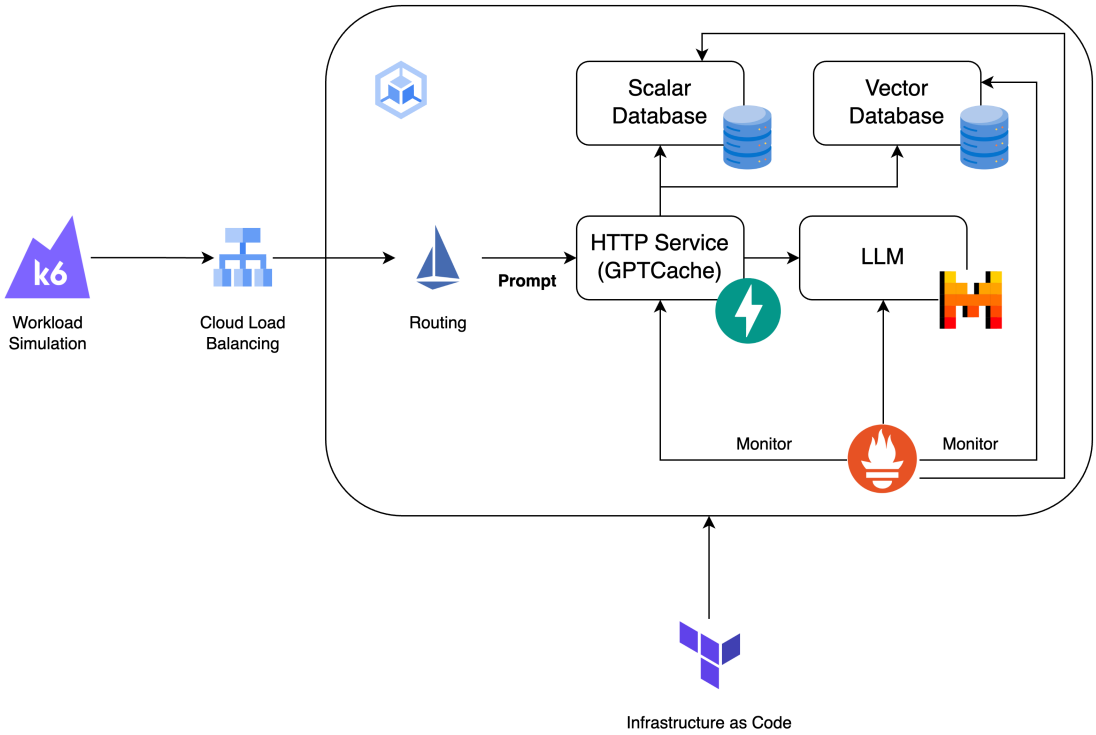


Figure 3.2: Infrastructures Setup of Benchmark Group

doi:10.6342/NTU202504158

3.2.2 Dataset and Workload Generation

To ensure the experiment's results are consistent and comparable with existing research, this study utilizes the QQP dataset. The QQP dataset has been human-annotated to indicate whether the two questions in a pair are semantically duplicate (is_duplicate = 1) or not (is_duplicate = 0). This structure perfectly models the operating conditions of a semantic cache, allowing for the scientific simulation of cache hit and cache miss scenarios. In this research, 30,141 questions in a pair are selected, and all of them are labeled as semantically duplicate, as Figure 3.3 shows.

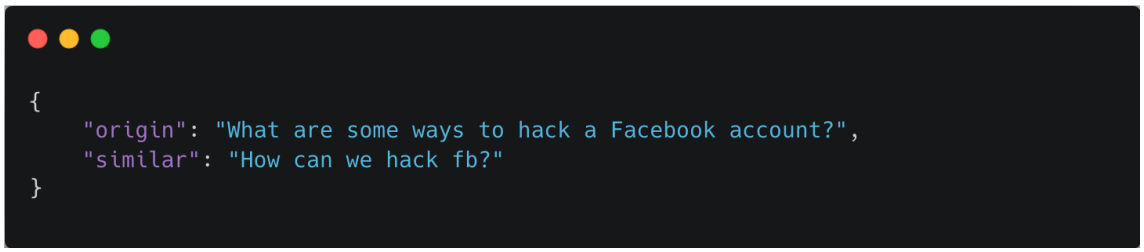


Figure 3.3: Example in QQP Dataset

Workload generation was managed by k6, a modern, scriptable load-testing tool. To simulate realistic user traffic, the k6 scripts were designed to gradually increase the number of virtual users (VUs) over a 10-minute experimental run. The maximum number of concurrent VUs was predetermined through a series of preliminary stress tests to identify the system's stable load capacity without causing out-of-memory (OOM) errors or excessive latency, a necessary step to ensure that the experiment measures a system under pressure but not one that is failing.

doi:10.6342/NTU202504158

3.2.3 Energy Measurement Framework

A multi-tool approach was required to achieve comprehensive and granular energy measurement within the virtualized GKE environment.

- Kepler was deployed to estimate pod-level energy consumption. As public cloud VMs do not expose low-level hardware interfaces like Running Average Power Limit (RAPL), Kepler utilizes eBPF to collect kernel-level statistics and applies machine learning models to estimate the power draw of CPU and DRAM for each pod.

- NVIDIA DCGM Exporter was deployed to specifically measure the power consumption of the NVIDIA T4 GPU. Kepler has technical limitations in directly measuring GPU power, making the DCGM exporter an essential component for capturing the energy consumed by the most power-intensive part of the LLM inference workload.

- Prometheus was used as the central time-series database to scrape and store all metrics exposed by Kepler and the DCGM exporter.

3.3 Experiment Variable

The primary independent variable investigated in this study is the `similarity_threshold` within the GPTCache configuration. This parameter is a floating-point value that dictates the minimum semantic similarity score required between an incoming user query and a cached query for a cache hit to occur.

doi:10.6342/NTU202504158

This variable was chosen because it represents the core trade-off in a semantic caching system . A low threshold (e.g., 0.7) is more lenient, increasing the probability of a cache hit. This has the potential to maximize energy savings by avoiding GPU-intensive inference but carries the risk of returning a semantically different, or ”false positive,” answer, potentially degrading response quality. Conversely, a high threshold (e.g., 0.95) is stricter, ensuring that only highly similar queries result in a hit, thus preserving response accuracy. However, this may significantly lower the cache hit rate, diminishing the cache’s overall benefit and potentially transforming it into a parasitic overhead that increases total energy consumption.

The range of values for this experiment—0.7, 0.75, 0.8, 0.85, 0.9, and 0.95—was selected to systematically analyze this trade-off . The starting point of 0.7 was informed by the original GPTCache paper, which identified it as a balanced value in their experiments. The subsequent values were chosen to provide a linear progression from a lenient to a very strict setting, enabling a detailed analysis of the parameter’s non-linear impact on system energy consumption and the identification of a potential ”benefit inversion point” .

3.4 Experiments and Measurement

To ensure the scientific rigor of the findings, a strict and repeatable experimental protocol was followed.

3.4.1 Procedure

Each experimental configuration—including the baseline and each `similarity_threshold` value in the benchmark group—was executed five times to account for performance vari-

doi:10.6342/NTU202504158

ability and ensure statistical significance . To maintain the independence of each trial, all Kubernetes resources (deployments, services, etc.) were systematically destroyed and re-deployed after every run. This ”clean slate” approach prevents confounding factors such as cache state persistence or resource fragmentation from influencing subsequent results.

3.4.2 Data Collection and Measurement

The final energy consumption data was calculated with precision to ensure accuracy. The following process was used for each experimental run:

- Precise Timing: The exact start and end timestamps for each 10-minute load test were programmatically recorded in k6.
- Average Power Calculation: After each run, Prometheus range queries using the `avg_over_time()` function were executed . This provided the average power consumption in Watts (W) for the CPU/DRAM (from Kepler) and the GPU (from DCGM) over the precise duration of the experiment.
- Total Power Aggregation: The total average power of the system was calculated by summing the average power values of all relevant components (the FastAPI/GPTCache pod, the Ollama pod, the pgvector/PostgreSQL pod).

This method of converting average power over a known duration into total energy provides the ultimate metric for comparing the overall energy cost of each experimental configuration . The final results were visualized using Grafana dashboards for initial analysis and later exported for statistical testing.

doi:10.6342/NTU202504158

Chapter 4 Results and Discussion

This chapter presents the empirical results of the experiments conducted to evaluate the energy consumption and performance characteristics of an LLM inference service integrated with a semantic cache. The data, collected following the methodology described in Chapter 3, is systematically analyzed to reveal the impact of the `similarity_threshold` on system behavior. The chapter begins by detailing the measured power and latency metrics, followed by an in-depth discussion that interprets these findings across different threshold ranges, and concludes with a summary of the key outcomes.

4.1 Results

The experiments yielded distinct and quantifiable differences in power consumption and response time between the Baseline group (no cache) and the Benchmark group (with GPTCache enabled at various `similarity_threshold` settings). The key performance indicators are summarized in Table 4.1 and Figure 4.1, with all data representing the average of five independent runs, as specified in the experimental protocol.

The baseline configuration, which sends all requests directly to the Ollama inference service , established a total average power consumption of 155.877 W and an average response time of approximately 30,294 ms. Over the 10-minute duration of the experi-

doi:10.6342/NTU202504158

Table 4.1: Relation between Cache Hit Rate and Power Consumption

Metric	Similarity Threshold						Baseline
	0.7	0.75	0.8	0.85	0.9	0.95	No Cache
Cache Hit Rate	99.99%	99.99%	99.30%	87.30%	54.90%	29.40%	N/A
Avg. Response Time	105.395	108.165	177.12	4634.66	19638.797	29035.722	30294.104
Avg. Power (excluding GPU)	37.893	35.211	51.748	76.729	75.537	74.969	72.21
GPU Avg. Power	50.8	52.267	70.733	98.5	98.967	97.433	83.667
Avg. Power (W)	88.78	87.478	122.481	175.217	174.504	172.402	155.877
Ttl. Energy Consumption (J)	53269.8	52486.8	73488.6	105130.2	104702.4	103441.2	93256.2

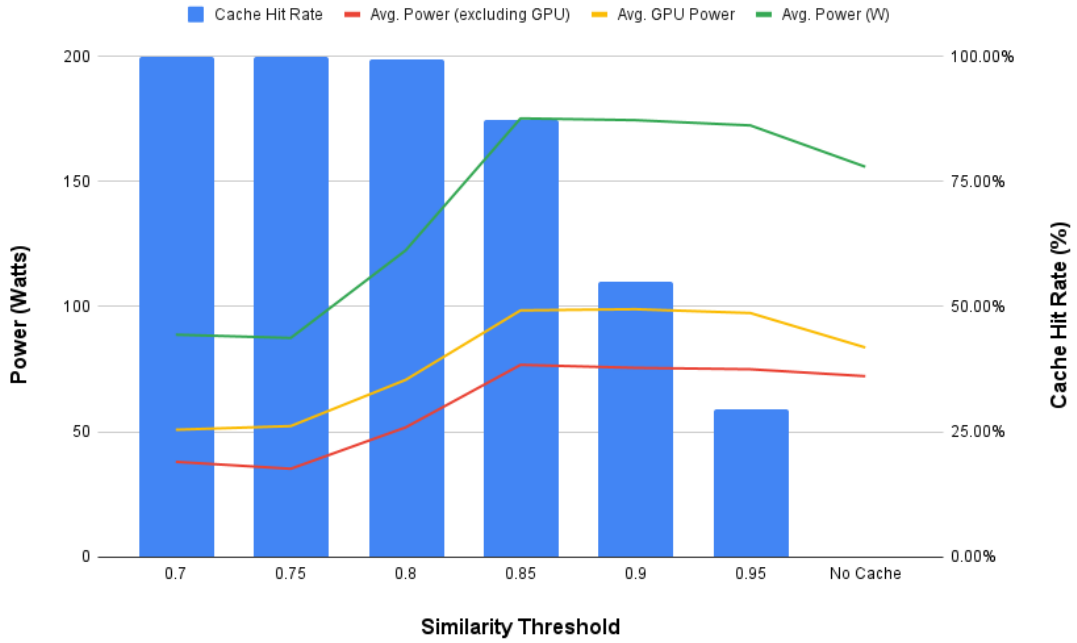


Figure 4.1: Metrics for Various Similarity Thresholds

ment, this resulted in a total energy consumption of 93,256.2 J, which serves as the critical benchmark for our energy efficiency evaluation.

In contrast, the benchmark configurations demonstrate a clear, non-linear relationship between the similarity threshold and the system's power and performance profile. At lenient thresholds (0.7, 0.75), the total power consumption was dramatically reduced to 88.78 W and 87.478 W, respectively—a reduction of over 43% compared to the baseline. This directly translates to a significant reduction in total energy consumption, which fell to as low as 52,486.8 J—a saving of over 40,000 J compared to the non-cached system.

doi:10.6342/NTU202504158

As the threshold was made stricter, total power consumption increased, eventually surpassing the baseline at a threshold of 0.85 (175.217 W) and remaining higher for all subsequent stricter settings. A similar trend was observed in average response times. The lenient thresholds of 0.7 and 0.75 yielded extremely fast responses (approx. 105-108 ms). However, as the threshold increased, response times grew exponentially, approaching the baseline's latency at the strictest setting of 0.95. These results indicate the existence of a "benefit inversion point," where the semantic cache transitions from an energy-saving component to a parasitic overhead. Crucially, the total energy consumed at thresholds of 0.85 and above exceeded 100,000 J, quantitatively proving that an improperly tuned cache consumes more total energy than having no cache at all.

4.2 Discussion

The observed results can be explained by the fundamental trade-off between cache hit rate and computational overhead, which is governed by the `similarity_threshold`. The following sections analyze the system's behavior at different threshold intervals.

4.2.1 Analysis at Low Similarity Thresholds (0.7, 0.75, 0.8)

At the lenient thresholds of 0.7 and 0.75, the system achieved near-perfect cache hit rates of 99.99%. This is attributed to the lenient matching criteria combined with the nature of the Quora Question Pairs dataset, which is designed to contain many semantically similar pairs. With such a high hit rate, the vast majority of incoming requests were handled by GPTCache and its pgvector backend, almost completely avoiding calls to the GPU-intensive Ollama service.

doi:10.6342/NTU202504158

This is directly reflected in the power data. The average GPU power was minimal (around 50-52 W), while the primary power draw came from the CPU-bound processes of the API Gateway and the semantic cache itself. The result was a total system power consumption (approx. **87-88 W**) that was significantly lower than the baseline's **155.877 W**. This demonstrates the ideal operating scenario for a semantic cache: the modest energy cost of the cache lookup is far outweighed by the massive energy savings from avoiding LLM inference. Consequently, the total energy consumed during the entire test run was the lowest in these configurations, bottoming out at 52,486.8 J. This figure represents a tangible energy saving of 43.7% compared to the baseline, providing strong evidence for the cache's effectiveness when properly configured for a given workload.

At a threshold of 0.8, the cache hit rate decreased slightly to 99.30%. While still very high, this minor drop meant that a larger fraction of requests resulted in a cache miss and were forwarded to Ollama. Consequently, GPU power consumption rose to 70.733 W, pushing the total average power up to **122.481 W**. This marks the beginning of a clear trend: even a small decrease in cache hit rate leads to a disproportionate increase in total energy consumption due to the high energy cost of GPU activation. Specifically, the total energy consumption rose to 73,488.6 J; although still more efficient than the baseline, the system consumed over 21,000 J more than it did at the 0.75 threshold, highlighting the system's sensitivity to even minor drops in cache hit rate.

4.2.2 Analysis at an Intermediate Threshold (0.85, 0.9)

The experiment revealed a critical "benefit inversion point" at a similarity threshold of 0.85. At this setting, the cache hit rate dropped to 87.30%. While still substantial, this meant that over 12% of requests resulted in a cache miss. For each miss, the system

doi:10.6342/NTU202504158

incurred a double performance fallback:

1. **Cache Search Overhead**: The CPU and DRAM resources were consumed by GPT-Cache to perform the initial (failed) similarity search.

2. **Inference Cost**: The request was then forwarded to Ollama, consuming the full amount of GPU power and time for inference.

This compounding effect caused the total average power to surge to **175.217 W**, exceeding the baseline's 155.877 W. At this point, the cache was no longer an energy-saving mechanism but a parasitic overhead. It not only failed to prevent a significant portion of expensive GPU work but also added its own computational load to the system. This is further evidenced by the dramatic increase in average response time to 4,634 ms, reflecting the combined latency of a cache miss and a full LLM inference. From a total energy perspective, this configuration is the most wasteful, consuming 105,130.2 J. This is a crucial finding: the cache caused the system to consume nearly 12,000 J more energy than the baseline system without a cache. This quantifies the "parasitic overhead" and proves that deploying a cache without careful tuning can be actively detrimental to energy efficiency. The trend continued at the 0.9 threshold, where a 54.90% hit rate was insufficient to offset the energy cost of frequent misses, resulting in a similarly high total power draw of 174.504 W and a total energy consumption of 104,702.4 J.

4.2.3 Analysis at a Strict Threshold (0.95)

At the strictest threshold of 0.95, the cache's effectiveness was severely diminished, with a hit rate of only 29.40%. The system's behavior closely resembled the baseline, as

doi:10.6342/NTU202504158

nearly three out of every four requests were forwarded to the Ollama service for processing. This is clearly reflected in the average response time of 29,035 ms, which is nearly identical to the baseline's 30,294 ms.

The total average power, while slightly lower than at the 0.85/0.9 thresholds, remained high at **172.402 W**, still significantly above the baseline. This demonstrates that even when a cache is largely bypassed, its mere presence as an intermediary layer contributes a persistent energy overhead. For the majority of requests, the system paid the full price of GPU inference plus the additional, non-trivial energy cost of a highly selective, and therefore frequently unsuccessful, cache lookup. This confirms that a poorly tuned cache, particularly one with an overly strict threshold for a given workload, is detrimental to both energy efficiency and performance. The total energy data reinforces this conclusion: the system consumed 103,441.2 J, which is over 10,000 J more than the baseline. This proves that for the duration of the workload, the energy penalty of the frequent, failed cache lookups accumulates into a substantial and wasteful overhead.

4.3 Summary

The findings from this study lead to several important conclusions. First, the relationship between a semantic cache's similarity threshold and its energy efficiency is decidedly non-linear. Semantic caching is not a universally beneficial solution for energy reduction. Instead, its effectiveness is entirely conditional on careful parameter tuning.

The core principle revealed by this research is that the energy benefit of a cache is fundamentally derived from its ability to successfully offload computations from a more energy-intensive downstream component—in this case, shifting work from the GPU to

doi:10.6342/NTU202504158

the CPU. When the cache hit rate is high, this workload shift is highly effective. This is demonstrated by the system's total energy consumption being reduced by over 43%, from 93,256.2 J to 52,486.8 J. However, when the hit rate falls, the energy cost of the cache's own operations becomes a significant factor, and the system begins to suffer from the added overhead rather than benefiting from it. This is quantitatively proven at the " benefit inversion point" (threshold 0.85), where the total energy consumption peaked at 105,130.2 J, an increase of over 12% compared to the baseline.

This carries a critical implication: the similarity threshold must be treated as a key hyperparameter that requires diligent tuning and monitoring. Deploying a semantic cache with default or untested settings risks creating a system that is less performant and consumes more energy than a simpler, non-cached architecture. The optimal threshold will likely vary based on the specific LLM, the nature of the user queries, and the desired balance between response accuracy and energy efficiency. This underscores the need for robust monitoring of cache hit rates in production environments to ensure the caching layer is consistently providing a net positive impact.

doi:10.6342/NTU202504158

Chapter 5 Conclusion

The rapid adoption of Large Language Models (LLMs) has introduced a new class of energy-intensive, latency-sensitive workloads into the cloud-native ecosystem. This thesis set out to investigate semantic caching as a potential strategy to mitigate the high energy consumption of LLM inference services. By conducting a series of empirical experiments on a Kubernetes-based platform, this study has demonstrated that while semantic caching can yield substantial energy savings and performance improvements, its effectiveness is critically dependent on the configuration of its `similarity_threshold`.

5.1 Key Findings

This research systematically quantified the non-linear relationship between the semantic cache's similarity threshold and the overall system efficiency. The core of the study was to measure how adjusting this single parameter impacts the trade-off between energy consumption and service performance. The key empirical findings are as follows:

1. The experimental data confirms that when the similarity threshold is set to a lenient value (e.g., 0.7), the system achieves a cache hit rate of up to 99.99%. This high hit rate successfully offloads the vast majority of computational requests from the energy-intensive GPU to the CPU. As a result, the total system power consump-

doi:10.6342/NTU202504158

tion was reduced from 155.877 W to 88.78 W, which is by over 43% compared to the non-cached baseline, while response times were also drastically improved from 30294.104 ms to 105.395 ms. More concretely, this corresponds to a drop in total energy consumption from 93,256.2 J in the baseline to 52,486.8 J, saving over 40,000 J during the experimental run. This demonstrates the ideal operating scenario where semantic caching acts as a highly effective energy-saving component.

2. This study also pinpoints the critical risk of improper cache tuning. As the similarity threshold becomes overly strict (e.g., 0.85 and above), the cache hit rate plummets. In this scenario, the system incurs both the overhead of a failed cache lookup and the full cost of GPU inference. This transforms the cache layer from an asset into a parasitic liability, causing the total system power consumption to increase by more than 12% above the baseline, from 155.877 W to 175.217 W. From a total energy perspective, this means the system consumed 105,130.2 J, which is approximately 12,000 J more than the baseline. The cache didn't just fail to save energy; it actively caused the system to waste it.

5.2 Research Contribution

This thesis makes several key contributions to the field of sustainable software engineering and LLM application architecture:

1. **First Quantitative Energy Profile of Semantic Caching in a Cloud-Native Environment**: This study is the first to quantitatively measure and document not just the instantaneous power, but the total energy consumption (in Joules) of an LLM semantic cache system within a representative cloud-native architecture (Kuber-

doi:10.6342/NTU202504158

netes with GPU acceleration). By providing a detailed, empirical baseline of power consumption for both cache hits and cache misses, this work establishes a foundational energy model. This model is a prerequisite for developing more sophisticated, carbon-aware workload schedulers that can make intelligent decisions based on the energy cost of different operations.

2. **Empirical Evidence of Caching's Limitations and the "Benefit Inversion Point"**: The research empirically proves that semantic caching is not inherently energy-efficient and has clear limitations. It demonstrates that when the similarity threshold is set too high, the system suffers from a "double penalty" where the low cache hit rate forces it to incur the energy cost of both the cache lookup and the full LLM inference. This results in a total system power draw that is higher than the baseline, effectively making the cache detrimental to energy efficiency. This finding provides concrete evidence of a "benefit inversion point," quantifying it not just as a power increase but as a net total energy loss of over 12% (an additional 12,000 J consumed). This serves as a critical insight for practitioners, highlighting that improper configuration can negate and even reverse the intended benefits of a caching layer.

5.3 Future Work

Building upon the findings of this research, several promising avenues for future work have been identified:

doi:10.6342/NTU202504158

- Comprehensive Parameter and Quality Analysis: The current study focused on the similarity threshold. A logical next step is to investigate other critical parameters, such as the cache eviction policy (e.g., LRU, LFU), to understand its impact on the trade-off between memory consumption and hit rate. Furthermore, to address the risk of ”false positives” at lenient thresholds, future work should incorporate NLP evaluation metrics like BLEU and ROUGE to measure accuracy of the cached responses, providing a more holistic view of the energy-accuracy trade-off.

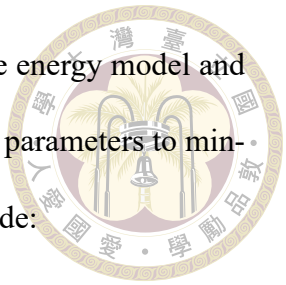
Another significant future project would be to develop a custom Kubernetes Operator that moves beyond passive energy measurement to active carbon optimization. The energy consumption model established in this thesis provides the foundational data needed for such a system to make intelligent, real-time decisions. The ultimate goal is to create an operator that automates the management of LLM workloads to minimize their total carbon footprint.

This ”Carbon-Aware Operator” would integrate several advanced strategies discussed in green software engineering literature and would be designed to:

- **Integrate the Energy Consumption Model:** The operator’s core logic would be built upon the energy-performance trade-off model quantified in this thesis, allowing it to accurately predict the power consumption of different operational states (e.g., cache hit vs. cache miss).

- **Consume Real-time Carbon Intensity Data:** It would connect to real-time APIs (e.g., from providers like Watt Time) to fetch current or forecasted marginal carbon intensity data for the electricity grid.

doi:10.6342/NTU202504158



- **Execute Dynamic, Multi-faceted Optimization:** Based on the energy model and live carbon data, the operator would dynamically adjust system parameters to minimize total carbon emissions (gCO_2eq). Its actions would include:
  - **Temporal Shifting via Elasticity:** Dynamically adjusting pod replica counts, scaling up during low-carbon periods and scaling down when the grid is " dirtier," a concept demonstrated by systems like CarbonScaler [25]. This thesis provides the data to avoid the performance pitfalls highlighted in previous studies [26].
  - **Spatial Shifting via Traffic Routing:** For multi-cluster deployments, using a service mesh (e.g., Istio) to intelligently route inference requests to data centers in regions with the lowest real-time carbon intensity, operationalizing the principles of schedulers like the Low Carbon Kubernetes Scheduler [**?**].
  - **Application-level Adaptation:** Modifying application behavior by, for example, dynamically tuning the semantic cache's `similarity_threshold`. During high-carbon periods, the operator could lower the threshold to maximize the cache hit rate and reduce GPU activation, a form of graceful degradation inspired by systems like BrownoutCon [27].

The development of such an operator represents the logical evolution of this thesis. It transforms the empirical findings from a static analysis into a dynamic, automated control system for sustainable AI operations, bridging the gap between foundational energy measurement and practical carbon reduction.

doi:10.6342/NTU202504158

References

[1] Charlotte Freitag, Mike Berners-Lee, Kelly Widdicks, Bran Knowles, Gordon S. Blair, and Adrian Friday. The real climate and transformative impact of ict: A critique of estimates, trends, and regulations. Patterns, 2(9):100340, 2021.

[2] Sasha Luccioni, Yacine Jernite, and Emma Strubell. Power hungry processing: Watts driving the cost of ai deployment? In Proceedings of the 2024 ACM conference on fairness, accountability, and transparency, pages 85–99, 2024.

[3] Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga, Jinshi Huang, Charles Bai, et al. Sustainable ai: Environmental implications, challenges and opportunities. Proceedings of machine learning and systems, 4:795–813, 2022.

[4] David Patterson, Joseph Gonzalez, Urs Hölzle, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David R So, Maud Texier, and Jeff Dean. The carbon footprint of machine learning training will plateau, then shrink. Computer, 55(7):18–28, 2022.

[5] Aled James and Daniel Schien. A low carbon kubernetes scheduler. In ICT4S, 2019.

[6] Andrew A Chien, Liuzixuan Lin, Hai Nguyen, Varsha Rao, Tristan Sharma, and Rajini Wijayawardana. Reducing the Carbon Impact of Generative AI Inference

doi:10.6342/NTU202504158

(today and in 2035). In Proceedings of the 2nd Workshop on Sustainable Computer Systems, pages 1–7, Boston MA USA, July 2023. ACM.

[7] Baolin Li, Yankai Jiang, Vijay Gadepally, and Devesh Tiwari. Toward Sustainable GenAI using Generation Directives for Carbon-Friendly Large Language Model Inference, March 2024. arXiv:2403.12900 [cs].

[8] Fu Bang. GPTCache: An Open-Source Semantic Cache for LLM Applications Enabling Faster Answers and Cost Savings. In Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023), Singapore, Singapore, 2023. Empirical Methods in Natural Language Processing.

[9] Green Software Practitioner. https://learn.greensoftware.foundation/introduction/, 2024.

[10] Green Software Patterns. https://patterns.greensoftware.foundation/, 2024.

[11] Yao Lu, Song Bian, Lequn Chen, Yongjun He, Yulong Hui, Matthew Lentz, Beibin Li, Fei Liu, Jialin Li, Qi Liu, Rui Liu, Xiaoxuan Liu, Lin Ma, Kexin Rong, Jianguo Wang, Yingjun Wu, Yongji Wu, Huanchen Zhang, Minjia Zhang, Qizhen Zhang, Tianyi Zhou, and Danyang Zhuo. Computing in the Era of Large Generative Models: From Cloud-Native to AI-Native, January 2024. arXiv:2401.12230 [cs].

[12] Erik De Castro Lopo. Cloud-Native AI: Building and Deploying Large Language Models with Scalable Cloud Architectures. 1(8), 2024.

[13] Suyash Joshi, Basit Hasan, and R Brindha. Optimal Declarative Orchestration of Full Lifecycle of Machine Learning Models for Cloud Native. In 2024 3rd International

doi:10.6342/NTU202504158

Conference on Applied Artificial Intelligence and Computing (ICAAIC), pages 578–582, Salem, India, June 2024. IEEE.

[14] Salman Taherizadeh and Marko Grobelnik. Key influencing factors of the Kubernetes auto-scaler for computing-intensive microservice-native cloud-based applications. Advances in Engineering Software, 140:102734, February 2020. Publisher: Elsevier BV.

[15] Jose Santos, Tim Wauters, Bruno Volckaert, and Filip De Turck. Towards Network-Aware Resource Provisioning in Kubernetes for Fog Computing Applications. In 2019 IEEE Conference on Network Softwarization (NetSoft), pages 351–359, Paris, France, June 2019. IEEE.

[16] Luping Wang, Lingyun Yang, Yinghao Yu, Wei Wang, Bo Li, Xianchao Sun, Jian He, and Liping Zhang. Morphling: Fast, Near-Optimal Auto-Configuration for Cloud-Native Model Serving. In Proceedings of the ACM Symposium on Cloud Computing, pages 639–653, Seattle WA USA, November 2021. ACM.

[17] Dongwon Lee and Wesley W Chu. Semantic caching via query matching for web sources. In Proceedings of the eighth international conference on Information and knowledge management, pages 77–85, 1999.

[18] Jiaxing Li, Chi Xu, Feng Wang, Isaac M. von Riedemann, Cong Zhang, and Jiangchuan Liu. SCALM: Towards Semantic Caching for Automated Chat Services with Large Language Models, May 2024. arXiv:2406.00025 [cs].

[19] Ramaswami Mohandoss. Context-based Semantic Caching for LLM Applications. In 2024 IEEE Conference on Artificial Intelligence (CAI), pages 371–376, Singapore, Singapore, June 2024. IEEE.

doi:10.6342/NTU202504158

[20] Banghua Zhu, Ying Sheng, Lianmin Zheng, Clark Barrett, Michael I. Jordan, and Jiantao Jiao. On Optimal Caching and Model Multiplexing for Large Model Inference, August 2023. arXiv:2306.02003 [cs].

[21] Carlo Centofanti, José Santos, Venkateswarlu Gudepu, and Koteswararao Kondepu. Impact of power consumption in containerized clouds: A comprehensive analysis of open-source power measurement tools. Computer Networks, 245:110371, May 2024. Publisher: Elsevier BV.

[22] Mathilde Jay, Vladimir Ostapenco, Laurent Lefevre, Denis Trystram, Anne-Cécile Orgerie, and Benjamin Fichel. An experimental comparison of software-based power meters: focus on CPU and GPU. In 2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid), pages 106–118, Bangalore, India, May 2023. IEEE.

[23] Marcelo Amaral, Huamin Chen, Tatsuhiro Chiba, Rina Nakazawa, Sunyanan Choochotkaew, Eun Kyung Lee, and Tamar Eilam. Kepler: A Framework to Calculate the Energy Consumption of Containerized Applications. In 2023 IEEE 16th International Conference on Cloud Computing (CLOUD), Chicago, IL, USA, July 2023. IEEE.

[24] Google Kubernetes Engine. https://cloud.google.com/kubernetes-engine, 2025.

[25] Walid A. Hanafy, Qianlin Liang, Noman Bashir, David Irwin, and Prashant Shenoy. CarbonScaler: Leveraging Cloud Workload Elasticity for Optimizing Carbon-Efficiency. Proceedings of the ACM on Measurement and Analysis of Computing

doi:10.6342/NTU202504158

Systems, 7(3):1–28, December 2023. Publisher: Association for Computing Machinery (ACM).

[26] Haben Birhane Gebreweld. Evaluating the energy consumption impact of a carbon aware autoscaling in microservice-based applications on the public cloud: A sustainability perspective. 2023.

[27] Minxian Xu and Rajkumar Buyya. BrownoutCon: A software system based on brownout and containers for energy-efficient cloud computing. Journal of Systems and Software, 155:91–103, September 2019. Publisher: Elsevier BV.

doi:10.6342/NTU202504158