Vrije Universiteit Amsterdam

Bachelor Thesis

Kavier: Exploring Performance, Sustainability, and Efficiency of LLM Ecosystems under Inference through Cache-Aware Discrete-Event Simulation

Author: Radu Nicolae (2760443)
r.nicolae@vu.nl

1st supervisor:	Prof. Dr. Ir. Cav. Alexandru Iosup	(VU Amsterdam)
Daily supervisor:	Dr. Animesh Trivedi	(IBM Research Europe)
2nd reader:	Dr. Ir. Jesse Donkervliet	(VU Amsterdam)

A thesis submitted in fulfillment of the requirements for the VU Bachelor of Science degree in Computer Science.

July 15, 2025

Many thanks to AtLarge team, our partners, and people involved in this research.

IBM Research
Europe/Zurich

Abstract

Large Language Models (LLMs) are widely used by our increasingly digitalized society, but raise sustainability, performance, and financial concerns, especially as inference workloads grow. To improve the design and operation of LLM ecosystems, we envision simulators and simulation-based digital twins becoming primary decision-making tools. LLM ecosystems leverage many heterogeneous components, making simulation a non-trivial, yet critical operation. The simulation challenge is exacerbated by the absence of a comprehensive reference architecture of LLM ecosystems; the lack of such a conceptual model can be costly and could misguide the designers and engineers. Without a reference architecture, even the most experienced stakeholders could tinker in researching, engineering, or maintaining LLM ecosystems. In this work, we bring a three-fold contribution to the scientific community. Firstly, we synthesize, propose, and validate a reference architecture (RA) of LLM ecosystems under inference. Then, adhering to the reference architecture, we design Kavier, the first simulation instrument able to predict the performance, sustainability, and efficiency of LLM ecosystems under inference, through discrete-event and cache-aware simulation, focusing on Key-Value-(KV-)Caching and prompt prefix caching policies.

Through experiments with a Kavier prototype and real-world traces, (i) we measure the accuracy of Kavier and its performance in massive-scale simulations, (ii) we compare the performance of different KV-Caching policies, and (iii) we analyze the performance, sustainability, and efficiency of LLM ecosystems under various prefix caching policies. Through experiment (i), we demonstrate that Kavier can simulate hundreds of GPU hours in a matter of seconds, at second granularity, and with error rates of less than 10%. Through experiments (ii) and (iii), we identify and quantify operational aspects of caching in the context of LLM inference. Specifically, in experiment (ii), we quantify improvements of 2-3 orders of magnitude on performance when LLM ecosystems adopt KV-Caching. In experiment (iii), we identify that prefix caching can reduce latency by up to 65%, with cascading improvements also in environmental and financial costs. Overall, we show that Kavier enables operators, researchers, and engineers to predict LLM ecosystems in a time, performance, and cost-efficient way.

Keywords

LLMs, LLM ecosystems, KV-Cache, discrete-event simulation, performance, sustainability, efficiency, energy utilization, OpenDC

Acknowledgments

"Massivizing Computer Systems, from the Metaverse to the Continuum" – this was my very first lecture at Vrije Universiteit Amsterdam, on September 6th, 2022. Back then, I already had a few years of experience in software engineering, and I was in love with computers and programming (turns out, years later, I still am!), but I had no idea what science was. I remember the excitement the lecturer had towards computer science and towards researching distributed computer ecosystems with impact on millions. That passion was quite contagious, and suddenly I was also excited about the massive-scale computer (eco)systems. I wanted to learn more. The conversation afterwards with the lecturer sparked a large interest in the field.

Almost three years later, I am taking the last steps of my journey towards my Bachelor's degree. During this time, I had unique research and teaching opportunities. Through my enrollment in the Honours Programme, I had the opportunity to lead an ambitious scientific research project on datacenter multi-model simulation through which I broke new ground. Still on the scientific side, I had the pleasure to give numerous invited talks, write a scientific article for peer-reviewed publication, and co-supervise students in their first scientific steps. On the teaching side, I had the opportunity to run large-enrollment courses, give lectures, and contribute to shaping the computer science and honours programme curricula of the VU. I was deeply honoured to be selected as the student of the year by the Faculty of Science and to be awarded the Student Talent Award by the Faculty. I owe much to all the people who facilitated all of these.

For the remainder of this (too) limited space, I would like to offer both nominal and non-nominal gratitude to all the people who have been with me through this exciting journey, either always or episodically.

Alexandru, *mulțumesc.* You have been my guide through science and taught me almost everything I know about "compsys," research methodology, and scientific reasoning. You believed in me from the very beginning and invited me to be part of *the* leading computer systems research group in the country. You've offered me high-tier, high-responsibility opportunities, and we've built some big things together. I'm looking forward to seeing how they'll concretize in the future (see DT :D). Thank you for the long and varied conversations, for mentoring me when I was unsure, and for the encouragements when I felt low (the *"capul sus"* always made the situation at least n% better). From *"fishing"* hours, 6 am, to late-evening meetings, you've always inspired me to learn more, challenge myself, and seek the highest level of depth. For all, I'm grateful.

Animesh, it's been a pleasure to work together and learn more about your research and teaching style. I am grateful for all the shared knowledge and insightful papers, and especially thankful for the constructive feedback and healthy harshness. Jesse, thank you for teaching me research methodology, computer networking, metaverse, and online gaming systems, and I'm deeply grateful for our conversations on both professional and personal topics. I also want to thank the entire AtLarge team for always being so helpful, approachable, and supportive; especially, many thanks (alphabetically), Daniele, Dante, Hexiang, Krijn, Matthijs, Sacheen, Tiziano, Vincent, and Xiaoyu.

Thank you to the CS Department for your support and opportunities. Ivano, I had a lot of fun running the Network Institute together; it was very insightful to explore the endless world of interdisciplinary science. Thank you, Sara, for your help in so many things - I'd probably double the size of this thesis if I were to mention all of them. Thilo, it's been a pleasure to run computer programming together - thank you for your trust and support. Thank you, Mojca, *"the mother of the CS department"*, and your always good energy during our sometimes too long conversations; Mojca, you never failed to put a smile on my face.

All these would not have been possible without my amazing friends and colleagues, who helped me disconnect from scientific fun when I needed a break. Thank you, Daniel, Cristi, Sofia, Ana, Lara, Maja, Clara, Cătălin, Traian (Finu), Lennart, Dovydas (Dovy), Sorin (Sorinel), Matei, Isidora, Nader, Ștefan (Fane), Alexis ... . A special thank you to Fadime – *sağ ol*, Fa. Thank you to my riding friends who constantly reminded me that two wheels and some miles are the best therapy: Rareș, Voshon, Fifi (wherever you'd be, bud), and Daniel.

I would like to deeply thank my family for all their support, of all kinds, and at all times. This goes beyond my Bachelor's and beyond my career – thank you for always being here and e-here. Thank you for supporting me in my passion and in moving to the other side of the continent to follow this passion. Regardless of the proximity, parts of my heart will always be in Bucharest, Victoria, and Hațeg. Lastly and firstly, thank you, D.S. and A.B.

Radu Nicolae

Contents

1 Introduction

LLM ecosystems are being adopted at an unprecedented scale [1, 2], and are hosted on massive-scale, intensely used ICT infrastructure, hence raising concerns about performance, sustainability, and efficiency [3, 4, 5, 6]. To understand ICT infrastructure, numerous simulators have been proposed in the past decades, such as OpenDC [7], DCSim [8], GDCSim [8], or CloudSim [9]. However, no simulator supports the prediction of LLM ecosystems under inference and cache-awarely. For LLM ecosystems, the caching system is crucial, especially Key-Value Caching (KV-Caching) and prompt prefix caching [10, 2, 11], and proven to have significant impacts on performance, sustainability, and efficiency, sometimes of orders of magnitude. Although simulators for predicting LLMs have been proposed (e.g., Vidur [12], LLMServingSim [13]), none of them can cache-awarely simulate the performance, sustainability, and efficiency of LLM ecosystems under inference. Exacerbating this challenge, the absence of a reference architecture of LLM ecosystems under inference prevents rigorously designing and implementing a scientific simulation instrument; adhering to state-of-the-art methodology [14], followed by top-tier publications [15, 16, 17, 7], simulators should be designed and implemented as mapped to validated reference architectures. Furthermore, the absence of a reference architecture can misguide even the advanced groups of engineers, researchers, and operators, which could overlook crucial aspects of the ecosystem (e.g., prompt prefix caching). Identifying the absence of a conceptual model and of a simulation instrument leads to the main research question: *(MRQ) How to enable analysis of LLM Ecosystems under inference through discrete-event simulation?* In this work, we address the MRQ and propose a dual main contribution, first, a detailed and comprehensive reference architecture to guide how LLM-inference systems are designed, deployed, and analyzed; second, the Kavier tool to simulate and analyze LLM-inference systems based on the reference architecture. Kavier facilitates the community to explore real-world LLM ecosystems in a time and cost efficiency way, through simulation-driven experimentation. Having a better understanding of massive-scale computer ecosystems, especially LLM ecosystems, can lead to significant improvements in these systems' performance, sustainability, and efficiency overall [1].

Our society and economy are increasingly dependent on AI services, especially on Large Language Models (LLMs) [1, 2, 4]. Correspondingly, LLMs are becoming more accessible to the public at large, while leveraging more complex architectures and consuming massive computational resources [3, 4]. Since the launch of GPT-3 in November 2022, LLMs have been and are being increasingly embedded in operational processes across industry, government, and academia [18, 19]. In industry, as of 2025[1], tech giants integrate LLMs into their search engines (e.g., Gemini, Copilot), and customer service platforms, processing billions of queries daily [19, 20]; in academia, LLMs are widely used in research, especially in field as computer science, medicine, chemistry, or biology, in processes of scientific writing, data analysis, or programming assistance [21, 22, 23, 24]; governments deploy LLMs for public service automation [25] and policy analysis [26, 27]. The wide use of these services is reflected in various costs – LLMs are trained and run at massive sustainability, financial, and performance costs [2, 1].

[1] This thesis has been written and submitted in 2025, including information and data which might change over time. Although apparently ephemeral, the presented numbers highlight the massive scale of LLM services and their societal impact, which is projected to grow in the upcoming decades.

The environmental footprint of LLMs is massive and is only expected to grow, further exacerbating the already concerning global challenges in resource allocation, energy consumption, and CO2 emissions [3]. This footprint begins as soon as the production of the hardware on which the LLM ecosystem is deployed [19]. For example, manufacturing a single NVIDIA H100 GPU, widely used in AI training and inference infrastructure, generates hundreds of kgCO2 (estimated between 200-500kg CO2 per GPU unit) [28, 3]; equipping a hyperscale datacenter with 10,000 NVIDIA H100 GPUs results in over 20,000-50,000 tCO2 emissions before even deploying the LLM ecosystem and without considering other components of such a datacenter [29, 20]. The footprint of the training process is significant even for GPT-3, a small LLM by 2025 standards, which consumed 1.28 GWh of energy, and emitted 553 metric tons of CO2 [30], equivalent to 123 gasoline cars driven for a year [31]. Larger LLMs, such as Google's PaLM-2 (340B parameters), require 3.4x more energy than GPT-3 during training, while models like Antropic's Claude 3 (500B parameters) consume over 5 GWh per training run [18]. Lastly, inference exacerbates exponentially this climate footprint - ChatGPT service, at peak usage, consumes over 1 GWh daily [19], on par with approximately 40,000 Dutch houses [32]. However, the CO2 intensity varies by the energy source: LLMs run on coal-based power grid can emit up to 2-3 orders of magnitude more CO2 than LLMs run on renewable or nuclear-powered datacenters [33, 34]. Cumulatively, in 2025, LLMs are estimated to account for approximately 3% of the electricity consumption of the global datacenters, and this proportion is expected to exponentially grow [35].

The financial costs of training and running LLM are increasing at an unprecedented pace, with an estimated growth of an order of magnitude per year in compute costs [5]. *Cottier et al.* note the magnitude growth in the regression mean for training frontier AI models, from approximately \$10k in 2016, to approximately \$8M (\$0.08B) in 2024, and expected to overtake \$1B by 2027 [5]. Inference costs are equally staggering [36, 37, 38]: ChatGPT costs \$700,000 per day to operate, and using GPT-4 to support customer service can cost a small business \$21,000 a month [37]. The financial costs can grow exponentially for million-scale token contexts (e.g., Google's Gemini 1.5), which increases GPU memory usage by 4-8x, and proportionally the cloud hosting fees [38]; some cloud providers estimate a single 100K-token prompt at \$0.50 scale in cloud compute fees, while a small prompt at only fractions of a cent [36, 38]. Still, it is essential to note that many factors could influence the price of LLM inference, and, although the number of tokens influences the price, there is no correlation between these two metrics; for example, the measured cost of running Jamba 1.5 Large (256k parameters) on Amazon Bedrock was of \$0.32 per 100k tokens of prompt (equivalent to \$0.81 per prompt), while the cost of running Clause 3.5 Sonnet (200k parameters) on the same infrastructure was of \$0.54 per 100k tokens of prompt (equivalent to \$1.08 per prompt) [36]. Scaled to an audience of tens/hundreds of millions of active, intense users, these costs become unsustainable.

The performance costs of LLMs are reflected into a "modern-day Moore's law" [6], where available infrastructure fails to meet the ultra-high demands of ultra-large LLMs. GPT-4 was trained in 5,000 - 10,000 GPU years[2]; the Ice Age terminated ≈11,700 (human) years ago, if we sequentialize the GPUs and assume no external factors, the training of GPT-4 would have started a few millennia before pyramids were built, just after the end of the Ice Age [39, 40, 5, 41, 42]. Albeit the massive scale of the training stage, the training becomes the smaller sibling of the inference, which increases proportionally with the exponentially growing number of users *and* size of models; Google estimates the ratio between training and inference as 40 to 60, thus showing how "modern"-day AI spends most of its lifetime in the inference stage [43, 44]. *Chien et al.* analyze ChatGPT and Google AI services and observe that annual inference needs 25x, respectively 1,386x more compute resources, than were needed to train GPT-3 [45].

LLMs run on LLM ecosystems, which are *"non-trivially heterogenous groups of computer systems, distributed in nature"* [46], and spanning across all three layers of the Compute Continuum: endpoint, edge, and cloud [16]. We argue that predicting LLM ecosystems is a society-critical yet non-trivial simulation problem that could lead to significant service improvements, cost savings, and greener LLMs towards a better sustainability of worldwide digital services. Simulation enables large-scale and fine-grained exploration, analysis, and comparison of systems technologies [7, 47]. The constant accelerating, increasing rate and demand for computing power, further exacerbated by the public-wide availability of LLM, has led to a substantial expansion of datacenter infrastructure, especially in scale and complexity, making datacenter simulation essential

[2] GPT-4 is estimated to have used 25,000 NVIDIA A100 GPUs run for 3-4 months [39, 40, 5]. Official numbers are undisclosed by OpenAI, for undisclosed reasons.

from economical, performance, and environmental perspectives [7, 48]. The climate impact of experimentation through simulation of a datacenter configuration under workload, compared to the climate impact of experimenting with a real-life building, configuration, and running the workload, is 8 to 12 orders of magnitude lower, assuming the simulation is accurate and correct [49, 7, 47]; for example, an analysis conducted by Mastenbroek et al. estimate a ratio of 1:116,000,000,000 in energy consumed to conduct simulations over the equivalent real-world experiments [7]. Although the high importance of simulating ICT infrastructure which hosts highly resource-hungry systems, the current state-of-the-art simulators can predict only individual components e.g., memory, CPUs, GPUs, networking, or only shallowly integrated; it has never been proposed a simulator, nor a comprehensive reference architecture of such an instrument, able to simulate performance, sustainability, and efficiency of LLM ecosystems run on large-scale ICT infrastructure.

Although such a simulation instrument does not exist, the development and hosting of LLM ecosystems is only accelerating and considered to be a modern-day Moore's law [50, 6], with number of parameters growing exponentially and increasing the performance gap between LLM size and hardware performance [51, 52, 53]. While most LLM-oriented research is focused on improving the performance of LLM training, towards higher accuracy, higher throughput, and lower latency, the high-level picture of the LLM ecosystem [51], highly integrated and heavily distributed, is yet blurry and remains unexplored [54]. One approach to meet the performance-hardware gap, and host the increasingly heavy LLM ecosystems, is scaling up the hardware used for training and inference [50, 18]; this is the current approach adopted by AI giants [55, 56], yet projected to reach soon the so-called "modern-day Moore's law" [51, 6]. Another approach to bridge this gap is by carefully anticipating datacenters, through accurate and reliable simulation processes that predict ICT infrastructure under real-world workloads and systems (e.g., LLM ecosystems); such anticipation can happen both before building datacenters, or after building, during operation, helping in dynamic adjustments and resource allocation [17, 48, 7]. Many vetted and community-wide tested simulators already exist [54, 57, 58, 7, 59]. However, none support the simulation of performance, sustainability, and efficiency of LLM ecosystems. Exacerbating the simulation challenge, there is no comprehensive reference architecture for LLM ecosystems, which could be further integrated into simulation processes.

In this work, we identify and address the major sustainability concerns raised by LLMs, the increasing performance gap between LLM ecosystems and ICT infrastructure, and the efficiency concerns of LLM ecosystems. We identify the lack of understanding of how these ecosystems operate and how their internal (eco)systems interact. Addressing these community-knowledge gaps, we propose a high-level conceptual model of the LLM continuum, which we design and validate across real-world ecosystems. We refer to this conceptual model as *reference architecture.* Then, we propose Kavier, a scientific instrument for simulating the LLM continuum; our instrument adopts *discrete-event simulation*, where the operation of a system is represented as a sequence of events over time, with the assumption that no changes occur in-between events [60, 7]. We design a Kavier capable of predicting the performance, sustainability, and efficiency of LLM ecosystems under inference, as well as cache-aware simulation. We then implement a prototype of this design, which we release as open science. We identify the absence of traces showing the relationship between the amount of prefill/decode tokens and their impact on performance. To address this challenge, we deploy LLM ecosystems on real-world infrastructure and conduct measurements. Lastly, through trace-based experimentation, we successfully validate the engineered prototype and demonstrate the superiority of simulation-driven experiments over real-world infrastructure experimentation. We then evaluate the impact of different caching policies on performance, sustainability, and efficiency by first analyzing prefix caching and subsequently examining KV-Caching in autoregressive transformer LLMs.

1.1 Problem Statement

Our society and economy are increasingly dependent on AI services, especially on LLMs, which are being increasingly embedded in operational processes across industry [19, 20], academia [21, 22, 23, 24], and government [25, 26, 27]. However, LLMs are not free to train and host, and surely not cheap: inference of ChatGPT consumes energy, daily, on par with approximately 40,000 Dutch houses [19, 32], GPT-4 training time is estimated at 5k-10k GPU years [39, 40, 5], and the training process of GPT-3, an already "small" model by nowadays standards, consumed 553 metric tons of CO2 [30]. AI services, including LLM ecosystems, are trained and hosted on large-scale ICT infrastructure [1, 2]; therefore, the performance and sustainability of LLM services is directly dependent on the performance and sustainability of the datacenter on which the systems are run.

Proven to be critical instruments in datacenter designing, scaling, maintaining, and building, many datacenter simulators have been developed and used worldwide [54, 57, 58, 7, 59]. Simulation is a powerful tool that can help stakeholders anticipate ICT infrastructure at a fraction of a cost; *Mastenbroek et al.* estimate a ratio of 1:116,000,000,000 in energy consumed to conduct simulations over the equivalent real-world experiments [48, 49].

An alternative to simulation is scaling up the infrastructure [56, 55] (“scaling-by-credit-card” [49]). This approach, albeit currently functional, is unsustainable in the long run, leading to a projected “modern-day Moore’s law” [50, 6], where the number of parameters grows exponentially, and the hardware performance logarithmically, at best linearly [51, 52, 53]. To cover this gap, we propose carefully and responsibly anticipating datacenters hosting LLM ecosystems, through accurate and reliable simulation processes. Therefore, we identify **PS1**:

PS1 Although robustly simulating ICT infrastructure is highly important for the community, especially simulating LLM ecosystems, no simulator currently is capable of predicting performance, sustainability, and efficiency of LLM ecosystems under inference. Without such a scientific instrument, exploration of LLM ecosystems, timely and efficiently, can be hindered.

To rigorously design a datacenter simulator, tailored to predicting LLM ecosystems, the state-of-the-art is materializing a reference architecture into a simulation tool or instrument [17, 7]. However the high importance of simulation and, thus, the high importance of a reference architecture, currently, there is no comprehensive reference architecture for inference of LLM ecosystems, hence exacerbating PS1. Although progress has been made in this direction, proposed reference architectures are incomplete [61], non-inference oriented [61, 62, 63], assume a (too) high degree of homogeneity of LLM ecosystems [62], vetted, following state-of-the-art approaches in distributed systems, but too universal [16], or do not follow a distributed systems approach [62]. The existing reference architectures are not necessarily wrong, yet they are unsuitable for materializing within a unitary simulation entity.

We identify various ecosystems for serving LLM inference in practice, such as the IBM inference ecosystem, highly homogeneous and self-contained, the Databricks inference ecosystem, highly heterogeneous and self-contained, and the inference ecosystem envisioned by Ubicloud, highly heterogeneous and distributed. We identify the main requirement of proposing a reference architecutre abstract enough to model these distinct natures of these ecosystems, while still being specific enough to model intricate behaviour of LLM inference as opposed to traditional, more homogeneous computer ecosystems (e.g., storage, video streaming).

Moreover, we identify very visible effects of the lack of a reference architecture over the compute continuum of LLM ecosystems under inference. The lack of comprehension of this conceptual model leads to incomparable designs, inability to discuss practical shortcomings of existing ecosystems, and difficulties in operating and expanding existing ecosystems.

In this work, we propose the first reference architecture of LLM ecosystems under inference and map our reference architecture to numerous vetted, standard, and universal reference architecture, thus generalizing our model to even more applications that can leverage LLM pipelines. In other words, we are generalizing over the continuum and tailoring for LLM inference pipelines. This raises **PS2**:

PS2 Currently, there is no comprehensive reference architecture for inference of LLM ecosystems.

Recent advances in KV-Caching optimization focus on local improvements: PyramidInfer achieves double throughput and halves GPU memory reduction in KV-Cache, using layer-wise dynamic allocation [64]; AIBrix proposes a distributed KV-Cache approach, which boosts token reuse across nodes, leading to a 50% increase in throughput and a 70% reduction in inference latency [65]; Jenga introduces a two-level memory allocator that reduces fragmentation by 80% through LCM-based page sizing and request-aware allocation, thus improving throughput by 1.8x in heterogenous LLMs. However, these approaches neglect system-wide impacts, primarily focusing on KV-Cache-GPU interaction, without modeling datacenter-scale performance, energy usage, or CO2 emissions. This leads to a critical gap: current KV-Caching research focuses on isolated operational layers of the Compute Continuum, mainly KV-GPU, while this work pioneers vertical simulation of the KV-Caching system, as integrated within an LLM ecosystem. Therefore, we identify **PS3**:

PS3 Current research and experimentation focuses on isolated operational layers of the Compute Continuum, mainly on the interaction between KV-Caches and GPUs, and fails to provide a vertical simulation of the KV-Caching system as integrated within a unitary, highly-distributed, and heterogeneous LLM ecosystem.

1.2 Research Questions

To address the aforementioned challenges, we raise the main research question **(MRQ)**, from which we refine a sequence of four research questions **(RQ)**.

MRQ How to enable analysis of LLM ecosystems, through discrete-event simulation?

Research Question 1

LLM ecosystems have a massive societal impact when run at a worldwide scale, and raise significant sustainability, performance, and economic concerns, especially when workload grows [2, 1, 3]. LLM ecosystems are run on large-scale infrastructure, resource-hungry, and with an increasing gap between the LLM needed resources and infrastructure capabilities [51, 6]; scaling up datacenters is only a temporary fix [55, 56, 6]. As the size and demand of ICT infrastructure grow, we envision simulators and simulation-based digital twins becoming primary decision-making tools, helping to meet Service Level Objectives (SLOs) without trading sustainability. LLM ecosystems become increasingly heterogeneous [2, 51], making simulation a non-trivial, yet critical operation. The simulation challenge is exacerbated by the absence of a comprehensive reference architecture of LLM ecosystems.

Toward answering RQ1, we propose a high-level abstraction of LLM ecosystems, leveraging the entire process from user's input to the system output. We identify the main components of such a system tailored with industry-standard technologies, and individually describe each component, its purpose, and its interaction with other components. Proposing a high-level and comprehensive abstraction of LLM ecosystems raises the research question:

RQ1 How to synthesize and validate a reference architecture of LLM ecosystems?

Research Question 2

It has never been proposed a scientific instrument for simulating the performance, sustainability, and efficiency of LLM ecosystems under inference, following a cache-aware and discrete-event simulation model. We identify the main challenge of proposing a design of such a scientific instrument which ensures not only meeting the functional requirements, but also simulating with a close-to-reality accuracy, lightning-fast performance, and seamless integration with a peer-reviewed and community-vetted datacenter simulator.

Toward answering RQ2, we propose Kavier, a scientific instrument for (KV)-cache-aware simulation of the continuum of LLM ecosystems under inference. Following AtLarge Design Methodology [14], we establish a set of functional and non-functional requirements to guide our design process. Then, also adhering to the reference architecture obtained after answering RQ1, we analyze multiple design choices and select the best-aligned decisions with the established requirements. Then, we focus on each core simulation component (e.g., performance, sustainability, and efficiency) and individually detail the simulation approach for each such model. We also detail the cache-aware simulation component and how caches affect prefill and decode time. The design process is a critical and non-trivial step in the research process, which raises the main research question:

RQ2 How to design Kavier, a scientific instrument for cache-aware simulation analysis of the performance, sustainability, and efficiency of LLM ecosystems under inference?

Research Question 3

Proposing a (successful) novel simulation concept and scientific instrument is a rarity in our field and represents a potential massive-scale contribution if widely adopted, especially in widely used technologies such as LLM ecosystems, heavily reliant on KV-Caches, with potential improvements of orders of magnitude [2, 47]. The main challenge is demonstrating the ability to rigorously implement and integrate Kavier into a vetted simulator. This raises three sub-challenges. Firstly, we identify the challenge of materializing the design proposed in RQ2 into an engineered prototype with minimal redundancy, maximized accuracy, performance, integration, and abstraction, following industry-standard, state-of-the-art techniques. Secondly, we identify the challenge of integrating the engineered prototype with a top-tier, peer-reviewed simulator while respecting the functional and non-functional requirements. Thirdly, we identify the challenge of integration towards further development and research processes, following principles of open-source and open-science, and allowing for further steps towards simulating LLM ecosystems, following the reference architecture proposed in RQ1.

Toward answering RQ3, we propose Kavier, a scientific instrument able to simulate LLM ecosystems and strictly adhere to the design obtained by answering RQ2. To answer RQ3, we identify the main requirement of integrating Kavier within a large-scale, top-tier datacenter simulator. We engineer Kavier as able to predict by leveraging multiple simulation models, following principles of Multi-Model [47, 66, 67] and Meta-Model simulation [68, 47]. We integrate Kavier within a top-tier datacenter simulator, and leverage its peer-reviewed capabilities of simulating sustainability. The implementation and component raise the research question:

RQ3 How to implement and integrate Kavier within a peer-reviewed, discrete-event datacenter simulator?

Research Question 4

Reiterating the statement above, proposing a (successful) novel simulation concept and scientific instrument is a rarity, yet with society-wide impact [68, 7]. Towards evaluating Kavier, we design three experiments. The experiments focus both on quantifying and evaluating Kavier, against well-defined criteria (i.e., functional and non-functional requirements), and on exploring real-world scenarios using Kavier (e.g., exploring sustainability - CO2 emissions, energy consumption - of LLM ecosystems, evaluating workload performance).

To answer RQ4, we establish the experiment setup and synthesize an overview of the experiments. State-of-the-art methodology in the field adopts simulation to check various operational properties (both functional-, and especially non-functional requirements) for distributed systems and ecosystems [14, 69, 17, 70, 7]. We firstly quantify Kavier's accuracy and its performance during simulation. We further explore two real-world scenarios using the just-designed and prototyped instrument, and summarize the main findings. The experimentation journey raises the research question:

RQ4 How to evaluate a Kavier prototype with trace-based realistic scenarios?

1.3 Approach

Throughout the research and engineering process, we approach the problem statement and the subsequent research questions with a distributed systems approach, *"a combination of conceptual, technical, and experimental work"* [48], guided by the state-of-the-art AtLarge Design Process [14].

To answer RQ1, we analyze existing systems and technical documentation and discuss with selected experts in the field various variants of reference architectures of LLM ecosystems, and we contrast these with a set of selected peer-reviewed articles in the community. We summarize, per architecture, the positives and negatives. We present this overview in Chapter 2. Further, in Chapter 3, we propose a high-level abstraction of LLM ecosystems, leveraging the entire process, from the user's input to the system's output. We follow a distributed systems approach in the modeling process and model multiple layers of abstraction, focusing on how different components connect and interact. We describe both the reference architecture, from a high-level perspective, and individually describe the functionality and scope of each component of the distributed LLM ecosystem. We then detail the KV-Caching system and provide an extensive description of its integration within the system. Lastly, we validate the proposed reference architecture against LLM inference ecosystems from the industry and against a peer-reviewed reference architecture of the ICT Compute Continuum.

To answer RQ2, we propose Kavier, a first-of-its-kind discrete-event and cache-aware simulation instrument for predicting performance, sustainability, and efficiency of LLM ecosystems under inference. Following the vetted AtLarge Vision on the Design of Distributed Systems and Ecosystems [14], we center RQ2 towards researching a rigorous design of Kavier. We synthesize functional and non-functional requirements of the simulation system, with a focus on accuracy, performance, and system-wide embed of Kavier; we integrate the latest, state-of-the-art simulation techniques, such as simulation based on multi- and meta-models[3]. Then, we propose a high-level design for a simulation instrument and detail design choices, analysis, and simulation components and models of simulator.

To answer RQ3, we implement the design from RQ2 of Kavier, and produce an engineered prototype, following state-of-the-art software engineering technologies, methods, and principles [71]. We integrate Kavier within a top-tier, vetted simulator to simulate the inference process of LLM ecosystems. We detail the engineering and integration process in Chapter 5.

To answer RQ4, we evaluate the developed prototype of Kavier against the established functional and non-functional requirements using real-world scenarios and data. After answering RQ4, thus at the end of Chapter 6, also corroborated with analysis from previous chapters, we successfully validate all the functional and non-functional requirements of Kavier. We run trace-based experiments through a built prototype and analyze the impacts of various configurations of and workloads on metrics as performance, sustainability, and efficiency.

1.4 Contributions

This paper will impact the scientific community by providing a reference architecture of the most rapidly growing technology of the 2020s: LLMs and LLM ecosystems. A comprehensive reference architecture of the inference process for LLM ecosystems would be beneficial in generating a base of knowledge on how to design and build LLM ecosystems, as well as in easing the understanding of how the components of LLM ecosystems interact, communicate, and function together. Furthermore, such an abstraction of LLM ecosystems could facilitate further research in various directions, such as system simulation or simulation of individual components (e.g., KV-Cache). A reference architecture and simulation of the Compute Continuum behind LLM ecosystems is critical, especially with the rapid growth of these services; further exploration of these techniques could aim to responsibly *massivize* LLM ecosystems.

With this research, our key contributions are:

C1 We conduct an unsystematic literature study and detailed technology analysis to identify, synthesize, and characterize existing reference architectures of, preferably, but not restricted to, inference. We review existing system models and analyze the pros and cons of each selected model. We design a reference architecture for LLM ecosystems under inference, then validate this conceptual model against industry-leading LLM ecosystems, scientific community standards, and through structured and non-structured discussions with experts. We thus address **PS2**.

C2 We design Kavier, a simulator for predicting LLM ecosystems as modular and integrable with datacenter simulators. Kavier is a first-of-its-kind tool, a simulator able to predict performance, sustainability, and efficiency of LLM ecosystems under inference, following a discrete-event and cache-aware simulation model. We thus address **PS1** and contribute to addressing **PS3**.

[3] Multi-Model, or simulation using multiple models, is a novel simulation technique which uses multiple models for predicting datacenter infrastructure under workloads. These individual models are run in parallel, without interfering, and their predictions are further leveraged within a unitary prediction system, towards providing the user with a better explanation of the simulation results. The Meta-Model is an aggregation model that predicts using other models' predictions.

- **C3** We prototype Kavier, following state-of-the-art software engineering technologies and principles. We further integrate Kavier into a peer-reviewed, top-tier datacenter simulator. We use OpenDC, an open-source platform for cloud datacenter simulation, built through 8+ years of development and operations, and vetted across numerous venues [7, 48, 17]. Following principles of open science, we release the open-source integration Kavier-OpenDC. Engineering this prototype, thus, addresses **PS1** and contributes to addressing **PS3** through simulation-driven experimentation.
- **C4** We evaluate Kavier-OpenDC integration using real-world experimentation and traces. We seek traces that show the impact of the prefill/decode length on performance metrics, yet find no such traces available; we thus deploy and trace LLM ecosystems; we release these traces as open science. We also release the validation of Kavier's accuracy and performance as open science. Lastly, we analyze, with Kavier and trace-based experiments, the caching impacts on LLM performance, sustainability, and efficiency. We thus address **PS3**.
- **C5** We release all the artifacts in this work as FAIR [72] datasets and software. The artefacts from this thesis have been peer-reviewed by members of AtLarge Research Group and are available via: `https://github.com/atlarge-research/On-Simulating-LLM-Ecosystems-under-Inference`.

This work also represents the culmination of over eight years of contact with the computer science field, out of which three years were invested in intense academic activities, research, and piles of accumulated knowledge. This paper has a significant impact on my personal development as an independent researcher, with contributions to the computer-science community and worldwide society, towards *Massivizing Computer Systems*. Many thanks to Alexandru, Animesh, and team AtLarge (more in §Acknowledgements).

1.5 Impact on Society and Computer Systems Community

Through our contributions, we envision a significant impact on society and the computer systems community.

- **I1** We anticipate our proposed reference architecture to be beneficial in generating base knowledge on how to design, engineer, operate, and expand LLM Ecosystems. Such a conceptual model could be a major step towards standardisation and allow stakeholders to compare existing ecosystems, identify strong and weak points, and take better-informed decisions on improving efficiency. This would, thus, help to address current efficiency concerns related to LLMs, such as low performance, high energy, or high CO2 footprint.
- **I2** We anticipate Kavier as a simulator which could be adopted by large datacenter providers, to anticipate how various ICT configurations would function under various large- and massive-scale workloads of LLM inference. We envision a multi-step approach, starting from our collaborators as a proof-of-concept (e.g., IBM, Solvinity), and internationally scaling to the largest LLM providers (e.g., Google, Meta, OpenAI). A large-scale adoption of simulation-driven experimentation would allow operators and C-level stakeholders to better reason about their infrastructure and products, and potentially alleviate the concerning resource over-exploitation.
- **I3** We anticipate the LLM Trace Archive introduced in this work, a FAIR dataset [72], to be highly beneficial for researchers and students exploring LLM inference in the future. A unified dataset would thus alleviate efforts otherwise spent on data collection and would instead allow scientists to focus on other research processes at a higher depth. Furthermore, the LLM Trace Archive contains traces unique in the community, and we are the first to FAIRly release measurements on the relationship between the amount of prefill and decode tokens and the system performance. Releasing these traces thus offers researchers access to information otherwise inaccessible (inexistent), difficult to obtain (e.g., via tracing), or not possible to obtain in case of lack of access to such infrastructure.
- **I4** We plan to develop educational material around simulating LLM ecosystems, aided by Kavier, and deliver as a series of interactive workshops, seminars, and assignments to educate groups of various academic ages. Furthermore, thanks to the FAIR nature of all our contributions, such material can be developed both by us and by other researchers and educators from the community, and can be in-depth explored by students who would engage in these educational activities. We envision expanding the

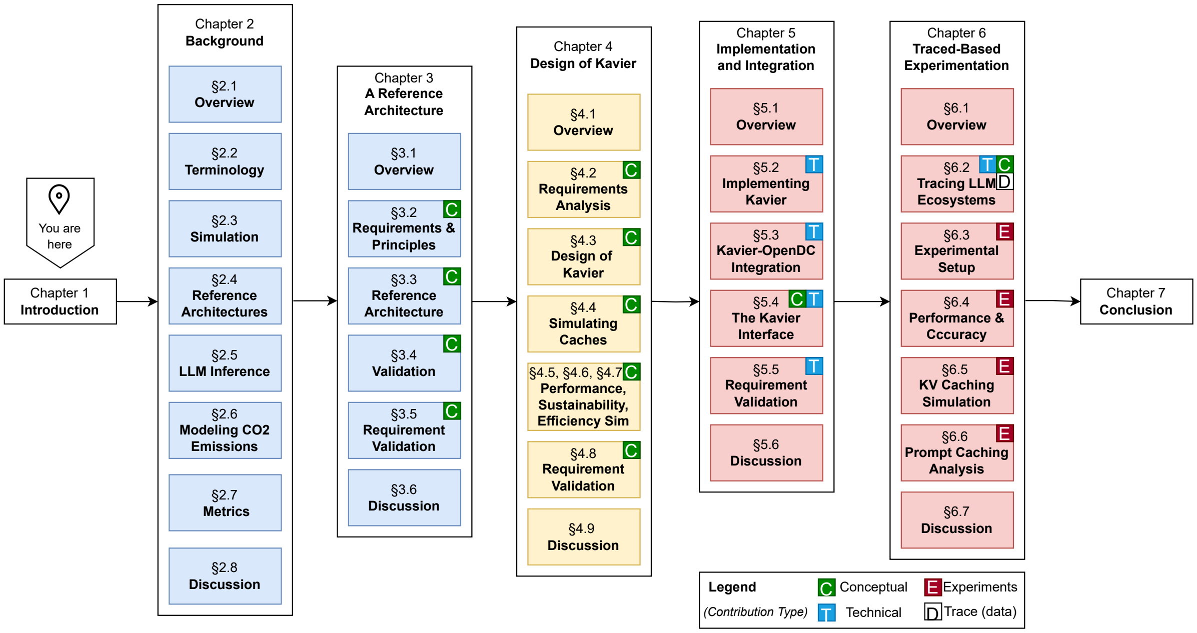

Figure 1.1: The structure of this thesis.

Modern Distributed Systems MOOC course on edX[4], which uses a form of OpenDC that leverages some of these concepts and will include an exercise based on Kavier in the next edition.

1.6 Plagiarism Declaration

I confirm that this thesis is my own work, is not copied from any source (person, Internet, or machine), and has not been submitted elsewhere for assessment. The work, findings, and formulations that do not represent my contribution are given explicit recognition via citations. The plagiarism declaration excepts Section 1.6, which is *ad litteram* copied from the template report :).

1.7 Thesis Structure

Figure 1.1 visually represents the structure of this work, and highlights the four main types of contributions of this work, to the scientific community: conceptual (C), technical (software) (T), contributions from measurements of real-world LLM ecosystems synthesized in data-traces (D), and contributions from trace-based experiments (E).

In Chapter 2, we describe relevant background information on LLM ecosystems, simulators, and latest simulation innovations in the community. In Chapter 3, we propose a Reference Architecture of LLM ecosystems under inference and validate it by aligning this conceptual model with industry ecosystems and peer-reviewed reference architectures of the Compute Continuum [16]. In Chapter 4, we design Kavier following vetted design processes in the computer systems community [14, 1]. In Chapter 5, we implement Kavier and integrate the resulted prototype within OpenDC, a state-of-the-art and peer-reviewed datacenter simulator with over 8 years of development [7]. With the Kavier-OpenDC integration, in Chapter 6 we evaluate Kavier-prototype with trace-based realistic scenarios, following experimentation processes introduced by AtLarge [7, 17, 48, 68], currently standards in the computer systems community.

[4] https://www.edx.org/learn/computer-science/delft-university-of-technology-modern-distributed-systems

2

Background

In this chapter, we present a comprehensive, yet not exhaustive, background on subjects relevant to reference architectures, simulation, and LLM inference. Sections 2.2.2, 2.3, 2.6, and 2.7, are adapted from my honours programme thesis *"M3SA: Exploring the Performance and Climate Impact of Datacenters by Multi-Model Simulation and Analysis"* [47], authored by Radu Nicolae (myself), and supervised by Prof. Dr. Ir. Cav. Alexandru Iosup and Dante Niewenhuis.

2.1 Overview

We now present an overview of this chapter through a top-down presentation. In this chapter, our contribution is six-fold:

1. We firstly present, in Section 2.2, terminology used in operating LLMs and ICT infrastructure and introduce concepts such as *token*, *prefill/decode*, *workload*, *trace*, or *model*, further used for the rest of this work.
2. In Section 2.3, we provide background on datacenter simulation, and expand on a peer-reviewed, community-vetted, open-source datacenter simulator.
3. In Section 2.4 we conduct an analysis of existing reference architectures of LLM ecosystems under inference, and present for each architecture advantages and drawbacks. This is a crucial foundation for Chapter 3, where we propose a comprehensive, state-of-the-art, following a distributed-systems approach reference architecture.
4. We then provide background and how our community addresses, through simulation, the emerging concern of CO2 emissions from massive-scale ICT infrastructure under workload (Section 2.6).
5. We then present background on KV-Caching, with a top-down approach, starting from what KV-Caching is, what its main functions are, and delving into the latest scientific discoveries on KV-Caching and its potential magnitude-scale impact on system throughput and latency (Section 2.5).
6. Then, in Section 2.7, we offer an overview of metrics used in our community to quantify ecosystems (e.g., power usage effectiveness, sustainability, performance, efficiency) and metrics to quantify simulation accuracy (e.g., MAPE).

2.2 Terminology

We now present terminology used in this work. We begin by defining terminology related to AI and LLM ecosystems in Section 2.2.1, then provide terminology related to ICT simulation in Section 2.2.2.

2.2.1 A background on terminology

AI Inference is *"the ability of trained AI models to recognize patterns and draw conclusions from information that they haven't seen before"* [73].

"A token is a collection of characters that has semantic meaning for a model. Tokenization is the process of converting the words in your prompt into tokens" [74]. In this work, we consider one token to be one word.

Prefill is *"the stage in which the model processes the prompt tokens of a new request. It computes the transformer attention and stores the Key (K) Value (V) tensors of the attention for each token and each layer into the KV cache blocks"* [75].

Decode is *"the stage in which the model generates the next output token or the output tensor for intermediate layers repeatedly for ongoing requests. It reuses the stored KV-Cache of all preceding tokens and computes the Query (Q) tensors based on the most recent token"* [75].

KV-Cache is *"the GPU memory region used to store the transformer attention keys and values for each token in a request. It is managed globally across all requests and devices in vLLM"* [75]. We provide more background on KV-Cache(ing) in Section 2.5.

A distributed ecosystem is *"a non-trivially heterogeneous group of computer systems distributed in nature, collectively called constituents. Constituents are autonomous, but often in competition and even antagonistic with each other. The ecosystem structure and organization ensure its collective responsibility: completing functions with humans in the loop, providing desirable non-functional properties that go beyond traditional performance, subject to agreements with clients. Ecosystems experience short- and long-term dynamics: operating well although challenging, possibly changing conditions external to the control of the ecosystem"* [46].

2.2.2 A background on ICT simulation terminology

"Simulation is defined as the imitation of the operation of a system or real-world process over time, and in many cases, manufacturing provides one of the most important applications of simulation" [76]. Simulation provides datacenter stakeholders with operational insights into how the ICT infrastructure behaves under different configurations, workloads, and operational phenomena (e.g., infrastructure failures).

Workloads contain tasks operated on physical machines, virtual machines (VM), or containers [68].

Traces are fine-grained recordings of real-world events, capturing detailed operational data of infrastructure under different workload(s); traces provide a granular view of resource usage, essential for driving simulations or replaying real-world scenarios [68]. In this work, traces are monitored at a constant time granularity to provide details on the computational demand over time, and are crucial in replaying real-world scenarios to predict system behavior, energy consumption, and CO2 emissions.

Predictive models in large-scale computer systems are empirical prediction systems that analyze, combine, and compute various input elements to produce fine-grained output predictions [68, 77]. In other words, *predictive models* are empirical prediction systems that analyze, combine, and compute atomic input elements to produce a comprehensive, sometimes exhaustive output. We use models to predict real-world workloads run on ICT infrastructure with various specifications modeled by users. Models help understand and optimize resource allocation, workload management, and monitoring of overall performance metrics, such as energy consumption and CO2 emissions.

The export rate of the simulator represents the granularity at which the instrument samples and exports simulation data. For example, an export rate of 30 seconds will lead to 2 exported samples per minute. In this work, we address simulations with different sample rates, towards analyzing various metrics (e.g., performance) of the researched and developed tools.

Multi-Model proposes leveraging multiple predictive models, run in parallel, without interference, into a unified tool. The *Meta-Model* simulation vision proposes aggregating multiple predictive models into a unified model, the Meta-Model, able to predict based on other models' predictions [68, 47]. OpenDC supports Multi- and Meta-Model simulation [7, 68].

2.3 The Main Analytical Tool: Simulation

Datacenters serve as vital cloud infrastructure, playing a crucial role in the digital society by serving stakeholders from industry, government, and academia [15, 17, 7, 78]. Extensive research has been conducted in this field, including analyzing and predicting data traffic evolution, developing datacenter simulators, and proposing novel scheduling techniques. In this section, we present the data traffic trends (Section 2.3.1), which increased by one order of magnitude within the last decade; this data, is handled, created, transferred, and reproduced via massive-scale ICT infrastructure, which is in a continuous expansion and growth [49, 78, 79]. The current state-of-the-art consists of simulating before building; we further expand on datacenter simulation frameworks in Section 2.3.2.

2.3.1 Data Traffic Trends

Reinsel et al. analyzed data traffic trends, estimating an one-order-of-magnitude increase to 163ZB reached in 2025, compared to just 16ZB in 2016. These data include 25ZB of critical information and 4ZB of hypercritical data, directly impacting users' health, life, commercial air travel, military security, and numerous other situations. In addition, as of 2025, approximately 75% of the global population is estimated to be connected to the Internet. The research carried out by Reinsel et al., as part of an IDC White Paper sponsored by Seagate, underscores the vital importance of establishing reliable and efficient datacenters, scalable to billions of people and tens of billions of devices [80].

2.3.2 Simulation with OpenDC

"Simulation is defined as the imitation of the operation of a system or real-world process over time, and in many cases, manufacturing provides one of the most important applications of simulation" [76]. Simulation provides datacenter stakeholders with operational insights into how the ICT infrastructure behaves under different configurations, workloads, and operational phenomena (e.g., infrastructure failures).

Iosup et al. analyzed existing datacenter simulators, highlighted, and addressed simulation challenges by introducing OpenDC 1.0 [78], succeeded by OpenDC 2.0 [7], introduced by *Mastenbroek et al.* OpenDC is an open-source platform for modeling, simulation, and experimentation with cloud datacenters. OpenDC 2.0 addresses multiple key challenges:

1. contains models for emerging technologies, such as serverless computing and machine learning workloads running in datacenters;
2. contains models for CO2 emission predictions and models for energy usage predictions, calibrated with real-life data;
3. provides an intuitive interface with enhanced visualization and interaction tools, supporting various input/output formats and metrics. OpenDC 2.0 facilitates the process of designing and sharing (parts of) complex datacenters;
4. provides both a GUI and JSON interfaces towards accommodating a wide range of stakeholders, including experts and general users.

OpenDC 2.0 is a pioneering and re-engineered iteration of the 1.0 prototype, becoming the first simulator to integrate serverless and machine-learning execution while leveraging discrete-event simulation. This simulator integrates a model for the TensorFlow ecosystem and primarily employs Kotlin as the main programming language for the codebase. The authors compare the developed datacenter simulation concepts and architecture with *i) Mathematical Analysis*, albeit faster, too high-level for the processes from a datacenter and with *ii) Real-world experimentation*, which yields accurate results, is non-trivial to run at a large scale due to high energy footprint and extensive waiting times.

With highly precise and accurate simulations, open-source nature, and a wide variety of distinct models used in simulations, OpenDC has proven results through multiple peer-reviewed, award-winning, top-tier publications [70, 81, 78, 49, 7, 69, 48, 17, 17]. We identify the OpenDC simulation framework and the related work as highly relevant for this research.

2.4 The Root of Every Systematic Simulation - the Reference Architecture

Prior to conducting this scientific research, we searched for relevant literature proposing reference architectures (RAs) for LLM ecosystems. We define a set of keywords: *"reference architecture LLM ecosystems,"* and explore existing literature on ACM Digital Library[1], Google Scholar[2], and DBLP[3]. We discover one relevant architecture for LLM ecosystems proposed by *Bucaioni et al.* (Figure 2.1) and a community-vetted reference architecture for the Compute Continuum, proposed by *Jansen et al.* (Section 2.4.2). Alongside the aforementioned RAs, we discover RAs with various purposes (e.g., for deploying LLMs [63]), albeit valuable, are not aligned with the scope of this research. However, although not in direct alignment with the scope of this work, in Section 2.4.3, we present the reference architecture proposed by *Lu et al.* for designing foundational model-based systems, which we envision as a highly relevant envision of the LLM evolution over the next decade(s).

In this section, we present two community-vetted reference architectures for LLM ecosystems and a reference architecture on the evolution of foundational models. For each RA, we present an overview and highlight present and absent key points. We regard this (sub)-section as critical for Chapter 3, in which we envision a comprehensive reference architecture for the inference process of LLM ecosystems, following a distributed systems approach, and mapped to the Compute Continuum.

2.4.1 A Functional Software RA

Overview: *Bucaioni et al.* propose a *"preliminary functional reference architecture as a conceptual framework"* which addresses the lack of systematic reasoning about the design and quality attributes of LLM software systems [62].

The authors conduct a literature survey and identify architectural concerns for "large language model-integrated systems" (LLM ecosystems) and propose a four-layer functional reference architecture emphasizing modular service decomposition and cross-layer monitoring (§III [62]), which we illustrate in Figure 2.1.

(i) Presentation layer: handles multimodal user interactions through a user interface (UI), linked to a connector component, and linked to the other layers via an orchestrator.

(ii) Application logic layer: contains a single, non-shared component, the orchestrator, which dynamically determines workflows based on user input. The orchestrator bridges the (i) presentation layer with the (iii) LLM integration layer. To address scalability concerns, the orchestrator supports asynchronous, event-driven workflows.

(iii) LLM integration layer: is the system's core and handles input-processing, detailing from the very first input formatting, called *pre-processing* (e.g., for single reasoning model workflows) and prompt engineering (e.g., for cascading model workflows, where the user prompt generates multiple, in-LLM prompts), to the *post-processing steps*, such as multi-modal integration, or formatting and translation. Although the authors include the *multi-modal* element in the RA, they do not further expand on this topic.

(iv) Data management layer: "ensures efficient data handling" [62]. This layer includes multiple elements for performance enhancement, such as a vector database for retrieval-augmented generation for knowledge-grounded outputs, thus improving accuracy, following the model of Pinecone's integration with Notion AI [62], or the integration of a memory component, which maintains context across sessions.

All layers: The authors propose two elements that cover the entire continuum and span over all four layers. The monitoring component proposes collecting performance metrics (e.g., latency, throughput), and user feedback. The guardrail component ensures security and privacy, ensuring law-compliance over all the layers; however, this component is still blurry, without a detailed description in the paper.

[1] `https://dl.acm.org/action/doSearch?AllField=reference%20architecture%20llm%20systems`

[2] `https://scholar.google.com/scholar?hl=en&as_sdt=0,5&q=reference+architecture+llm+systems`

[3] `https://dblp.org/search?q=reference%20architecture%20llm%20systems`

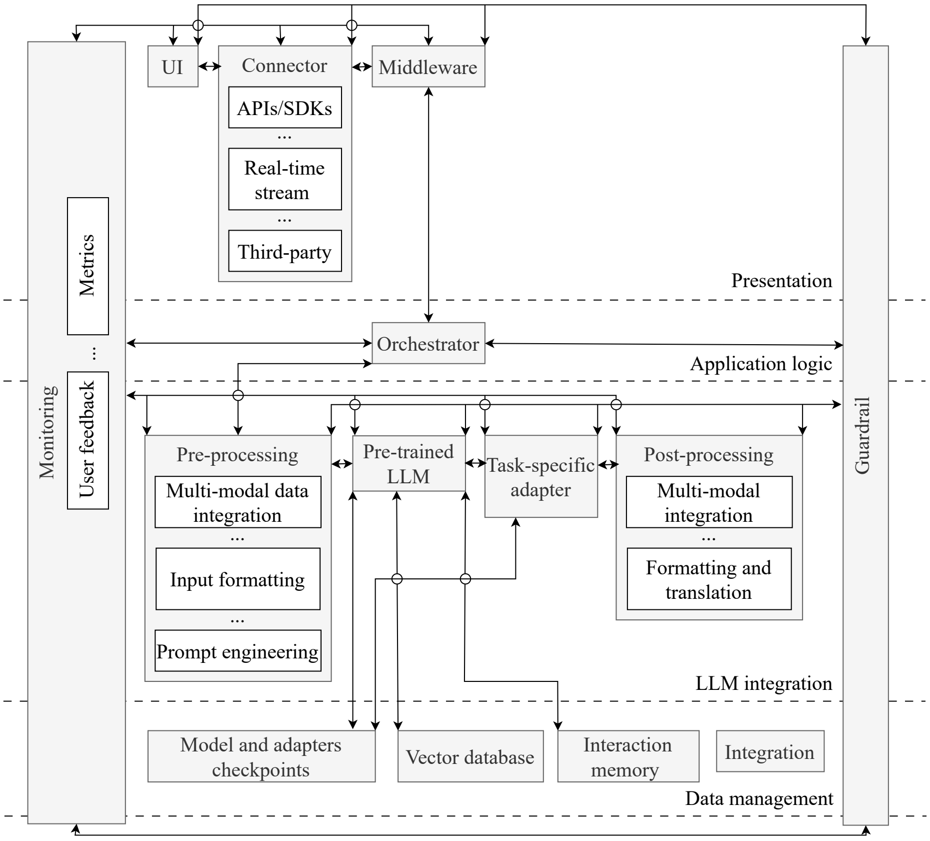

Figure 2.1: Existing reference architecture for LLM-integrated ecosystems, from [62].

Advantages of this RA: *Bucaioni et al.*'s architecture employs layered interoperability, rather than a strict stack, which allows for both vertical flow (layers (i)-(iv)) and horizontal flow, with monitoring and guardrail elements covering each layer. This RA allows for modularity through a high-level approach, while maintaining end-to-end workflow cohesion.

Weak points: While *Bucaioni et al.* propose a valid, community-wide recognized, and thus peer-reviewed, reference architecture, we argue this RA exhibits two critical gaps, making it insufficient for abstracting inference of LLM ecosystems operating across the Compute Continuum. While these elements don't invalidate the correctness of the RA proposed by *Bucaioni et al.*, it makes the RA insufficient for the scope of our work.

G1: High homogeneity: We argue this reference architecture proposes a too-high degree of homogeneity, especially in the memory systems and computing infrastructure. The reference architecture contains an *Interaction memory* component, which, however, fails to reflect the real-world degree of heterogeneity, especially hierarchical layers of the memory component. Furthermore, the computing infrastructure is disregarded from this architecture, making it unclear where the heavyweight computation is connected to.

G2: Blurry memory component: While this reference architecture abstracts (some of) the components of the LLM ecosystems under inference workload, it fails to provide an in-depth model of the memory component, which, in reality, is highly hierarchical, and contains elements which can impact performance by orders of magnitude. One such component, the KV-Caching system (background provided in Section 2.5), is a key component of nowadays massive-scale LLM ecosystems, and a main focus of this paper. The reference architecture proposed by *Bucaioni et al.*, fails to present how the KV-Caching system is integrated with the memory hierarchy and with the Compute Continuum of the system

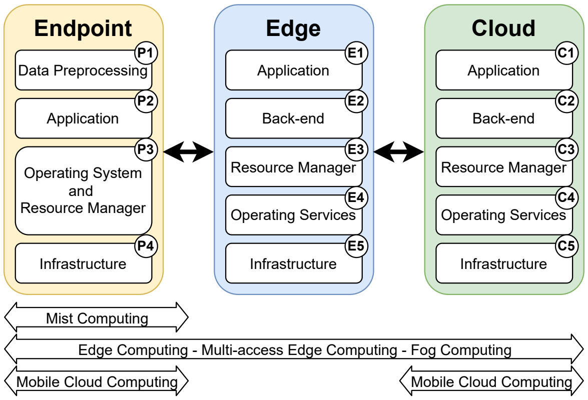

Figure 2.2: Reference architecture for the Compute Continuum taken from [16].

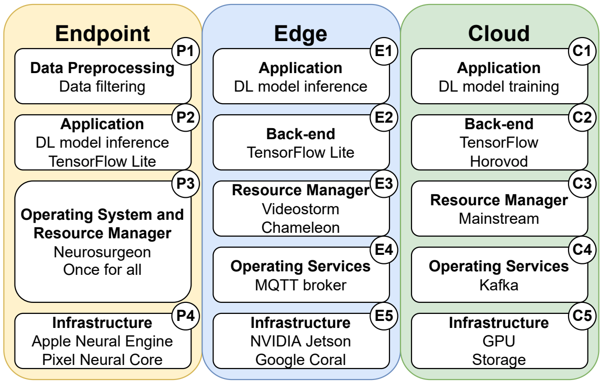

Figure 2.3: Deep learning architecture applied on the reference architecture proposed and taken from [16].

2.4.2 The Compute Continuum

Jansen et al. [16] propose a three-tier reference architecture of a Compute Continuum, as depicted in Figure 2.2, comprising cloud, edge, and endpoint, and addressing the fragmentation of 17 existing computing models, via systematic synthesis of common characteristics and design patterns, following the community-vetted AtLarge Design Process [14].

Overview: *Jansen et al.* have a five-fold contribution, out of which contributions (ii) and (iii) are highly representative for this work. In [16], the authors: (i) conduct a literature survey on computing models, synthesize properties, and identify opportunities for unification; (ii) propose a unified reference architecture, the first in the community to consider the entire edge-cloud Compute Continuum; (iii) synthesize two domain-specific architectures, one for deep learning, illustrated in Figure 2.3, highly relevant for the scope of our work, and one for industrial IoT; (iv) offer an open-source workload deployment and benchmarking framework; (v) formulate analytical performance models for exploring workload deployment scenarios in the continuum.

The authors propose the SPEG-RG reference architecture illustrated in Figure 2.2. The architecture comprises three tiers of systems: cloud, edge, and endpoint, where each component is further expanded in §III [16].

Endpoints are *"the last hop of processing and connectivity to users"* [16], typically single-tenant [82], resource and energy-constrained (e.g., smartphones, cameras, sensors), with four main responsibilities: (P1) pre-processing data before pushing to the edge and cloud servers, (P2) running user-defined logic to process incoming data and make decisions, (P3) OS-level resource managers and multiplexers for workload management, and (P4) the physical computing available to the operating systems (e.g., processing units, memory, network, storage).

Edge and cloud are presented as sharing the same high-level design, both able to run multi-tenant workloads on shared infrastructure [16]. Edge and cloud mainly differ by the resources and energy constraints, where cloud operates at a higher scale. Unlike cloud and uniquely at the edge, there should be support for application offloading both vertically (cloud to edge and back) and horizontally (from one edge system to another) [16, 83]. The authors propose five elements in the edge and cloud: (E1) Applications are the first step from endpoint to cloud and are in the best position to make decisions regarding placements, offloading, scheduling, or conduct other user-defined decision-making processes, towards meeting workload-specific objectives, (E2) the backend represents more general-purpose application execution frameworks, usually light(er)weight in the edge than in the cloud, (E3) Resource Managers manage systems' application-independent physical and virtual resources, such as virtual machines and containers, (E4) Operating services are described as providing ”support to build distributed applications, and their responsibilities include (but are not limited to) communication, metadata management, consensus services, monitoring, storage services, etc." [16], and (E5), similarly to endpoints, yet at orders-of-magnitude higher scale, infrastructure contains compute, memory, networking,

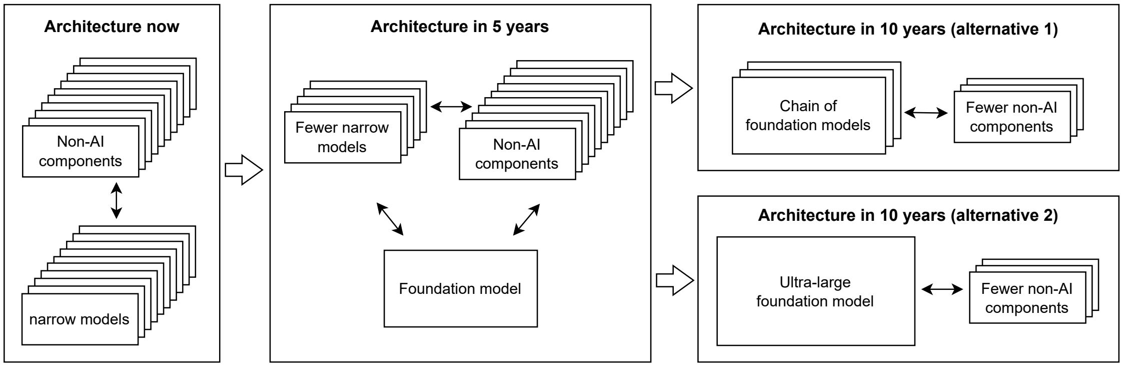

Figure 2.4: From “foundation-model-as-a-connector” to “foundation-model-as-a-monolithic-architecture,” taken from [61].

and storage resources; however, unlike endpoints, infrastructure contains resources split into physical and virtual resources.

Deep Learning architecture mapped to the Compute Continuum: *Jansen et al.* create architectures for deep learning and industrial IoT; we regard the architecture of deep learning systems as highly aligned with the scope of this research and further provide background. In Figure 2.3, we illustrate the deep learning architecture proposed in §III, [16]. As the authors mention, an important trend for deep learning in the continuum is that model training and inference tasks are split across all the tiers of devices (i.e., endpoint, edge, cloud).

Advantages of this RA: The SPEC-RG reference architecture poses several advantages. This is the first peer-reviewed RA in the community to consider the entire edge-cloud Compute Continuum and further synthesize the proposals in a unitary architecture, thus leveraging previously fragmented computing models under a cohesive framework. Furthermore, this RA proves its validity through systematic research methodology and evaluation, following state-of-the-art AtLarge Design Process [14]. Lastly, the authors contribute to open science by providing an engineered prototype of an open-source framework, released as open-source.

Weak points: While comprehensive and, at present, regarded as the state-of-the-art for the computer (eco)systems community, this reference architecture exhibits some limitations in regards to the scope of this work, limitations which, however, do not undermine the validity of the Compute Continuum. The RA focuses primarily on a higher-level design, yet does not present technical depth on how real-world large-scale, highly heterogeneous systems would map to this architecture (e.g., highly distributed LLM ecosystems). Furthermore, the RA proposed by *Jansen et al.* presents a high-level abstraction of e.g., Application components (P1, E1, C1), which, in this work, alongside other components, we expand and detail.

2.4.3 RA for Foundation Model Based Systems

Lu et al. identify a broad consensus that ”foundational models will be the fundamental building blocks for future AI systems” [61]; however, there is a lack fo systematic guidance on the architecture design. The authors present an architecture evolution of AI systems (Figure 2.4), then propose a pattern-oriented reference architecture for designing responsible foundation-model-based systems.

Overview: In this section, we detail Figure 2.4, as *Lu et al.* present architectures in three time periods. The authors identify the architecture now as containing ”many narrow AI models and many non-AI components”, yet this is prone to change over the next decade. The predicted transition in the next five years regards foundational models (FM) as a connector, with one FM, a few narrow models, and many non-AI components.

The authors envision two evolution alternatives of LLM FMs within the next 10 years. *Alternative one* proposes a chain of FMs and only a few non-AI components, an alternative in which most of the software components could be absorbed into the FMs, chained together, without requiring additional training or fine-tuning. Those FMs would be connected via APIs, with external non-AI components that offer additional functionalities (e.g., robotic systems or web search engines). *Alternative two* proposes one ultra-large FM and only a few non-AI components, following a monolithic architecture. The ultra-large FM would be unitary, massive, and capable of performing a variety of tasks by incorporating different types of tasks, from searching, reasoning, self-inputting (self-prompt engineering), and outputting in various shapes and forms. The non-AI component may include context engineering components (e.g., multimodal context injection), prompt engineering components (e.g., AI-powered prompt optimizer), and responsible AI components, ensuring privacy, security, and a continuous risk assessment.

2.5 The Main Components of the LLM Inference Ecosystem – Self-Attention Mechanism and KV-caching

"Attention is all you need" [10]. In 2017, *Vaswani et al.* proposed the Transformer, *"a model architecture eschewing recurrent and instead relying entirely on an attention mechanism"*, a state-of-the-art approach allowing better parallelization, with a working prototype trained for a limited time on limited resources [10]. With approximately 180,000 citations as of 2025, this work revolutionized sequence modeling by replacing recurrence with self-attention, a pivotal contribution in enabling parallelized training and inference towards high throughput and low latency. Although this paper does not explicitly introduce **KV**-caching, the authors adopt an autoregressive approach involving **K**ey and **V**value matrices across time steps to avoid recomputation.

Zhang et al. introduce LLMs as *"autoregressive models that generate tokens iteratively, one at a time"*, with the inference engine storing *"KV caches-intermediate tensors produced by attention layers." "The computation of a new token depends on interactions between its embedding and the previously stored intermediate KV cache tensors [2]"*.

HuggingFace presents KV as follows: KV vectors are used to calculate attention scores; KV scores are calculated depending on the previous tokens [84]. However, this means that *"each prediction depends on the previous tokens"*, which is equivalent to the model performing *"the same computation each time"* [84]. The key-value (KV) vectors are used to calculate attention scores. For autoregressive models, KV scores are calculated every time because the model predicts one token at a time. Each prediction depends on the previous tokens, which means the model performs the same computations each time.

2.5.1 Self-Attention Mechanism

To better comprehend KV-Caching, we first focus on the self-attention mechanism. *Vaswani et al.* describe an attention function as *"mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where a compatibility function of the query with the corresponding key computes the weight assigned to each value."* They propose the following equation, a standard in the nowadays community:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.1}$$

*where Q is the query token for which the model is computing attention, K represents all tokens (**k**eys) in the sequences used to determine the relevance to the query, V contains the information to be aggregated (**v**)alues, weighted by the attention scores, and d_k denotes the dimension of the key vectors.*

Inputs: Q, K, V are each matrices derived from the input embeddings (i.e., the previous layer outputs) via the learned linear transformation. Considering the input sequence of length n and hidden size d, Q and K have the shape of (n, d_k), and V has the shape (n, d_v).

Outputs: The attention function outputs a matrix of shape (n, d_v), where each row is a contextually weighted sum of the value vectors and corresponds to each input position. The attention weights can be interpreted as how much each input token attends to every other token [85].

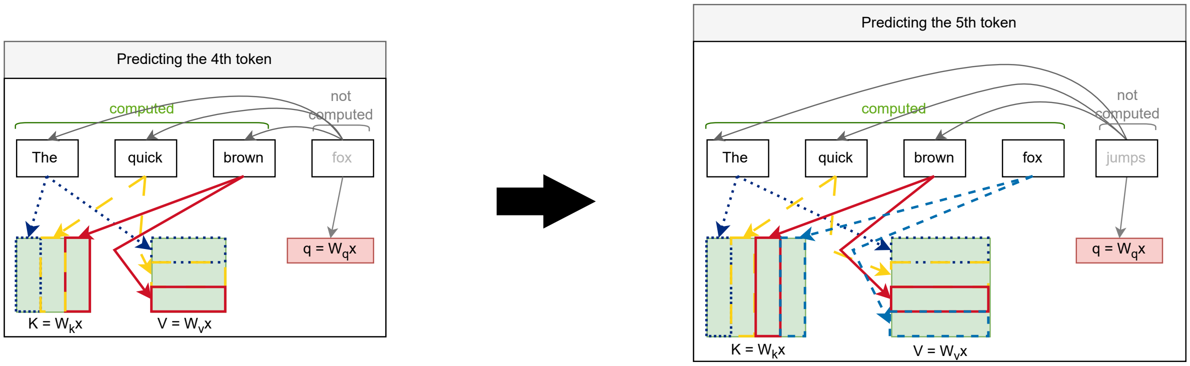

Figure 2.5: LLMs predicting without KV-Caching. $O(n^2)$ time complexity.

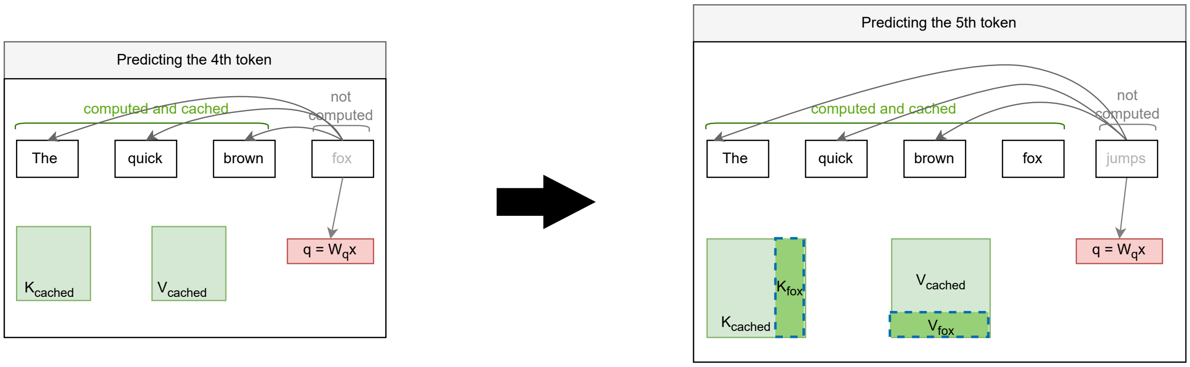

Figure 2.6: LLMs predicting with KV-Caching. $O(n)$ time complexity.

In Figure 2.5, we present an example of an LLM predicting without KV-Caching. For the sake of simplicity, we illustrate a scenario where the LLM contains a short, visually comprehensive sequence.

In the left part of Figure 2.5, the LLM contains the sequence *"the quick brown"*, where each word is a stored token. We generate one word at a time, following the mechanism employed by autoregressive encoding. The K and V matrices contain information about the entire sequence, while the query vector Q contains information only about the last token (i.e., *"fox"*).

In the right part of Figure 2.5, the LLM contains the sequence of *"the quick brown fox"* and predicts the fifth token. We observe that matrices K and V don't change much, but rather receive an additional column and row for each additional token. However, the model still needs to perform the heavy, yet redundant, computational work of computing the key and value vectors for each word.

This approach results in $O(n^2)$ time complexity for total generation, where n represents the number of tokens per input. While, in this example simplified for visual comprehension, $O(n^2)$ is not computationally heavy, the absence of KV-Caching quickly increases the number of computations needed for predicting large phrases.

2.5.2 Key-Value Caching

A KV-Cache stores the calculations from Section 2.5.1 so they can be reused without recomputing them. Efficient caching is crucial for optimizing model performance because it reduces computation time and improves response rates [84].

The KV-Caching proposes a simple, yet powerful technique: when the model reads a new token, it generates the query vector (Q), similarly to the approach presented in Section 2.5.1, yet the system caches the values already computed for the previous tokens to reduce redundancy and multiple calculations for each token.

Instead, the model only computes a new column and a new row for the key-value matrix.

In Figure 2.6, we illustrate the KV-Caching approach, using the same example as in Figure 2.5. With this design, for the sequence *"The quick brown"*, the memory holds two 3-by-3 matrices, one for keys, one for values, and a query vector, similarly to the no-KV-approach. However, when predicting the fourth token, *"fox"*, the system computes only one new column for the key matrix and one new row for the value matrix, instead of recomputing the entire matrix.

2.5.3 KV-Caching and GPU-memory

The cached key and value matrices (i.e., K and V, from Figure 2.6) need to be stored in the GPU's memory, such that the already-computed matrices can be reused when predicting the next token, thus preventing the otherwise redundant computation introduced by the no-KV alternative.

The KV-Cache memory usage can be computed with the formula:

$$memory = 2 \times L \times H \times d \times N \times sizeof(type) \tag{2.2}$$

where L is the number of transformer layers in the model, H is the number of attention heads, d is the dimension per head, N is the number of tokens in the sequence, and $sizeof(type)$ represents the size of the data type in bytes (e.g., float16 represents 2 bytes, float32 4 bytes). The factor of 2 represents storing two matrices, one for keys and one for values.

Bai Li, through Efficient NLP [86], computes the memory needed for OPT-30B [87], a 30-billion-parameter model, a small to medium-sized model. OTP-30B runs inference in 16 bits (i.e., 2 bytes), contains 48 layers, and has 7168 dimensions. In this example, Bai proposed a sequence length of 1024 tokens and a batch size of 128.

$$memory_{kv} = 2 \times L \times H \times d \times N \times sizeof(type) \tag{2.3}$$
$$memory_{kv} = 2 \times 48 \times 128 \times 7168 \times 1024 \times 2 \tag{2.4}$$
$$memory_{kv} = 176,160,768B \approx 176GB \tag{2.5}$$
$$memory_{model} = 2 \times 30B = 60GB \tag{2.6}$$

We thus observe that KV-Caches use 2.9x more memory than the model itself, a common effect for inference scenarios of LLMs, where the KV-Caching is a dominant factor [64, 2, 86]. However, thanks to KV-Caching, the time complexity decreases from $O(n^2)$ to $O(n)$ since the KV-Caching approach prevents the redundant recomputation of previous token scores, and caches these computations instead.

2.6 An Emerging Concern: Modelling CO2 Emissions

Niewenhuis et al. proposed FootPrinter, a *"first-of-its-kind tool that supports datacenter designers and operators in assessing the environmental impact of their datacenter"* [70]. As part of their research, engineering, and evaluation, matching the state-of-the-art AtLarge Design Process [14], the authors proposed and integrated into OpenDC a model able to predict the CO2 emissions of ICT infrastructure under workload.

FootPrinter simulation process leveragies the energy module of OpenDC, which simulates the amount of power draw and energy usage for infrastructure under workloads. Then, using the FootPrinter module, the simulator computes the amount of CO2 emissions based on the carbon intensity at the time and the power draw at the time

Niewenhuis et al. calibrated the researched CO2-emissions model with data from the ENTSO-E Transparency Platform [88]. The efforts have been concertized in a research paper, part of a top-tier conference, which hence confirms the validity of the research methodology, and accuracy and validity of the designed and engineered tool.

2.7 Community Agreement on What to Measure: Metrics

In this section, we present metrics stakeholders use to quantify datacenter sustainability and efficiency, and metrics the community uses to quantify simulation accuracy. In Section 2.7.1, we present metrics that quantify the energy efficiency of ICT infrastructure and emphasize the need for efficient ICT operation. In Section 2.7.2, we present carbon metrics which we use to simulate and quantify the sustainability of LLM ecosystems. In Section 2.7.3 we present metrics, such as throughput and latency, which we use to quantify performance of LLM ecosystems. In Section 2.7.4 we present metrics we use to quantify system's financial and sustainability efficiency. In Section 2.7.5, we present the Mean Absolute Percentage Error (MAPE) ratio, a widely used metric to quantify the accuracy of simulation models and divergence with real-world measurements.

2.7.1 Metrics on quantifying the energy effectiveness of the systems

Designing, building, operating, and expanding energy-efficient datacenters is becoming an increasingly concerning challenge for our increasingly digitalized society. The wide adoption of LLM ecosystems deepens the gap between LLM needs and hardware efficiency, which results in high energy usage. To quantify the efficiency of existing ICT infrastructure, we identify Power Usage Effectiveness (PUE) and Datacenter Performance Efficiency (DCPE) as core metrics.

2.7.1.1 Power Usage Effectiveness (PUE)

Introduced in 2006 by *Malone et al.* [89, 81], Power Usage Effectiveness (PUE) is an end-user tool consisting of a metric *"for understanding how well a datacenter is delivering energy to its information technology equipment"* [90]. PUE is the ratio of the total energy and the energy that is used for the actual computation.

$$PUE = \frac{E_T}{E_{IT}} \tag{2.7}$$

where E_T denotes the total energy used by the datacenter and E_{IT} denotes the energy used by the IT components of the datacenter.

Equation (2.7) [90] provides a high-level mathematical equation to compute the Power Usage Effectiveness factor of an ICT infrastructure. PUE can take values between a minimum of 1.0 and an infinite maximum. Lower values of PUE are better, and the aim is to get as close to 1.0 as possible. A PUE of 1.0 means that the IT equipment uses all the energy received by the datacenter, yet it is impossible to achieve due to the laws of physics.

The Climate Neutral Data Centre Pact [91] mandates that, by 2030, all datacenters must meet the PUE target of 1.3 in cool climates and 1.4 in warm climates. [91]. Although significant improvements in PUE occurred, from an average of 2.6 in 2007 to 1.6 in 2015, the decline has stagnated in recent years (Figure 2.7), while the overall energy usage is alarmingly increasing [70]. Although Google achieved an average annual PUE of 1.1 in 2023 [92], and BTDC (Sweden) set a PUE record of 1.014 in 2021 [93, 94], the global average PUE remains worryingly high, at 1.58 in 2023 [95]. Besides environmental concerns, the sharp increase in energy prices in 2022 has a significant economic impact on administrators of datacenters with a bad (high) PUE factor [69, 48].

To transpose the numbers above to real examples, we will consider a hyperscale datacenter, which aims to improve the average annual PUE. In our hypothesis, we consider the annual PUE of the datacenter equal to 1.58, denoted as x_1. This value represents the average PUE of datacenters worldwide in 2023 [95]. To meet and improve beyond the Pact-mandated metrics, the administrators want to improve and achieve an average annual PUE of 1.25 (i.e., the target PUE), denoted as x_2. We assume that the datacenter consumes 100 GWh per year (i.e., total facility energy) and is denoted as y. We denote the energy used for computation (i.e., IT Equipment Energy) as z_1, for the current PUE, and as z_2, for the target PUE. We assume the average price

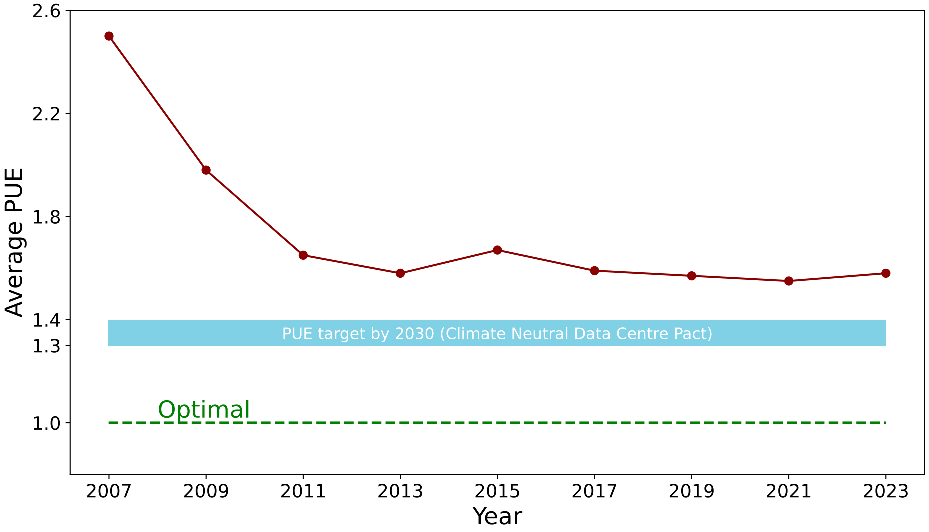

Figure 2.7: PUE Evolution Between 2007 and 2023 [95].

per GWh is approximately €350,000 (i.e., the average price per GWh in 2024, in the Netherlands [96]), and denote as p.

$$p \approx 350,000\,EUR \qquad \text{(approx. price per GWh, Netherlands, 2024 [96])} \tag{2.8}$$
$$x_1 = 1.58 \qquad \text{(current PUE of the datacenter)} \tag{2.9}$$
$$x_2 = 1.25 \qquad \text{(target PUE of the datacenter)} \tag{2.10}$$
$$y = 100\,GWh \qquad \text{(total yearly consumption)} \tag{2.11}$$
$$z_1 = \frac{y}{x_1} \approx 63.29\,GWh \qquad \text{(IT components yearly consumption with the } x_1\text{)} \tag{2.12}$$
$$z_2 = \frac{y}{x_2} = 80.00\,GWh \qquad \text{(IT components yearly consumption with the } x_2\text{)} \tag{2.13}$$

$$i = \frac{|z_1 - z_2|}{z_2} \approx \frac{|x_1 - x_2|}{x_2} \approx 20.89\% \qquad \text{(energy saved)} \tag{2.14}$$
$$\Delta z \approx z_2 - z_1 \approx 16.71\,GWh \qquad \text{(energy saved yearly)} \tag{2.15}$$
$$\Delta p \approx \Delta z \cdot p \approx 5,848,500\,EUR \qquad \text{(money saved yearly)} \tag{2.16}$$

Under the aforementioned hypothesis, we identify a 20.89% improvement in energy consumption, resulting in approximately 16.71 GWh saved per year, equivalent to savings of approximately 5,848,500 EUR.

2.7.1.2 Datacenter Performance Efficiency (DCPE)

Derived from PUE, Datacenter Performance Efficiency (DCPE), also referred to as Compute Power Efficiency (CPE), is a metric used to measure the computational efficiency of datacenters. DCPE was introduced by *Malone et al.* and used to capture the fraction of energy used for computation.

$$DCPE = \frac{U_{IT}}{P} = \frac{U_{IT} \cdot E_{IT}}{E_T} \tag{2.17}$$

U_{IT} is the IT Equipment Utilization, P is PUE, E_{IT} is the energy used by the IT components of the datacenter, E_T is the total energy used by the datacenter.

We observe that even slight changes in the Power Usage Effectiveness metrics significantly impact the datacenter Performance Efficiency factor. To illustrate this, we use the example of the hyperscale datacenter presented in Section 2.7.1.1. We analyze the increase in the DCPE factor between the PUE of $x_1 = 1.58$ and $x_2 = 1.25$. We determine a 26.98% improvement in the DCPE factor.

$$U_{IT} \qquad \text{(IT Equipemnt Utilization)} \tag{2.18}$$

$$d_1 = \frac{U_{IT}}{x_1} = \frac{1}{1.58} = 0.63 \qquad \text{(DCPE for PUE of } x_1 = 1.58\text{)} \tag{2.19}$$

$$d_2 = \frac{U_{IT}}{x_2} = \frac{1}{1.25} = 0.80 \qquad \text{(DCPE for PUE of } x_2 = 1.25\text{)} \tag{2.20}$$

$$p_i = \frac{|d_1 - d_2|}{d_1} = 26.98\% \qquad \text{(Performance Improvement)} \tag{2.21}$$

2.7.2 Metrics on quantifying CO2 footprint of the system

PUE is an excellent metric to quantify ICT infrastructure's performance and energy efficiency. However, PUE does not consider the energy efficiency of applications and workloads [97] and overlooks the type of energy used [70]. While there is a correlation between a datacenter's power draw (i.e., the energy consumed) and the CO2 emissions, several other factors influence the amount of CO2 emitted. Determining the CO2 footprint of the datacenter under a specific workload is an environment-critical, yet not trivial, challenge. Many datacenters use energy from the grid, generated through various sources, with various environmental impacts (e.g., solar, wind, coal). In some cases, energy used from renewable sources, such as wind or solar, can emit up to 20x less CO2 compared to traditional energy sources, such as coal [70, 8].

Niewenhuis et al. presents two types of CO2 emissions in the datacenters: *i) the embodied* carbon footprint and *ii) the operational* carbon footprint. Embodied carbon denotes the manufacturing and production results emissions. Operational Carbon Footprint is the CO2 emissions caused by energy usage during datacenter operations. This work proposes a simulation-based solution to alleviating the concerning and deepening environmental problem of CO2 emissions, focusing on the operational carbon footprint [70].

2.7.2.1 Carbon Intensity

The *Carbon Intensity* of an energy source defines the amount of CO2 emitted per unit of energy used [70]. The measurement unit in the international system is $[gCO_2/kWh]_{S.I.}$. Datacenters utilize energy from the grid [70]; the energy is often provided by multiple sources with distinct Carbon Intensities [70]. Therefore, the Carbon Intensity of the grid is calculated by adding up the Carbon Intensity of each source, proportional to the amount of energy consumed (Equation 2.22).

$$\mathrm{CI}_g = \sum_{s \in S} \mathrm{CI}_\mathrm{s} \cdot \frac{E_\mathrm{s}}{E_\mathrm{g}} \qquad [gCO_2/kWh]_{S.I.} \tag{2.22}$$

where CI_g is the Carbon Intensity of the grid, CI_s is the Carbon Intensity of the source, E_s is the energy from a specific source, E_g is the total grid consumption, s is the selected source, S the set of all available energy sources [70].

2.7.2.2 Carbon Emissions

The *Carbon Emissions* of the grid fluctuate depending on the geographical location (Figure 2.8), time of the day (Figure 2.9), temperature, weather conditions, et cetera. The amount of green energy delivered peaks during the day, while during the night, energy from ”grey” sources (e.g., coal) is predominantly used [98].

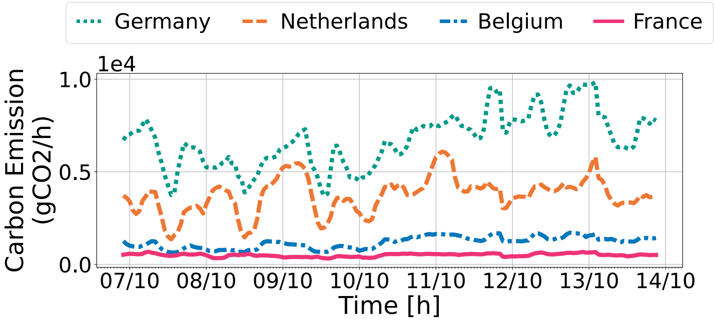

Figure 2.8: CO2 emission fluctuation, location-dependent. Taken with permission from [70].

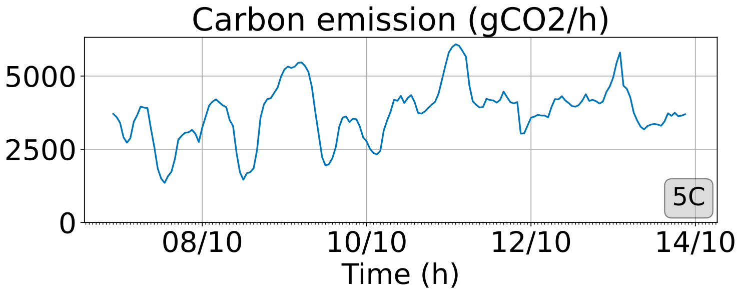

Figure 2.9: CO2 emission fluctuation, over time. Taken with permission from [70].

2.7.2.3 Operational Carbon Footprint

The *Operational Carbon Footprint* denotes the CO2 emitted when the system is running. The *Operational Carbon Footprint* can be computed using Equation (2.23).

$$\mathrm{C}_{op} = \mathrm{CI}_d \cdot E_{op} \qquad [gCO_2]_{S.I.} \tag{2.23}$$

where C_{op} *is the Operational Carbon Footprint,* CI_d *is the Carbon Intensity of the datacenter* $[gCO_2/kWh]_{S.I.}$*,* E_{op} *is the operational energy of the datacenter* $[kWh]_{S.I.}$ *[70].*

In this work, we employ simulation based on multiple models to predict the *Operational Carbon Footprint* of various configurations of datacenters, under distinct workloads and scenarios.

2.7.3 Metrics on quantifying the performance of the system

We identify performance as an increasingly concerning problem of nowadays LLM ecosystems, which is projected to have lead to an already-starting "modern-day Moore's Law" [6], which identifies a growing gap between the performance needs of LLMs and the actual performance of available physical infrastructure.

2.7.3.1 Latency

Latency is defined as the time delay between a cause and its observed effect, and measures *how long* an operation takes [99].

In this work, we quantify latency of LLM ecosystems by the amount of time required to process one token, or unit-scaled (e.g., million tokens). Thus, we measure latency in seconds.

2.7.3.2 Throughput

Throughput is defined as the rate at which a system completes operations, and measures *"how many"* operations (*"how much"* work) the system delivers within a given timeframe [99].

In this work, we quantify throughput of LLM ecosystems by measuring the amount of tokens processed in a per unit timeframe. Thus, we measure latency in tokens per second.

2.7.4 Metrics on quantifying the efficiency of the system

We identify efficiency metrics as crucial metrics for datacenter and LLM operators in making informed decisions about potential deployments, as this component offers a homogeneous comparison metric for each type of efficiency, directly comparable, simple to understand, represent, and explain.

2.7.4.1 Financial efficiency

We express financial efficiency as the cost per token per second, essentially for the monetary aspect of running LLM ecosystems at scale, in profit-driven processes. The financial efficiency is, thus, represented in *currency per token per second* e.g., *€/t/s*, *LEU/t/s*. Financial efficiency is computed as exemplified in Equation (2.24).

$$E_f = \frac{C}{T} = \frac{C}{\frac{T_p + T_d}{\Delta T_P + \Delta T_D}} = \frac{C \times (\Delta T_P + \Delta T_D)}{T_P + T_D} \tag{2.24}$$

Where E_f represents the financial efficiency, C represents the operational cost, T_P and T_D represent the amount of prefill and decode tokens, respectively, and ΔT_P and ΔT_D represent the total inference time for prefill and decode stages, respectively.

2.7.4.2 Sustainability efficiency

We express sustainability efficiency as the sustainability cost (e.g., energy, CO2 emissions) per token per second, essential to quantify and compare LLM ecosystems across the increasingly concerning problem of environmental sustainability. The sustainability efficiency is, thus, represented in *metric per token per second* e.g., *Wh/t/s*, *CO2/t/s*. Sustainability efficiency is computed as exemplified in Equation (2.25).

$$E_s = \frac{S}{T} = \frac{S}{\frac{T_p + T_d}{\Delta T_P + \Delta T_D}} = \frac{S \times (\Delta T_P + \Delta T_D)}{T_P + T_D} \tag{2.25}$$

Where E_s represents the sustainability efficiency, S represents the sustainability cost, T_P and T_D represent the amount of prefill and decode tokens, respectively, and ΔT_P and ΔT_D represent the total inference time for prefill and decode stages, respectively.

2.7.5 Metrics on quantifying the accuracy of the simulation

We quantify accuracy using the Mean Absolute Percentage Error (MAPE) ratio, also known as Mean Absolute Percentage Deviation (MAPD), and widely used in the field [68, 70, 100, 47, 101, 102]. MAPE equally penalizes positive and negative errors and is calculated using Equation (2.26), where n is the number of samples, R is real-world data, S is simulation data, i is the sample index [68]:

$$MAPE\ [\%] = \frac{1}{n} \sum_{i=0}^{n} \left| \frac{R_i - S_i}{R_i} \right| \times 100 \tag{2.26}$$

2.8 Discussion

In this chapter, we introduces the most important elements for this work. We adopted a top to bottom, conceptual to practical overview, starting from terminology and the conceptual concepts of simulation and reference architectures, continued with specifics of LLM ecosystems, caching, and simulating sustainability of ICT infrastructure, and concluded with community agreements on measuring various operational aspects of ICT infrastructure, LLM ecosystems, and simulation, through community standard metrics.

We identify several other important aspects, such as in-depth details of LLM inference, validation, experimental setup, and scientific methodology. However, we argue that, albeit overall relevant for this work, these topics are outside the scope of this chapter.

3

A Reference Architecture for LLM ecosystems

The lack of a reference architecture or a community-wide underlying conceptual model can be costly; conceptually, stakeholders (e.g., infrastructure operators, researchers) could overlook essential components, and even capable teams of researchers and engineers could tinker, leading to architectural and deployment challenges [15]. Furthermore, without a comprehensive reference architecture of LLM ecosystems, simulators of such infrastructure cannot be rigorously designed; later in this work (Chapter 4 - Chapter 6), we design, implement, integrate, and engineer a simulation instrument for KV-Caching system of LLM ecosystems under inference.

To rigorously design an ICT simulator, tailored to predicting LLM ecosystems, the state-of-the-art consists of materialising a reference architecture of the respective infrastructure, into a simulation tool or instrument, following scientific methodologies vetted and well-followed by the computer systems community [17, 14, 7, 15]. However, we argue that, currently, it doesn't exist no comprehensive reference architecture of LLM ecosystems under inference. This raises the research question: *(RQ1) How to synthesize and validate a reference architecture of LLM ecosystems?*

In this chapter, we address RQ1 by proposing the first reference architecture for LLM ecosystems under inference workloads, following community-vetted scientific processes for design and validation.

3.1 Overview

We propose the first reference architecture for LLM ecosystems under inference, following a distributed systems approach. Throughout this chapter, we match the state-of-the-art AtLarge Design Process [14]. Our contribution in this chapter is six-fold:

1. We define and establish design requirements and principles which guide in proposing the reference architecture (Section 3.2).
2. We synthesize a comprehensive reference architecture for LLM ecosystems under inference in Section 3.3, following the requirements and principles defined in Section 3.2. We model the entire interaction loop, from user input to system output, as a high-level picture, then we detail, individually, both the input and output process. Lastly, we detail the feedback loop, essential for processes of Reinforcement Learning from Human Feedback (RLHF).
3. We conduct a per-component analysis and discuss integration with other components of the ecosystem, providing real-life examples of employed technologies (Section 3.3).
4. We propose a detailed design for the KV-Caching system in Section 3.3.1. Albeit optional in LLM ecosystems, the KV-Caching sub-system can enhance, by orders of magnitude, the performance (e.g., throughput, latency) of the LLM ecosystems under inference.

5. We validate the reference architecture in Section 3.4; firstly, we validate the proposed architecture by aligning with the community-vetted, peer-reviewed Compute Continuum [16]; then, we align our proposed architecture with a domain-specific, industry existing LLM-inference ecosystem from OpenAI; then, we align the architecture with another state-of-the-art ecosystem, used by IBM for LLM inference; many thanks to IBM Research Europe for their many-fold contributions to open-science. Lastly, we present a high-level overview of how our proposed reference architecture aligns with real-world LLM ecosystems.
6. We address, in Section 3.5 each design requirement and principle presented in Section 3.2.

3.2 Design Requirements and Principles

In this section, we discuss the main design requirements and principles that guided our design process. We design to fulfill a set of requirements, corresponding to stakeholders of this reference architecture; we envision the main stakeholders of our work to be the researchers in the field of AI and LLMs, datacenter operators, C-level decision-making stakeholders, and students.

In [15], *Andreadis et al.* present a reference architecture for datacenter scheduling, a well-recognized scientific contribution published in SC18, the International Conference for High Performance Computing, Networking, Storage, and Analysis 2018. They define two main design requirements, *validity* and *usefulness*, also relevant for our reference architecture. Below, we expand on these **D**esign **R**equirements.

(DR1) **Ensure the *validity* of the reference architecture.**
Validity *"is the property of the proposed model to accurately represent the field of"* LLM systems [15]. The reference architecture should cover, for each component, the state-of-the-art from both industry and academia. Albeit fundamentally a subjective task quantifying the validity, we argue that mapping existing LLM systems to this reference architecture, and mapping the proposed reference architecture to the Compute Continuum prove the validity of this reference architecture to both abstract and align with real-world examples (i.e., real world LLM systems under inference) and align with higher level representations of the distributed ecosystems (i.e., the Compute Continuum).

(DR2) **Ensure the *usefulness* of the reference architecture**
Usefulness *"gives the reference architecture a real-world purpose which motivates its creation"*[15]. Alike *validity*, *usefulness* is a fundamentally subjective design requirement, yet we argue that usefulness can be evaluated by demonstrating the ability of the reference architecture to enable stakeholders to better reason about LLM system design in practice.

We further derive seven **D**esign **P**rinciples the reference architecture should follow to ensure a comprehensive, actionable, and future-proof architecture, contributing to an end goal of simulating and digitally twinning, holistically, accurately and robustly, the Compute Continuum.

(DP1) **Design components with clear distinct responsibilities.**
We regard distinct-responsibility components as essential for designing a state-of-the-art reference architectures, especially following community standards [15, 103]. Each system component should have its own set of responsibilities, defined boundaries of those responsibilities, and a set of interfaces which define its services to other components [15]. Albeit included in the reference architecture, we acknowledge that not all the components need to be used by every real-world stakeholder of LLM systems; e.g., while essential in system performance, KV-Caching can be omitted in LLM systems, making the system still functional, yet non-performant.

(DP2) **Group related components.**
Corresponding to best practices for packaging components [103], and following vetted design processes of reference architectures [15], related components should be grouped according to their responsibility. We acknowledge this introduces a degree of subjectivity and, thus, we regard also other reference architectures for LLM systems, although potentially not matching the same structure as the RA from this work proposes.

(**DP3**) **Aim for extensibility and modularity**
The architecture should be modular, corresponding to the best practices of designing software systems [103], thus allowing for independent extension, detailing, analysis, or replacement of components. We regard this as a critical design principle to ensure a future-proof property of the proposed reference architecture. Furthermore, such extensibility and modularity allow the reference architecture to seamlessly integrate and integrate with emerging technologies that will emerge, without redesigning the whole.

(**DP4**) **Separate mechanisms from policies and goals.**
Each architectural component should clearly distinguish between mechanisms (how it operates), policies (how decisions are made), and the metrics used in measurement and evaluation (e.g., latency, throughput, power draw, CO2 emissions). To provide the necessary [15] level of abstraction, the reference architecture should follow a qualitative model [103], without mandating specific policies [15].

(**DP5**) **Cover end-to-end prompt-to-response LLM workflow.**
The architecture should model the entire inference workflow, from the user input to the system output, and model all the intermediate stages, with the necessary level of abstraction. This end-to-end exhaustive view ensures a continuous interaction-feedback loop, and ensures that, e.g., bottlenecks, cross-component effects, and trade-offs can be analyzed and eventually simulated.

(**DP6**) **Support multiple users in the ecosystem.**
The architecture should support multiple users operating simultaneously, each giving workloads (i.e., prompts) to the LLM ecosystem, and each receiving responses, following the end-to-end prompt-to-response format proposed in (**DP5**).

(**DP7**) **Model components responsible for decision processes across the system.**
The architecture should represent decision-making processes that occur across each layer of the continuum. This includes (pre)processing, workload and resource allocation, inference management, and overall orchestration and monitoring.

(**DP8**) **Model different types of prompts execution workflow.**
The architecture should model different types of prompt-workflow and consider at least workflows with prompt reasoning, where each prompt generates a subsequent prompt, and workflows with branching prompts. This design principle allows for modelling a wide range of prompts, tailored to today's standards in LLMs and with future developments.

3.3 Overview of the Reference Architecture

In this section, we present an overview of the proposed reference architecture, present the overall design and workflow, which covers an end-to-end prompt-response-feedback loop, and the main responsibilities of each tier, component, and mechanism. Figure 3.1 presents the proposed reference architecture.

Following design principle (**DP1**), we propose a reference architecture of an ecosystem which leverages sets of components, distributed by tier of operation and grouped where related (**DP2**), with each layer and component abstracted as modular and extensible (**DP3**). We model all three tiers of the continuum [16], namely *endpoint*, *edge*, and *cloud*, and attach related components to the specific tier (**DP1**), (**DP2**).

Endpoint: The proposed reference architecture begins with the user's prompt and finalizes with the ecosystem's response, following an end-to-end prompt-to-response workflow (**DP5**); optionally, the user can offer feedback on the LLM's response, feedback further used for fine-tuning the LLM and for better tailoring responses to the user's preferences.

Edge: We propose a multi-user ecosystem, where each user-given workloads (i.e., prompts) are preprocessed in the edge, and forwarded to the most suitable cloud infrastructure, following decision processes (**DP7**), and the Machine Learning as a Service (MLaaS) operational model.

Cloud: Once in the cloud, each prompt is preprocessed and managed by an in-cluster workload manager

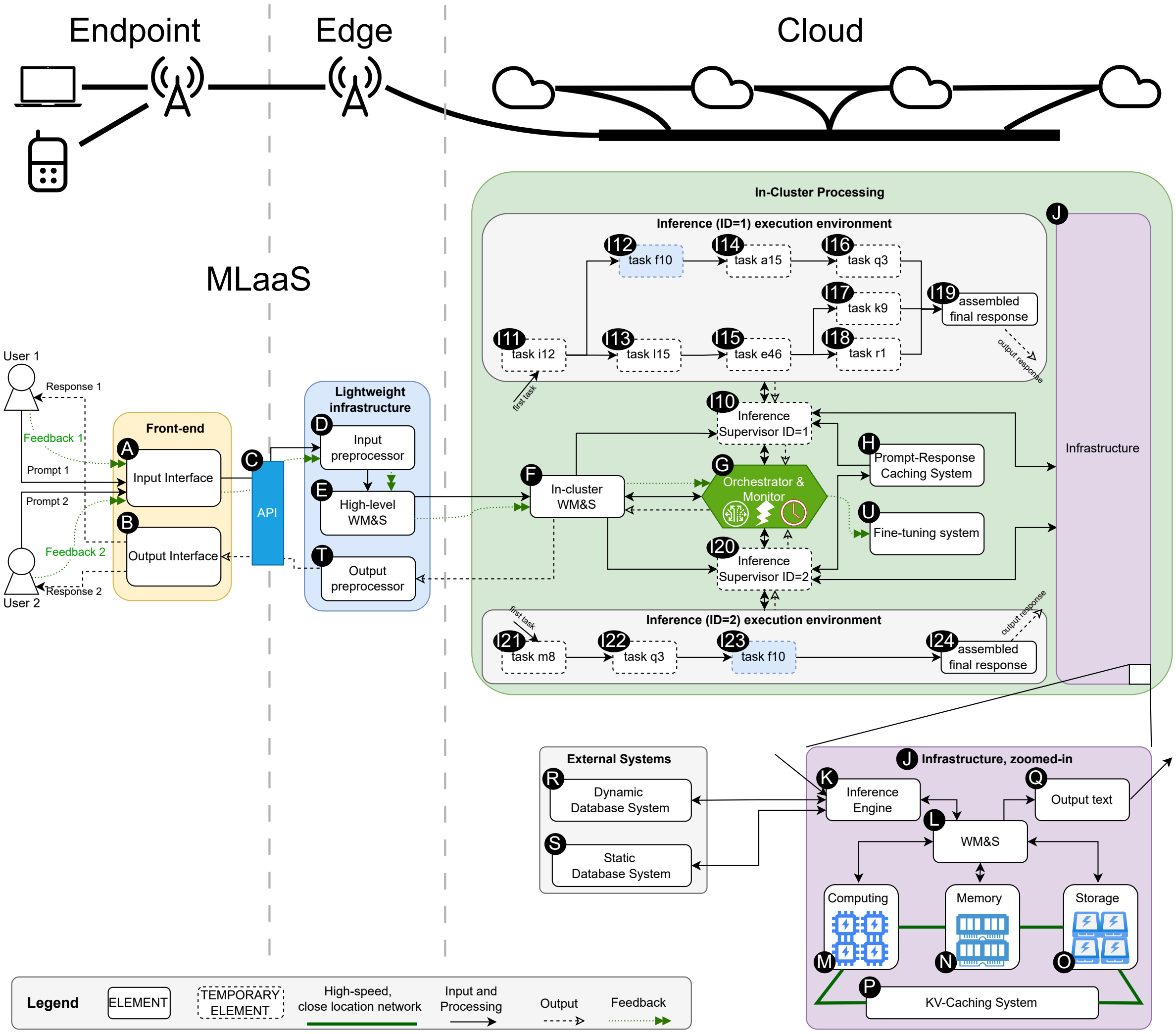

Figure 3.1: Reference Architecture of LLM ecosystems under multi-user inference workload.

and scheduler, a cloud decision-making component which ensures fair resource allocation per prompt, per user (**DP4**) (**DP6**) (**DP7**). We represent two inference processes, run in parallel, one with a complex, branching, reasoning workflow (inference id 1), and the other with a sequential reasoning workflow (inference id 2), both coupled to a shared caching system of prompts and responses (**DP8**). All processes executing in the cloud are orchestrated and monitored for performance, energy consumption, and failure detection, ensuring quality of service (QoS) and meeting service level objectives (SLOs) (**DR1**)-(**DR2**), (**DP1**)-(**DP8**).

Infrastructure: We now focus on component J. The workload enters the infrastructure through the *Inference Engine* K, which communicates with the *WM&S* L and orchestrates the computation, the backbone of the LLM inference process. We abstract the computation process as shared between three types of physical infrastructure, namely *Computing Units* M, *Memory Racks* N, and *Storage Units* O, all of them employing techniques and linked to a *System* P; the physical infrastructure is interconnected by a high-speed, close-location network (e.g., InfiniBand).

External Systems: The infrastructure is linked with *External Systems*, which expand the LLM capabilities beyond the isolated functionality an independent and disconnected LLM could give. Component S, the *Static*

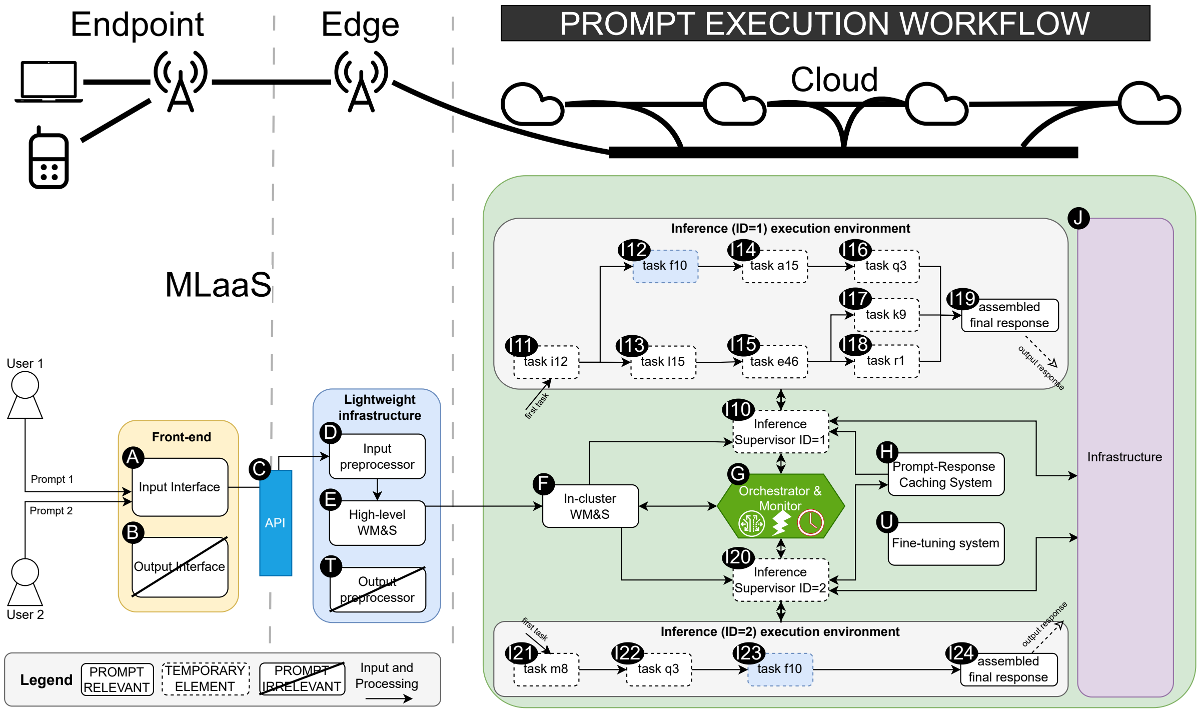

Figure 3.2: Reference Architecture of LLM ecosystems detailing the prompt execution workflow. The prompt execution workflow focuses on components from A to J.

Database System, allows the LLM to access local, in-cloud databases, which store, index, and manage embedding vectors used for similarity search and retrieval augmented generation (RAG) [104]. Component R, the *Dynamic Database System* enables the LLM to conduct web searches [105]. Both the Dynamic and the Static database systems are orchestrated by the *Inference Engine* K.

Workflows: We identify and detail three main workflows (**DP5**): *(i) in Section 3.3.1*, we detail the prompt-execution workflow, containing the steps between the user's input and the response of the LLM; *(ii) in Section 3.3.2*, we detail the response workflow, containing the processes between the LLM response and the display of the response on the user's interface; *(iii) in Section 3.3.3*, we detail the optional feedback workflow, through which users can review the LLM's response; further feedback is passed to the ecosystem for LLM fine-tuning and prompt enhancement.

Note In the following sections, we detail one workflow per section, detailing specifics of each workflow, and de-focusing non-relevant components for better visual comprehension. In Section 3.3.1 we detail the prompt execution workflow, in Section 3.3.2 we detail the prompt response workflow, and in Section 3.3.3 we detail the feedback workflow.

3.3.1 Detailed Design of Prompt Execution Workflow

In this section, we detail the *prompt execution workflow*, which starts with the user prompt and ends with the final leveraged response. Figure 3.2 illustrates this process.

3.3.1.1 Front-end (endpoint) tier

The prompt execution workflow begins with the users, who provide input (prompts), visually represented in the left-most part of the reference architecture from Figure 3.2. Although we present only two users for

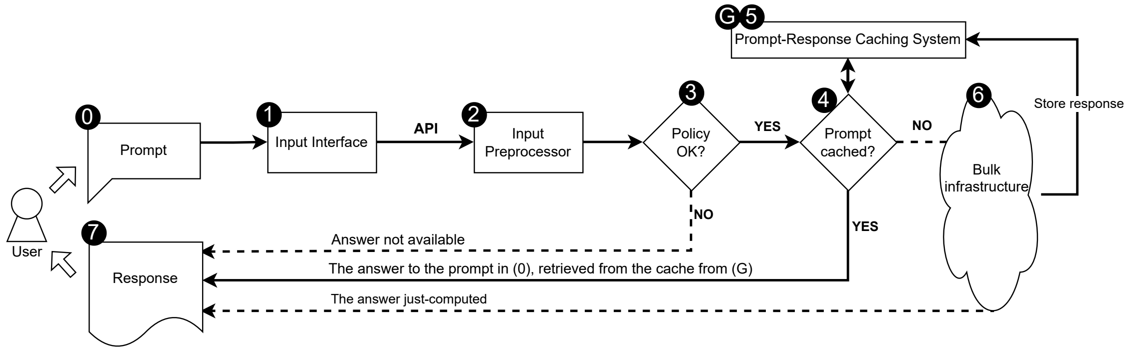

Figure 3.3: Prompt-response workflow, preventing redundant computation of already-generated responses to prompts, employing an inference, prompt-level caching system.

visual purposes, the proposed RA can scale indefinitely from a design perspective, yet is upper-bounded by limitations of physical resources.

Users interact with the ecosystem via an *input interface* 🅐, as part of the *front-end*, which is often through a web interface or through a mobile application, but could also be through an API call via e.g., a command line interface (CLI) environment. The *front-end* tier is the equivalent to the *endpoint* tier from the RA of the Compute Continuum [16]; we further expand and align our RA with the Compute Continuum in Section 3.4.3.

3.3.1.2 Lightweight infrastructure (edge) tier

Following the MLaaS operational model, the user's prompt is transferred to the *cloud* where the heavyweight computation happens; however, in this transfer process, the edge plays a crucial role.

The user's prompt is transferred from the *front-end* to the *lightweight infrastructure* via an *application programming interface (API)* 🅒 and is further parsed by an *input preprocessor* 🅓. We argue that the overall process of data preprocessing (mainly happening in 🅒) is critical in reducing the amount of data transferred from the edge to the cloud, and in restricting the execution prompts to only desired (e.g., policy-adherent) prompts.

3.3.1.3 A detailed design on the Prompt-Response Caching System

In Figure 3.3, we present a prompt-response workflow that prevents the re-computation of prompts the ecosystem has already responded to, thus minimizing redundancy. Although not essential for the system's base functionality, the illustrated caching approach offers a theoretical advantage by restricting the execution of prompt workflows to only when encountering a non-cached prompt. OpenAI employs a prompt-caching system that considers only prompts exceeding 1,024 tokens [11]. OpenAI claims to perform only *"halfway prompt caching,"* where they store prefill weights in the caching system and always run the decode stage for each cache hit, for each user: *"the actual response is computed anew each time based on the cached prompt."* [11]. The OpenAI codebase is closed-source.

In Figure 3.3, aligned with the overall RA presented in Figure 3.1 and Figure 3.2, the process begins with the *user's prompt* ⓿, through an *input interface* ❶, and further parsed through an *input preprocessor* ❷. In the preprocessing step, the system checks if the prompt matches the company's policy and laws in the country of operation (e.g., prompt content, lawfulness) ❸. If the prompt complies, the system checks whether the prompt is already cached ❹ in a *prompt-response caching system* ❺; the *prompt-response caching system* is equivalent to element 🅖 from Figure 3.1. This process is mainly handled by (i) the *High-Level WM&S* from the *lightweight infrastructure* (🅓, Figure 3.2), (ii) the *In-Cluster WM&S* (🅖, Figure 3.2), and (iii) the *Orchestrator and Monitor* (🅕, Figure 3.2), all decision-making components of their respective layer (**DP7**).

If the prompt is cached, the already generated and cached response is retrieved and delivered to the user.

We envision this approach as improving performance, boosting the ecosystem's throughput, reducing the system's latency, and overall reducing the amount of resources consumed by the infrastructure (e.g., power draw), with the caveat that the caching system is well-designed and efficient. The response retrieval process doesn't consume more resources than generating the response itself. If the prompt-response pair is not found in the caching system, the system forwards the workload to a bulk infrastructure which generates a response; this response is ultimately stored in the *prompt-response caching* system G/5, and offered to the user 7.

In practice, OpenAI employs a similar technique; however, they claim to store only prefill weights in a caching system, and for each cache hit, the decode stage is rerun for each user. With this halfway caching technique, OpenAI claims to have OpenAI claims this technique to have reduced latency *"by up to 80% and cost by 50% for long prompts"* [11].

3.3.1.4 In-cluster (cloud) tier

Orchestrator & Monitor: The bulk computation and storage occur in the cloud, a massive-scale, highly heterogeneous, and distributed infrastructure, where a core, central component supervises the overall process, the *Orchestrator & Monitor* G. Component G serves as a core decision-making component and manages, using data obtained from monitoring the ecosystem, other supervisors of the ecosystem, e.g., *inference supervisor* I10, I20, *in-cluster WM&S* F. The monitoring responsibility involves measuring performance metrics (e.g., throughput, latency), sustainability metrics (e.g., the amount of hourly emitted CO2, energy consumption), and system failures (e.g., uptime, amount of jobs completed/failed). Based on the metrics gathered from the monitoring process, *Orchestrator and Monitor* G analyses and predicts further behaviour (e.g., using an ecosystem simulator), and makes decisions in orchestrating inference workloads. Furthermore, the *Orchestrator and Monitor* G has management and supervision access over the part of the cloud dedicated to the ecosystem.

Edge-cloud WM&S communication: The *high-level WM&S* E, from the edge, forwards prompts to the *in-cluster WM&S* F, which manages prompts and ensures fair, policy-compliant responses to users' requests. F generates inference supervisors for each prompt received; in Figure 3.2, we exemplify using two prompts, each with its own inference supervisor, namely supervisors I10, I20.

The inference supervisor: serves as a middleware between the workload tasks and the overall datacenter orchestrator & monitor F. For example, *inference supervisor 1*, represented in the upper half of Figure 3.2, handles the inference process for prompt 1, from user 1.

Complex, branching, prompt: We exemplify a prompt that requires a reasoning process that branches, where each task generates one or more new tasks, until a final response is obtained. In the illustrated reference architecture, the *inference supervisor* firstly generates a starting task I11, which, after completion, generates two new tasks I12 and I13. This recurrent process recurrently repeats and, depending on the task, one or more tasks are generated until a *final response* is assembled, in our example I19. The *inference supervisor* determines when a response is final.

Simple, sequential, prompt: However, prompts can also generate simpler, sequential, and non-branching tasks, such as prompt 2, assigned inference supervisor 2 and the tasks I21-I24.

Prompt-response caching system: For each generated task (I11-I19, I21-I24), the workload is managed by the corresponding inference supervisor, which checks the *prompt-response caching system* G. If the response to the specific task[1] is found in the *prompt-response caching system* (cache hit), then the *inference supervisor* retrieves and uses the response as the response to the specific task. Otherwise, when the response to the prompt is not already cached (cache miss), the inference supervisor forwards the workload to the infrastructure J. The infrastructure J computes and redirects the response to the caching system H;

[1] in nowadays ecosystems, e.g., OpenAI's ChatGPT [106] or Google's Gemini [92], each answered task is temporary saved and also phrased as a prompt for the future task(s).

here, if the caching policy is matched, the response is saved. Further, the response is forwarded to the inference process, either as an intermediate response or as an assembled final response (e.g., I19, I24).

Identical tasks: In Figure 3.2, we exemplify two prompts with an identical intermediate task, namely *task f10* (I12, I23). This could happen in user prompts with similar tasks, and, thus, an identical intermediate task to achieve the response to a certain task. We exemplify below with two analogies, one with a Mathematical analogy and one with a Path(Route)-Finding example.

Analogies of identical tasks: In the *Mathematical Analogy (Listing 3.1)*, we showcase a scenario where two users give two distinct prompts, yet with an identical sub-task (i.e., computing 10 factorial, equivalent to 10!, where $10! = 10 \times 9 \times 8 \cdots \times 2 \times 1$); this task happens only once, for the first encountered prompt, and is retrieved for the second prompt, instead of redundantly re-computed.

In the *Path-Finding Analogy (Listing 3.2)*, similarly, two users give two distinct prompts. The first user requests a path from Amsterdam to Bucharest, and the LLM ecosystem, unable to identify the response to such a prompt in the cache, computes, generates, and caches the response to each intermediate task. The second user requests a path from Amsterdam to Bratislava; the LLM finds this path as cached, and only retrieves the response from the *prompt-response caching system*, without redundant (re)computation. Then, the LLM ecosystem only computes the rest of the response.

```
Task 1: Calculate 12 x 6 x 1980 x 10!
Task 2: Calculate 23 x 2 x 2004 x 10!

LLM approach:
Step 1.1: Determine 12 x 6 x 1980. Not cached. Compute. Cache.
Step 1.2: Determine 10!. Not cached. Compute. Cache.
Step 1.3: Solve final task 1. Not cached. Compute. Cache.

Step 2.1: Determine 23 x 2 x 2004 and cache. Not cached. Compute. Cache.
Step 2.2: Determine 10!. Cached! Retrieve!
Step 2.3: Solve final task 2. Not cached. Compute. Cache.
```

Listing 3.1: A Mathematical analogy of the LLM inference and prompt-response caching process.

```
Task 1: Find a path from Amsterdam to Bucharest for a motorbike drive.
Task 2: Find a path from Amsterdam to Bratislava for a motorbike drive.

LLM approach:
Step 1.1: Find intermediate checkpoints for the most time-efficient route between Amsterdam and Bucharest
(e.g., LLM finds Amsterdam, Leipzig, Bratislava, Arad, Bucharest). Not cached. Compute. Cache.
Step 1.2: Find the most efficient route Amsterdam - Leipzig. Not cached. Compute. Cache.
Step 1.3: Find the most efficient route Leipzig - Bratislava. Not cached. Compute. Cache.
Step 1.4: Find the most efficient route Brastislava - Arad. Not cached. Compute. Cache.
Step 1.5: Find the most efficient route Arad-Bucharest. Not cached. Compute. Cache.
Step 1.6: Generate and export GPX file to user. Not cached. Compute. Cache.

Step 2.1: Find intermediate checkpoints for the most time-efficient route between Amsterdam and Bratislava
(e.g., LLM finds Amsterdam, Leipzig, Bratislava). Not cached. Compute. Cache.
Step 2.2: Find the most efficient route Amsterdam - Leipzig. Cached! Retrieve!
Step 2.3: Find the most efficient route Leipig - Bratislava. Cached! Retrieve!
Step 2.4: Generate and export GPX file to user. (Partially) Cached! Retrieve! Compute the rest. Cache.
```

Listing 3.2: A Path-Finding analogy of the LLM inference and prompt-response caching process.

3.3.2 Detailed Design of the LLM Response Workflow

In this section, we detail the *prompt-response workflow*, which begins once the final response is assembled and finalised, and ends once the response is displayed on the user's interface Figure 3.4 illustrates this process.

Cloud: The response output process begins from I19, for inference with ID=1, and from I24, for inference with ID=2. The output response is further transferred to the corresponding *inference supervisor*, further transferred to the *in-cluster WM&S* F. The entire workflow executed in the cluster is constantly monitored by the *orchestrator and monitored component* G.

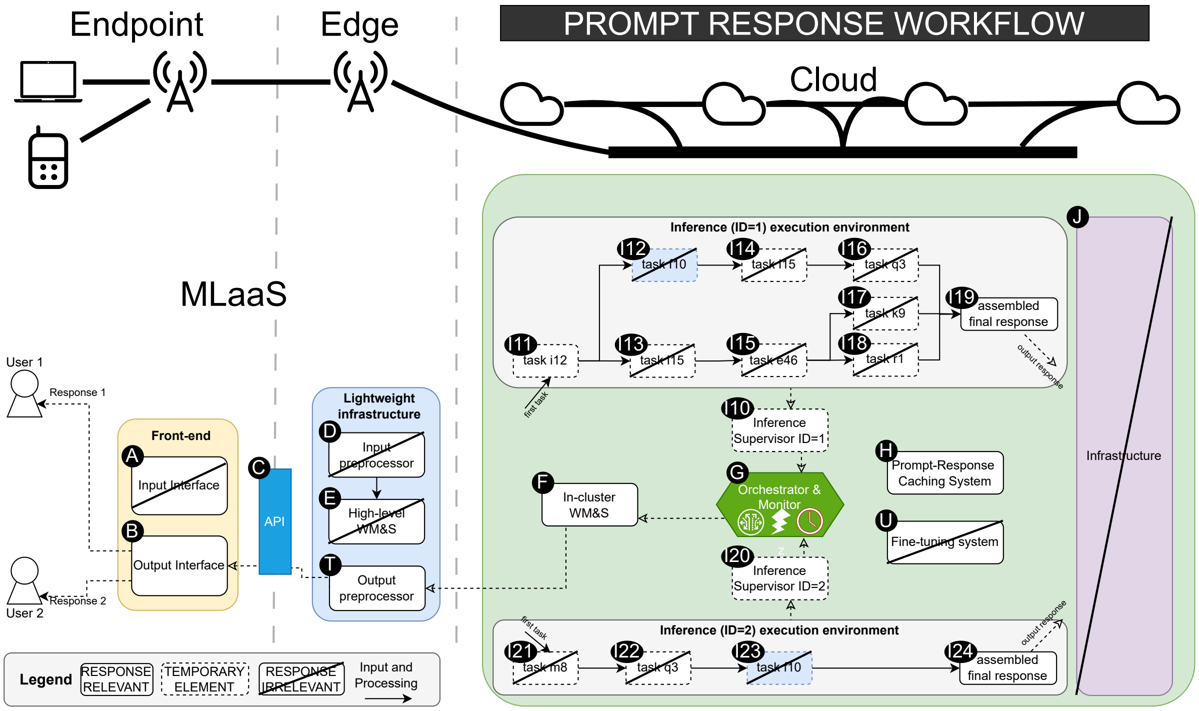

Figure 3.4: Reference Architecture of LLM ecosystems detailing the LLM response processing workflow. The prompt-response workflow focuses on components I19, I24, I10, I20, H, G, F, T, C, and B.

Edge: The leveraged response, now located in F, the WM&S of the cloud, is transferred to the *output processor* T from the *edge* (i.e., *lightweight infrastructure*).

Endpoint: Lastly, the response is transferred from the *edge*, via the API, to the *endpoint* and displayed on the *output interface* B of the LLM ecosystem, belonging to the *front-end*.

3.3.3 Detailed Design of Feedback Workflow

In this section, we detail the *feedback processing workflow*, an optional workflow of the inference process which users often skip, yet is critical for LLM finetuning and tailoring responses to users' needs and preferences [107, 108]. This workflow starts with the user's feedback, through the input interface and ends with the system receivng and optionally adopting this feedback. Figure 3.5 illustrates this process.

Why feedback matters: Feedback is crucial for refining LLM responses and improving performance through Reinforcement Learning from Human Feedback (RLHF). For example, OpenAI researchers identify even small RLHF-trained models, of 1.3B parameters, as better-preferred, higher-accurate, than larger models such as 175B GPT-3, although having 100x fewer parameters [108]. While limited reports exist from large LLM providers, RLHF is a widely used technique, cheap to scale compared to traditional LLM finetuning or extended training, preventing and reducing financial, performance, and sustainability costs [39, 109, 107, 108].

Endpoint: We model a scenario in which both user 1 and user 2 evaluate the LLM's prompt. The feedback detail depends on the platform, and can vary from e.g., selecting between positive and negative to e.g., giving detailed feedback on multiple categories, with written components. Users' feedbacks are inputted via component A, the *Input Interface*, and transferred to the cloud via the endpoint, similarly to users' prompts.

Edge: The *API* C links the *Endpoint* to the *Edge* (and vice versa), transferring the user's feedback to an *Input preprocessor* D. Depending on the platform design, D can filter users' text feedback and check policy adherence; if the feedback doesn't involve a text component, step D may be skipped. Further, a *High-level*

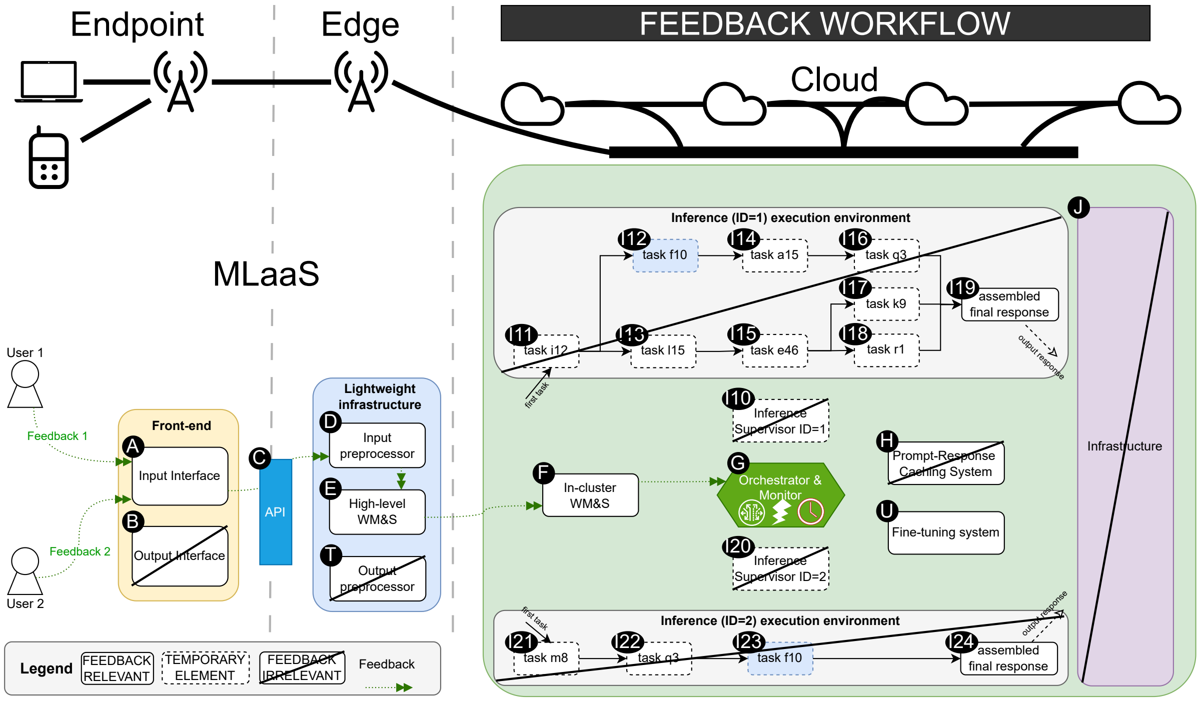

Figure 3.5: Reference Architecture of LLM ecosystems detailing the feedback processing workflow. The feedback workflow focuses on components Ⓐ, Ⓒ, Ⓓ, Ⓔ, Ⓕ, Ⓖ, and Ⓤ.

WM&S Ⓔ component forwards the feedback to the cloud.

Cloud: The feedback is processed by an *in-cluster WM&S* Ⓕ, which forwards user's feedback to an *Orchestrator & Monitor* Ⓖ component, a centric element of the cloud with monitoring, analysis, and decision capabilities over the datacenter part reserved for the inference process of the LLM ecosystem. The feedback is forwarded to a *Fine-Tuning System* Ⓤ which handles the feedback.

Feedback policies: How the ecosystem handles users' feedback is dependent on the provider's policies and regulations. For example, an LLM provider can choose to tailor LLM's responses only for the conversation in progress, without keeping cached feedback for other conversations, and without using users' feedback for fine-tuning the global LLM for other users. However, a provider with less strict privacy policies can use feedback for fine-tuning the model for all users, not only for the specific user who gave the feedback. To ensure generality and universality of the proposed reference architecture, we abstract the feedback and fine-tuning system into a unitary component Ⓤ.

Note The process described throughout this section (Section 3.3), and the last few pages, executes within (milli)seconds in real-world ecosystems [51, 6, 2].

3.4 Mapping Real-World LLM Inference Ecosystems to the Reference Architecture

Reference architectures are most useful when they accurately depict real-world instances [15]. In this section, we align our reference architecture with industry-leading LLM ecosystems and with a peer-reviewed, community-standard reference architecture of the Compute Continuum.

We identify some components as non-disclosed; non-disclosed components are components that are likely to

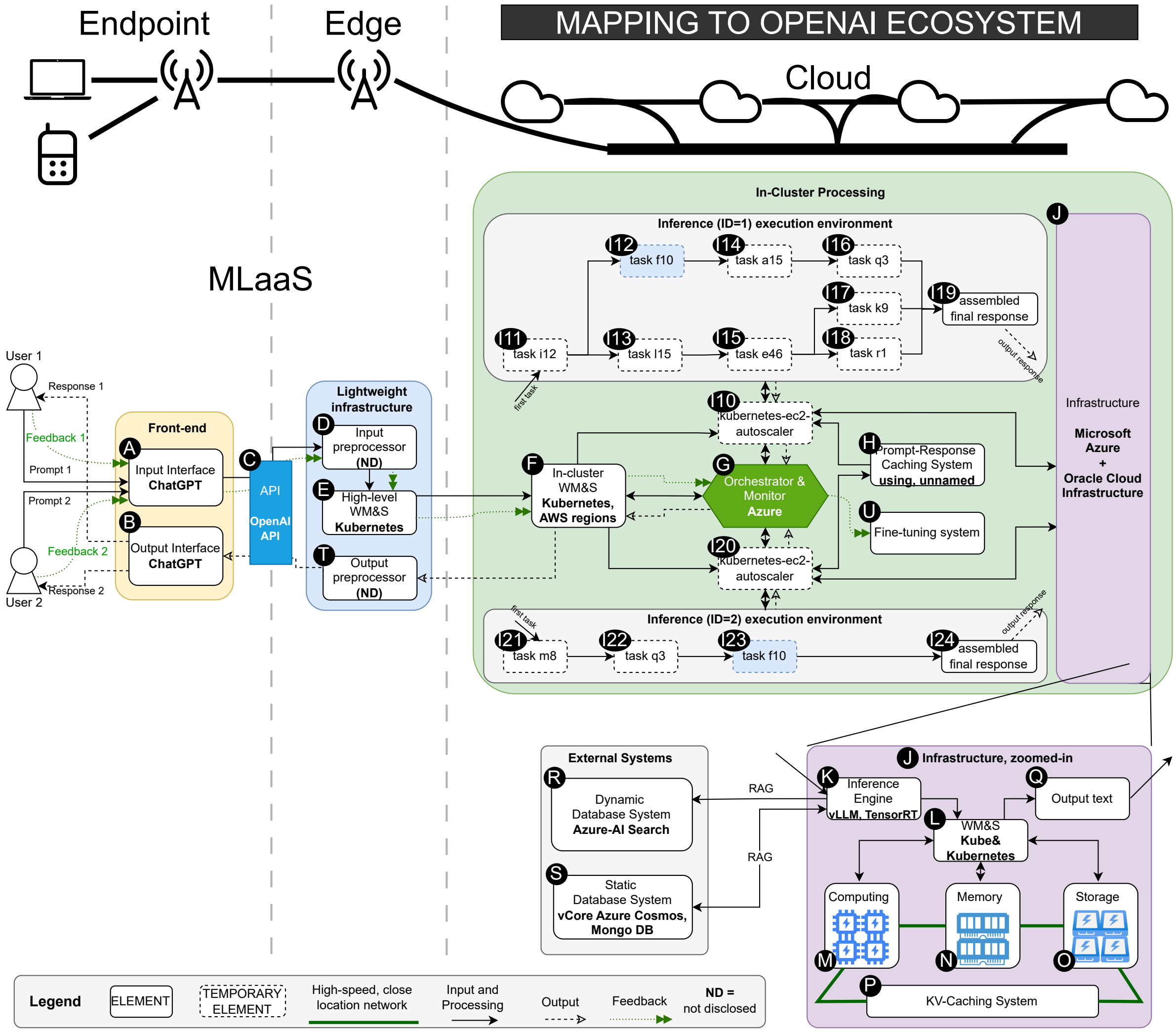

Figure 3.6: OpenAI LLM Inference Ecosystem mapped to the reference architecture.

exist in real-world (deployed) LLM-inference systems, but their presence is not disclosed by some designers and operators, and only inferred by communities of practice, e.g., on sites such as Hacker News and Reddit, or disclosed by other designers and operators. For example, OpenAI does not disclose the usage of input preprocessor, but the Ubicloud-envisioned ecosystem and the Databricks ecosystem disclose they use Llama Guard [110], and Databricks Guardrails [111], respectively, to serve this component.

3.4.1 Alignment with OpenAI LLM inference ecosystem

In this section, we validate the proposed reference architecture by mapping to it the OpenAI Ecosystem for LLM inference; we present the alignment in Figure 3.6. While OpenAI doesn't explicitly present a reference architecture of its ecosystem, as of May 2025, the company discloses technologies through publicly released web articles. However, several components remain non-disclosed, represented in Figure 3.6 as *ND*.

Endpoint: Users interact with the ecosystem via a front-end component with an input Ⓐ and output Ⓑ interface, such as the ChatGPT mobile application or website.

Edge: The *endpoint* communicates with the *edge* via the OpenAI API [112] Ⓒ. We note that OpenAI does not disclose information on the *input preprocessor* component D. OpenAI utilizes Kubernetes E at the edge as a high-level workload manager and scheduler, which redirects workloads to the appropriate cloud [113].

Cloud: Overall, OpenAI is highly reliant on Microsoft Azure services for system monitoring, orchestration, and management G, physical infrastructure J, and external systems R [114]. OpenAI's Kubernetes implementation, Kubernetes-ec2-autoscaler, addresses bursty and unpredictable workloads that can scale from single-machine operations to hundreds of cores. The Kubernetes-ec2-autoscaler is a batch-optimized scaling manager and maps to I10 and I20 in our reference architecture [113]. The resource autoscaling approach has been explored in recent literature and has proven its effectiveness in improving performance and SLO adherence; for example, Chiron is a hierarchical autoscaler for LLM-inference, which can enhance SLO attainment by 90% and GPU efficiency by up to 70% compared to its absence [115].

Prompt-Response Caching: OpenAI employs prompt caching, which is claimed to reduce latency by 80% and cost by 50% for long (more than 1,024 tokens) prompts [11], H. The system checks if the prefix of the prompt is stored in the cache, and if a matching prefix is found, the system uses the cache's result. Alternatively, the system processes the full prompt. OpenAI keeps caches active for 5-10 minutes and up to 1 hour during off-peak periods [11].

Infrastructure: The infrastructure, component J, orchestrates computational, memory, and storage for LLM inference in the *Cloud* tier. In 2016, OpenAI was mostly using *"TensorFlow (or Theano) for GPU computing; for CPU, we* (note: OpenAI) *use those or Numpy"* [113]. While the exact technologies used by the inference engine are undisclosed in 2025, we argue that OpenAI follows the community standard of employing vLLM, TensorRT, or similar technologies[116, 117, 118, 119]. For *workload management and scheduling* L, OpenAI uses Kubernetes customized for heavy ML workloads and scaled to managing thousands of nodes: in 2018, OpenAI was running 2,500 nodes, while in 2021, OpenAI was running 7,500 nodes [120]; exact numbers are undisclosed for 2025.

Critical to performance, OpenAI employs KV-Caching P, which prevents the redundant computation of the attention mechanism. Although the KV-Cache implementation remains undisclosed, OpenAI's approach proves to reduce latency by 80% and half the costs. Similarly, the exact infrastructure of OpenAI is undisclosed in 2025; we expect major hardware, middleware, and software advances in the upcoming years as a response to the significant funding announced for OpenAI (e.g., potential $500 billion from Stargate [121], $40 billion from Softbank [122]).

External systems: To access information available online (e.g., news) without retraining the model at a financially, computationally, and environmentally unsustainable granularity, OpenAI uses Azure AI Search, formerly Azure Cognitive Search, an *"information retrieval system for your heterogenous content"* [114], which we map to component R, the *Dynamic Database System.* As a *Static Database System*, OpenAI leverages Azure Cosmos DB, which allows for retrieval-augmented generation capabilities and stores frequently requested information as cached responses. S helps in reducing latency and costs by minimizing real-time web access R through pre-indexed content [123].

3.4.2 Alignment with IBM LLM inference ecosystem

In this section, we validate the proposed reference architecture by mapping it to the IBM Ecosystem for LLM inference and present the alignment in Figure 3.7. Although IBM does not exhaustively disclose technologies used in its reference architecture, as of 2025, IBM releases to the public and open science significantly more information than OpenAI. We validate the alignment of our reference architecture with OpenAI's ecosystem in Section 3.4.1.

IBM inference stack widely employs WatsonX, "*a portfolio of AI products that accelerates the impact of generative AI in core workflows to drive productivity*"[124].

Endpoint: IBM implements the endpoint component through Watson Assistant, with a simplistic and performant interface for input A and output B [125].

Edge: Following the MLaaS model, the *endpoint* connects to the *edge* through an API component; Watsonx API enables this functionality as a core component of the Watson stack, and maps to component C

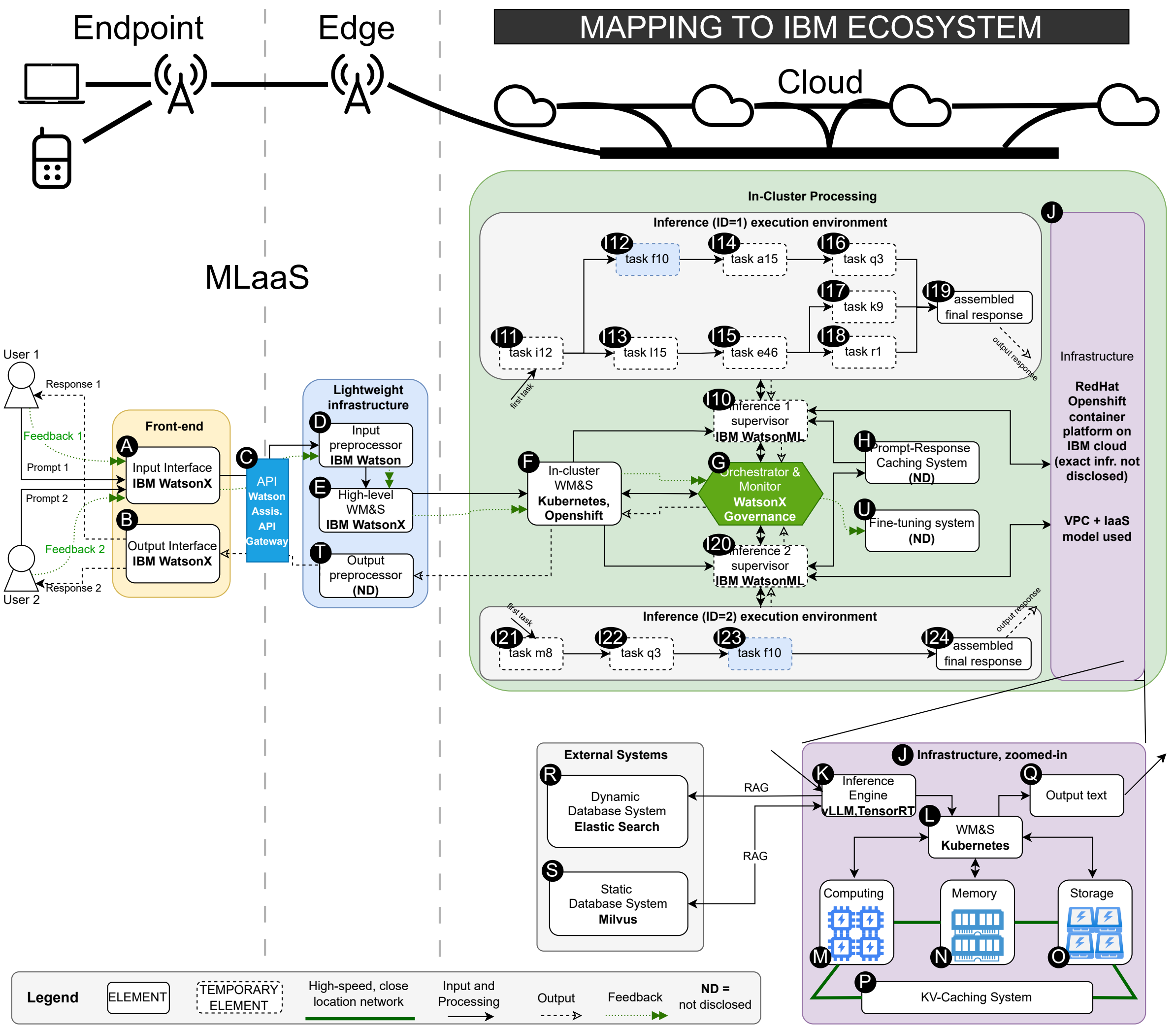

Figure 3.7: Reference architecture aligned with IBM LLM inference ecosystems.

from the proposed RA [125]. Once the user's input reaches the *edge*, Watson Assistant's natural language understanding processes and filters the prompt for policy compliance, formatting, and enhancement Ⓓ. This component, together with the *High-Level WM&S* Ⓔ, decides to route the customer's request to *"the appropriate resolution mechanism, which might be an action or a search of existing content"* [126].

Cloud: IBM relies on RedHat's OpenShift, which provisions and manages container images, workloads, and inference processes underlying Kubernetes Ⓕ. In direct communication with OpenShift, Watson Governance handles the overall system orchestration and monitoring for performance and cost Ⓖ [127]. Component Ⓘ maps to Watson Machine Learning services, which handle and supervise inference processes [128].

Prompt-Response Caching: While not explicitly disclosed, we argue that, similarly to OpenAI, IBM employs a *prompt-response caching system* Ⓗ, enabling prompt caching through Watson Assistant's conversation memory and action-based storage mechanisms [125].

Infrastructure: IBM implements infrastructure through IBM Virtual Private Cloud (VPC) [129]. Watsonx.ai handles the core LLM inference process; the Inference Engine Ⓚ follows a multi-framework approach and supports TensorFlow, PyTorch, as well as vLLM for KV-Caching Ⓟ [129]; IBM's infrastructure adopts

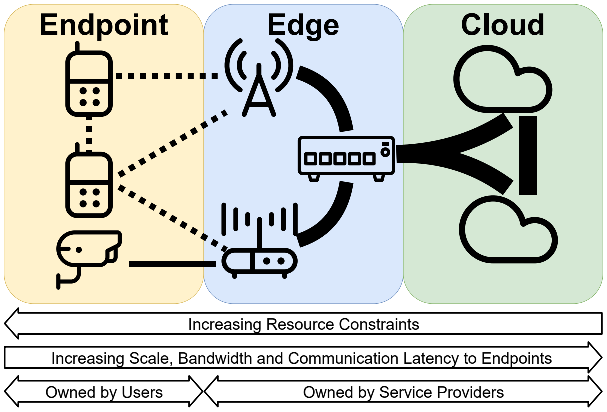
Endpoint
Edge
Cloud
Increasing Resource Constraints
Increasing Scale, Bandwidth and Communication Latency to Endpoints
Owned by Users
Owned by Service Providers

Figure 3.8: A high-level taken with permission from the Compute Continuum [16].

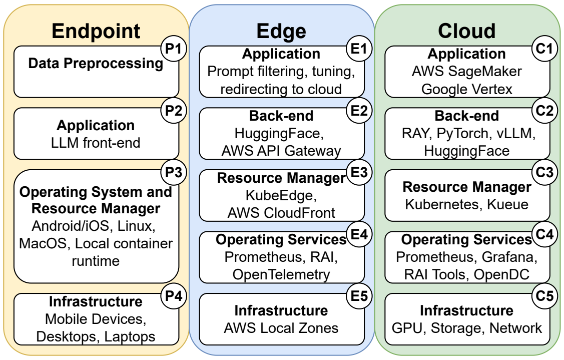
Endpoint
Data Preprocessing
P1
Application
LLM front-end
P2
Operating System and Resource Manager
Android/iOS, Linux, MacOS, Local container runtime
P3
Infrastructure
Mobile Devices, Desktops, Laptops
P4
Edge
Application
Prompt filtering, tuning, redirecting to cloud
E1
Back-end
HuggingFace, AWS API Gateway
E2
Resource Manager
KubeEdge, AWS CloudFront
E3
Operating Services
Prometheus, RAI, OpenTelemetry
E4
Infrastructure
AWS Local Zones
E5
Cloud
Application
AWS SageMaker Google Vertex
C1
Back-end
RAY, PyTorch, vLLM, HuggingFace
C2
Resource Manager
Kubernetes, Kueue
C3
Operating Services
Prometheus, Grafana, RAI Tools, OpenDC
C4
Infrastructure
GPU, Storage, Network
C5

Figure 3.9: LLM ecosystems architecture for inference workflows aligned with the Compute Continuum.

RedHat's Openshift Container Platform for workload management and determining the *"the optimal node in the cluster is for each pod to run on"* [130].

External systems: IBM implements component R, *Dynamic Database System*, through Elasticsearch, an IBM-provided service, coupled with Watsonx Assistant, with RAG capabilities and the ability to access web resources [131]. For component S, *Static Database System*, IBM uses Milvus within WatsonX Data to store precomputed embeddings, enable efficient similarity searches, and thus reduce real-time query load by up to 40% [132, 133]. The external systems are linked to the *Inference Engine* K and to the *Orchestrator and Monitor* G.

3.4.3 Alignment with the Compute Continuum

We align our RA with the Compute Continuum, a peer-reviewed reference architecture proposed by *Jansen et al.* and a community standard. In Figure 3.8, we present a high-level view of the continuum, comprising three main tiers: the endpoint, the edge, and the cloud. These pivotal elements align with the tiers outlined in the reference architecture we propose in Section 3.1.

Endpoint: The LLM inference process begins from the endpoint, where users give prompts via the LLM front-end P2. The main endpoint infrastructure is represented by regular user devices, such as mobile devices, desktops, or laptops P4, leveraging the operating system and resource manager of the device itself P3, yet the endpoint could also be accessed via APIs (e.g., OpenAI API). The endpoint infrastructure redirects the workload to the edge.

Edge: Running *large* language models *at scale* on the edge becomes unmanageable when demand grows [118, 134]; to address the growing demand of running large-scale LLMs (over 10B parameters, [118]) at nowadays massive-scale public of users, LLM services use the edge for redirecting the workload to the most suitable datacenter capable of providing the best combination of performance, financial, sustainability metrics [135, 119, 118]. The *edge tier* is responsible for prompt processing at the *application level* E1, where prompts are filtered for matching the service's policy, optionally system-enhanced (prompt tuning), and further redirected to the cloud. The *back-end component* E2, e.g., HuggingFace, provides lightweight AI analysis and contributes to AI-powered decision-making processes (e.g., routing to datacenters, NLP for analyzing prompts). At the same time, AWS API Gateway handles secure communication with both the *endpoint tier* and the *cloud tier*. While Edge doesn't conduct the core inference process, it still contains *resource managers* E3 for orchestrating edge devices (e.g., KubeEdge) and for in-edge workload distribution (e.g., AWS CloudFront). *Operating Services* E4 monitor, collect performance metrics (Prometheus), ensure responsible AI governance (RAI), and collect telemetry data for troubleshooting, debugging, and application management (OpenTelemetry).

Cloud: Once the *edge back-end* E2 redirects the prompt to the *cloud tier*, potentially in the form of a workload for execution, the *application* layer C1 enables inference processes under the available resources, using technologies such as AWS SageMaker or Google Vertex. C3, the *resource manager*, could employ the Kubernetes-Kueue tuple, where Kubernetes orchestrates containerized workload across the infrastructure, and Kueue ensures fair workload scheduling. Similarly to the *edge tier*, yet on a larger scale, *operating services* (e.g., C4) collect (e.g., Prometheus) and visualize (e.g., Grafana) datacenter metrics and ensure responsible AI governance (e.g., RAI). We envision future research in the field of digital twinning, where state-of-the-art simulators (e.g., OpenDC) serve as key decision-making tools in a simulation-infrastructure adjustment-simulation loop. The *back-end* C2 enables distributed computing across the infrastructure (with tools such as Ray), offer deep learning capabilities (with PyTorch as the state-of-the-art as of 2025 standards), and employs KV-Caching tools and techniques for enhanced performance (e.g., vLLM). The infrastructure is primarily composed of GPUs, typically A100/H100, storage systems, which are generally multi-layered for caching, temporary, and bulk storage purposes, and high-bandwidth networking, such as InfiniBand.

Cloud - Edge - Endpoint: Once the final task response is obtained, the *cloud tier* forwards the response to the *edge tier*, which forwards to the *endpoint tier*, to which the user has access via the LLM front-end. Depending on the LLM ecosystem and service, this process can happen in a single step, where the entire response is offered to the user at once (e.g., Google's Gemini) [136], or in multiple steps, where the response is sequentially offered to the user in intermediate steps (e.g., OpenAI's ChatGPT) [106].

3.4.4 Multi-Ecosystem validation

In this section, we present a high-level validation overview and compare our proposed reference architecture with real-world ecosystems for serving LLM inference.

OpenAI: OpenAI is one of the largest (perhaps, the largest) LLM providers as of 2025. In Section 3.4.1, we validated our proposed reference architecture against the ecosystem OpenAI uses to serve LLM inference. Although OpenAI does not disclose some of the components we include in our reference architecture, OpenAI still releases sufficient information for our validation. We identify that OpenAI mainly uses ChatGPT, Azure, Oracle, and vLLM; we also identify that components of our reference architecture closely match the components OpenAI use in their LLM inference ecosystem.

IBM: IBM is one of the largest LLM providers as of 2025. In Section 3.4.2, we validated our proposed reference architecture against the ecosystem IBM uses to serve LLM inference; IBM releases to the public large amounts of information on how they deploy and operate LLM ecosystems, although not fully disclosing all components (e.g., prompt-response caching system). We identify that IBM mainly relies on Watson, OpenShift, and vLLM, and the components of our reference architecture closely match the components IBM discloses as used for LLM inference.

Ubicloud: Ubicloud offers an open-source alternative to cloud providers like AWS, Azure, and Google Cloud [75]. Ubicloud envisions an LLM Inference stack and publishes details of this ecosystem through their engineering blog [75]. Ubicloud LLM service is EuroGPT [110], which runs Meta's Llama 3.1 405B model on European infrastructure and uses Llama Guard for prompt moderation [139]. In the overview of LLM ecosystems from Table 3.1, we provide a high-level overview of LLM inference technologies from Ubicloud [75]. We identify the Ubicloud ecosystem as mainly leveraging EuroGPT, Llama-3.1 405B [145], vLLM, and Kubernetes; our reference architecture closely matches the components of the ecosystem Ubicloud proposes.

Databricks: Databricks is one of the largest (perhaps the largest) AI and data lakehouse platforms as of 2025. Databricks offers services for serving AI, including serving LLM inference, and provides extensive information about their AI/LLM inference stack through their online documentation [142, 138, 111, 143]. In the overview of LLM ecosystems from Table 3.1, we align the components from our reference architecture with real-world components from the Databricks ecosystems. We identify that Databricks serves LLM inference mainly through MosaicAI, a *"platform for building, evaluating, deploying, and monitoring generative AI ap-*

Table 3.1: Overview of the proposed reference architecture against real-world ecosystems for serving LLM inference. U/U = using, but the component is unnamed, N/D = not disclosed.

ID	Component Name	IBM Ecosystem	OpenAI Ecosystem	Ubicloud	Databricks
A	Input Interface	WatsonX [125]	ChatGPT [106]	EuroGPT [110]	Databricks Notebooks [137]
B	Output Interface	WatsonX [125]	ChatGPT [106]	EuroGPT [110]	Databricks Notebooks [137]
C	API	WatsonAssistant [126]	OpenAI API [112]	Ubicloud API [75]	MosaicAI Serving [138]
D	Input Preprocessor	Watson [126]	N/D	Llama Guard [139]	Databricks Guardrails [111]
E	High-level WMS	WatsonX [126]	Kubernetes [113]	Scheduler,unnamed [140]	N/D
F	In-cluster WMS	Kubernetes, Openshift	Kubernetes, AWS [113]	Scheduler,unnamed [140]	Kubernetes [141]
G	Orchestrator Monitor	WatsonX Governance [127]	Azure [114]	EngineCore [140]	MosaicAI [138]
H	Prompt-Response Caching System	N/D	U/U [11]	N/D	N/D
I	Inference Supervisor	WatsonML [128]	kubernetes-ec2-autoscaler [113]	AsyncLLM [75]	MosaicAI [138]
J	Infrastructure	Openshift [130]	Azure, Oracle Cloud Infrastructure [114]	Ubicloud [75]	Databricks (U/U)
K	Inference Engine	vLLM, TensorRT [129]	N/D, LLM, TensorRT [116, 117, 118, 119]	vLLM [75]	TensorRT, TensorFlow [142]
L	Infrastructure WMS	Kubernetes	Kubernetes [120]	Cloud Hypervisor [110]	Kubernetes [141]
M	Computing Unit	U/U	U/U	U/U	U/U
N	Memory Racks	U/U	U/U	U/U	U/U
O	Storage Racks	U/U	U/U	U/U	U/U
P	KV-Caching	U/U	U/U	U/U	U/U
Q	Output text	U/U	U/U	U/U	U/U
R	Dynamic Database	Elastic Search [131]	Azure AI Search [114]	Lantern [139]	Databricks Vector Search [143]
S	Static Database	Milvus [132, 133]	vCore Azure Cosmos, Mongo DB [123]	PostgreSQL [139]	Delta Lake [144]
T	Output Preprocessor	N/D	N/D	Llama Guard [139]	Databricks Guardrails [111]

plications (gen AI apps)" [146], coupled with NVIDIA's TensorRT and TensorFlow for serving inference [142]. Moreover, Databricks uses their in-house build system for input/output filtering, Databricks Guardrails [111], and Databricks Vector Search and Delta Lake for dynamic and static database systems [143, 144]. Overall, we identify that the components from our reference architecture closely match (abstractise) the real-world components from the Databricks ecosystem.

3.5 Requirement Validation

In this subsection, we present how the proposed reference architecture addresses each design requirement and principle detailed in Section 3.2.

(DR1) Ensure the *validity* of the reference architecture.

We regard our reference architecture as situated in the middle of the spectrum, "low-high-level.", as a mid-level conceptual model.

Firstly, we align with *two mid-low-level* LLM ecosystems, used by the largest LLM providers of 2025, OpenAI and IBM. We validate our proposed reference architecture by mapping each component to elements of the real-world equivalent ecosystems. We identify that some components may not be disclosed by one provider, but are disclosed by the other provider (e.g., the prompt-response caching system). Similarly, we identify that some components are not recognized or distinguished as components. Still, the released documentation acknowledges the existence of such functionality and system (e.g., the input preprocessor from the edge).

Secondly, we align our reference architecture with a *high-level conceptual model* of the ICT Compute Continuum, a scientific, peer-reviewed, and community-standard model for conceptualizing the ICT field.

Thus, we align our reference architecture across three gradual steps on the low-to-high-level spectrum, encompassing both the industry and academic worlds. We, therefore, argue that we have proven the validity of our proposed reference architecture.

(**DR2**) **Ensure the *usefulness* of the reference architecture.**
We identify and address the design requirement of *usefulness*, defined and subjectively quantified by the *"real-world purpose which motivates its* (note: the reference architecture's) *creation."* We motivate the need for a reference architecture which would aid various groups of stakeholders, especially decision-making LLM/infrastructure operators, researchers, and students, in offering a high-level picture of the ecosystem, with high-level components which otherwise could be omitted or misinterpreted even by experienced groups [147].

(**DP1**) **Design components with clear distinct responsibilities.**
We identify and address the design principle of clearly distinguishing components with distinct responsibilities and, thus, adhering to the community standards for defining reference architectures. We distinguish components by two layers of abstraction. From a high-level perspective, we identify three main distinct large-components (*tiers*): the *front-end*, the *lightweight infrastructure*, and the *datacenter(s)* (heavyweight, massive-scale infrastructure). In other words, we identify the *endpoint*, the *edge*, and the *cloud*. From a lower-level perspective, yet still high enough to offer abstraction, we identify components for each tier, each with its own specific responsibility and scope. We detail in Section 3.3 how components interact and make the inference process of the LLM ecosystem tick.

(**DP2**) **Group related components.**
We identify and address the design principle of grouping related components according to their responsibility. We identify the main tiers, namely (i) front-end, (ii) lightweight infrastructure, and (iii) the datacenter, and identify the main components for each tier. (i) For the *front-end tier*, we identify the input and output interface, with which the user interacts and we, thus, group together. (ii) In the *lightweight infrastructure*, we identify components for preprocessing prompts, conducting workload management and scheduling, and outputting the response to the user; we identify a main common responsibility for each element in the lightweight infrastructure (edge) of being a middleware between the endpoint and the cloud. (iii) For the tier of *in-cluster processing* (i.e., cloud, datacenter), we group elements directly contributing to inference management, execution, and supervision. Furthermore, for each inference process, we group the inference supervisor with the corresponding set of (sub)tasks it executes (e.g., I10 supervisor grouped with I11-I15).

(**DP3**) **Aim for extensibility and modularity.**
We identify the design principle of extensibility and modularity by designing the RA in accordance with community-vetted standards for RAs [16, 15], adapted from software architecture practices [103]. Each component of the architecture contains a degree of abstraction sufficient to both understand the purpose of a specific component (e.g., infrastructure, orchestrator and monitor) and to be replaced or extended. For example, component G, the *Prompt-Response Caching System*, allows future stakeholders to extend, detail, or even skip entirely, based on individual needs and visions on the LLM ecosystem. This design principle is crucial for the lifespan and adoption of the proposed reference architecture.

(**DP4**) **Separate mechanisms from policies and goals.**
We identify and address the design principle of distinguishing between mechanisms, policies, and metrics. Corroborated with (**DP1**), we design each component as single-purpose, and separate mechanism elements, answering *how it operates*, (e.g., *input interface*, *prompt-response caching system*), from policy elements *how decisions are made* (e.g., *in-cluster WM&S*), from goals/metrics elements (e.g., *orchestrator and monitor*). Consequently, systems such as the *infrastructure* or *prompt-response*

caching system contain both mechanisms and policies, yet without diminishing the validity and adherence to (**DP4**) or (**DP1**).

(**DP5**) **Cover end-to-end prompt-to-response LLM workflow.**
We identify and address the design principle of modeling the complete feedback loop, from the user input to the system output. Specifically, this is represented by an outgoing arrow from the user to the *input interface*, and an incoming arrow from the *output interface* to the user. In Section 3.3.1, we detail the execution phase, spanning the workflow from the *user's prompt* to the *assembled final response*, and in Section 3.3.2, we detail the response phase, spanning the workflow from *the assembled final response* to its delivery on the *output interface*.

(**DP6**) **Support multiple users in the ecosystem.**
We identify and address the LLM ecosystem-specific design principle of designing a reference architecture which accommodates multiple users interacting simultaneously with the ecosystem. For better comprehension purposes, we visually represent only two users, yet present and detail the inference process for a many-user interaction (Section 3.1). We emphasize the role of each component responsible for the multi-user operability, especially the role of the *high-level WM&S* from the lightweight infrastructure, the *in-cluster WM&S*, the *prompt-response caching system*, and the *orchestrator and monitor*.

(**DP7**) **Model components responsible for decision processes across the system.**
We identify and address the LLM ecosystem-specific design principle of employing components responsible for decision-making processes, such as the workload manager and schedules D, E, the *orchestrator and monitor* F, central to the cloud infrastructure, the and the *inference supervisors* I10, I20. These decision-making components span both *lightweight infrastructure* and *datacenter* tiers, responsible for management and supervision across different layers of the ecosystem.

(**DP8**) **Model different types of prompts execution workflow.**
We identify and address the design principle of modelling flexible prompts by presenting two different inference execution processes, one of which contains branching prompts that ultimately aggregate in a final response, and one which contains no branching, but only reasoning. For simplicity and visual comprehension, we model only two examples, yet still covering the distinct execution behaviour inference processes have..

3.6 Discussion

We summarize contributions of this chapter, envision future research, and discuss potential threats to validity.

Summary: In this chapter, we propose and validate the first comprehensive reference architecture for LLM ecosystems under inference, following the vetted AtLarge Design Process [14], and addressing RQ1. We validate this reference architecture by aligning with the Compute Continuum [16], with a state-of-the-art ecosystem from IBM, and with an LLM inference ecosystem from OpenAI. The proposed reference architecture, combined with a design focused on long lifespan, enables LLM operators, researchers, and students to gain a better understanding of the ecosystem and, where applicable, make more informed decisions.

Future Research: We envision future research in modeling LLM ecosystems. LLM training is a resource-very-hungry [3, 39], computationally intensive [39, 40, 5], financially expensive [5, 36], and sustainability-concerning [148, 18, 19]; similarly, to the inference process, currently there is no comprehensive reference architecture of LLM ecosystems under the training phase. We also envision future work in modeling and simulating the inference process with different types of prompts, e.g., prompts with no intermediate tasks, deep research prompts [149], and prompts with intermediate tasks (reasoning).

Threats to Validity: The reference architecture proposed in this chapter has been designed in accordance with a well-established set of design requirements and principles, utilizing state-of-the-art design and validation methodologies, and leveraging resources and knowledge from the open-source and open-science communities.

While comprehensive and universal, we cannot guarantee full-spectrum validity regarding alignment with existing closed-source ecosystems. However, we expect those ecosystems to follow a similar (if not identical) reference architecture. Offering an analogy from physics, we can completely validate or invalidate only theories and principles applicable to the known universe; there is no theory or principle of physics for which we can guarantee it holds in the unknown universe. Similarly, there is no reference architecture for which one can guarantee its validity in the *"unknown universe."*

4

Design of Kavier: a tool for simulating LLM inference and KV-Caching

LLM ecosystems are becoming increasingly large, distributed, and heterogeneous, and raise performance, sustainability, and efficiency concerns [10, 86, 53, 3]. It is crucial to understand how LLM ecosystems, and the (eco)systems orchestrated by LLM ecosystems, operate and behave at scale. Addressing this concern, in Chapter 3, we proposed the first comprehensive reference architectures of LLM ecosystems under inference, which provide a vital conceptual overview of the LLM continuum. We envision simulation as a natural next step for systematically anticipating how LLM ecosystems would behave under different workloads and configurations; simulation enables experimentation and prediction of performance, sustainability, and efficiency in a time and cost-efficient way [7, 1].

Designing a simulator capable of cache-awarely predicting the performance, sustainability, and efficiency of LLM ecosystems under inference, using discrete-event simulation, is a critical yet non-trivial scientific challenge. Currently, there is no such scientific instrument. This raises the research question: *(RQ2) How to design Kavier, a scientific instrument for cache-aware simulation analysis of the performance, sustainability, and efficiency of LLM ecosystems under inference?*

In this chapter, we address the RQ2 by designing Kavier, a first-of-its-kind scientific instrument for cache-aware simulation of the performance, sustainability, and efficiency of LLM ecosystems under inference. In Chapter 5, we leverage this design to create an implemented prototype of the simulator, further integrated with a top-tier, community-vetted, and peer-reviewed data center simulator. In Chapter 6, we validate our design through real-world, trace-based experimentation.

4.1 Overview

We design Kavier matching the state-of-the-art AtLarge design process of designing computer systems and ecosystems [14]. Our contribution in this chapter is seven-fold:

1. We define and establish functional and non-functional requirements for Kavier in Section 4.2.
2. We propose a high-level design for the architecture of Kavier in Section 4.3.
3. We present models Kavier uses to simulate the performance of LLM inference in Section 4.5. These models simulate LLMs under various caching policies, both for KV-Caching and prefix matching.
4. We present models Kavier uses for predicting sustainability in Section 4.6.
5. We present models Kavier uses for computing efficiency metrics, namely performance-cost and sustainability-cost in Section 4.7.
6. We address each functional and non-functional requirement in Section 4.8.

7. Lastly, we reflect on our design and envision future work in designing simulation instruments for LLM ecosystems Section 4.9.

4.2 Requirements Analysis

In this section, we establish a set of functional requirements (FRs) and non-functional requirements (NFRs) that guide the design process of Kavier, a tool for simulating LLM ecosystems under inference, with a focus on the caching component. This matches stage 1 of the AtLarge Design Process on Distributed Systems and Ecosystems [14].

Main Functional Requirement (MFR): Simulate performance, sustainability, and efficiency of LLM ecosystems under inference.

4.2.1 Functional Requirements

We identify a set of six functional requirements which guide our design process and tell *"what the system should be able to do"* [46].

(FR1) Support holistic simulation of the LLM inference process.
The simulator should model the entire LLM inference process executed in the cloud tier, both the *prefill* and the *decode* stage. Kavier should support splitting the inference process between these two stages and tailor it accordingly to the different performance characteristics of each stage. Furthermore, Kavier should be a discrete-event simulator with a user-configurable prediction granularity. Without (**FR1**), Kavier would omit the distinct behaviours of prefill and decode and, thus, cannot accurately simulate the performance of LLM workloads.

(FR2) Simulate with cache awareness.
Kavier, as a cache-oriented simulator for LLM inference, should support the simulation of the key-value caching (KV-Caching) mechanism used in transformer models. Kavier should allow for enabling or disabling KV-Caching for the simulation scenario, thus enabling comprehensive modelling of real-world LLM processes and facilitating versatile experimentation. Without (**FR2**), the tool cannot explore impacts of caching policies on performance (**FR3**), sustainability (**FR4**), and efficiency (**FR5**).

(FR3) Predict the performance of the LLM ecosystem under workload.
The simulator should predict ecosystem performance, specifically, predict latency and throughput. We identify latency as the amount of time required to answer a prompt; we identify throughput as the number of tokens that can be executed per second. We consider a sequential execution of prompts and identify prompt parallelisation as an area of future research in the simulation of LLM inference. Results should be recorded in a structured trace format, both as a task-based trace, containing cumulated trace details, and as a fragment-based trace, detailing snapshots of each task, snapshots taken at a user-established granularity. Without (**FR3**), Kavier would not provide insight into performance metrics, thus limiting a further accurate evaluation of sustainability and efficiency metrics.

(FR4) Predict the sustainability of LLM ecosystems under workload.
The simulator should predict the ecosystem's sustainability, using models for estimating the energy consumption of the GPU infrastructure and the resulting CO2 emissions for the simulated workload. Kavier, coupled with a peer-reviewed simulator, should predict power usage over time, following the user-established granularity, and the total energy consumption run by a batch of LLM inference workloads. Addressing the increasingly concerning CO2 emissions for large-scale and massive infrastructure, Kavier should predict the carbon footprint of LLM inference workloads, using real-world CO2 traces. Without (**FR4**), the simulator cannot assess the sustainability impact of LLM inference, a pivotal concern in a digital world with increasingly overexploited resources.

(FR5) Predict the efficiency of LLM ecosystems under workload.
The simulator should predict efficiency metrics of the ecosystem, using a configurable financial model

and the performance and sustainability metrics expanded in (**FR3**) and (**FR4**), respectively. Kavier should allow for performance-cost metrics, estimating financial cost per token per second. Kavier should also allow for performance-sustainability metrics, estimating sustainability cost per token per second. (**FR5**) thus enables a clear and direct comparison between ecosystems, aiding stakeholders in making informed decisions when deploying, maintaining, and expanding LLM ecosystems.

(**FR6**) **Design Kavier compatible with other simulators and extensible.**
The simulator should allow simple integration with a peer-reviewed datacenter simulation framework, and be designed following modularity principles. Kavier's output should align with the datacenter simulator's input formats (e.g., input traces, experiment setup). Kavier should also be designed as modular, and further engineered strictly following this design approach, thus aligning with state-of-the-art software architecture and design principles [103]; this functional requirement is crucial for ensuring long software life and allowing for adding future functionality. Without (**FR6**), Kavier would have limited usefulness as a universally applicable simulator, hindering its adoption and evolution as part of a datacenter simulator.

4.2.2 Non-Functional Requirements

In addition to the set of functional requirements aforementioned, we identify four non-functional requirements, which guide the design and engineering process of Kavier, and tell us *"how well the features should work"* [46]. We address non-functional requirements at implementation and integration time, in Section 5.5.

(**NFR1**) **Provide in-meeting, near-interactive, same-day simulation results.**
Cloud infrastructure currently operates at an unprecedented scale [69, 47]. The system should run efficiently, output the simulation results promptly, and support predictions of very large-scale batches of tasks. Simulating system performance should take less than 1% of the actual run of the experiment, for prompts with prefill and decode times larger than 10 seconds cumulatively. For example, if a batch of 1,000 prompts, each 10 seconds long, would take 10,000 seconds in total, Kavier should offer predictions in a matter of 1-2 minutes. Similarly, we identify the requirement of selecting a fast and efficient, peer-reviewed datacenter simulator that can predict system sustainability rapidly. However, although relevant for the speed of the overall system's performance, optimizing external simulators (e.g., the datacenter simulator) is beyond the scope of this paper. Without (**NFR1**), the Kavier cannot be reasonably used in interactive settings or for large-scale batches of LLM workloads.

(**NFR2**) **Aim to provide adequate simulation accuracy.**
The predictions produced by Kavier should be on par with reality and within a Mean Absolute Error Ratio (MAPE) margin of 10%. MAPE penalizes overestimates and underestimates equally throughout a series of predictions, making it suitable for quantifying the accuracy of discrete-event simulations. The timing predictions should be calibrated against empirical data traced from real-world systems. Without (**NFR2**), the instrument would give unreliable insights. We later validate Kavier's accuracy of prefill and decode time, through trace-based experiments in Chapter 6.

(**NFR3**) **Facilitate reproducibility and open science.**
The results produced by Kavier should be fully reproducible, and Kavier should be built and released in accordance with open-science principles. The code, configuration, and experiment traces should be made available, thus adhering to principles of open source and open science. Simulation involving randomness should be controllable via seeds to ensure perfect reproducibility of the experiment. Kavier should be released with rigorous documentation and tutorials for usage. We regard (**NFR3**) as a critical requirement for Kavier to be considered a real and valuable contribution to the scientific community.

(**NFR4**) **Adhere to modern software design and development standards.**
The simulator's codebase must be maintainable and adaptable for future changes in LLM systems. The codebase should contain clean code and adherence to software engineering best practices, such as modularity, clarity, and tests. The system should not only be to this work but should integrate

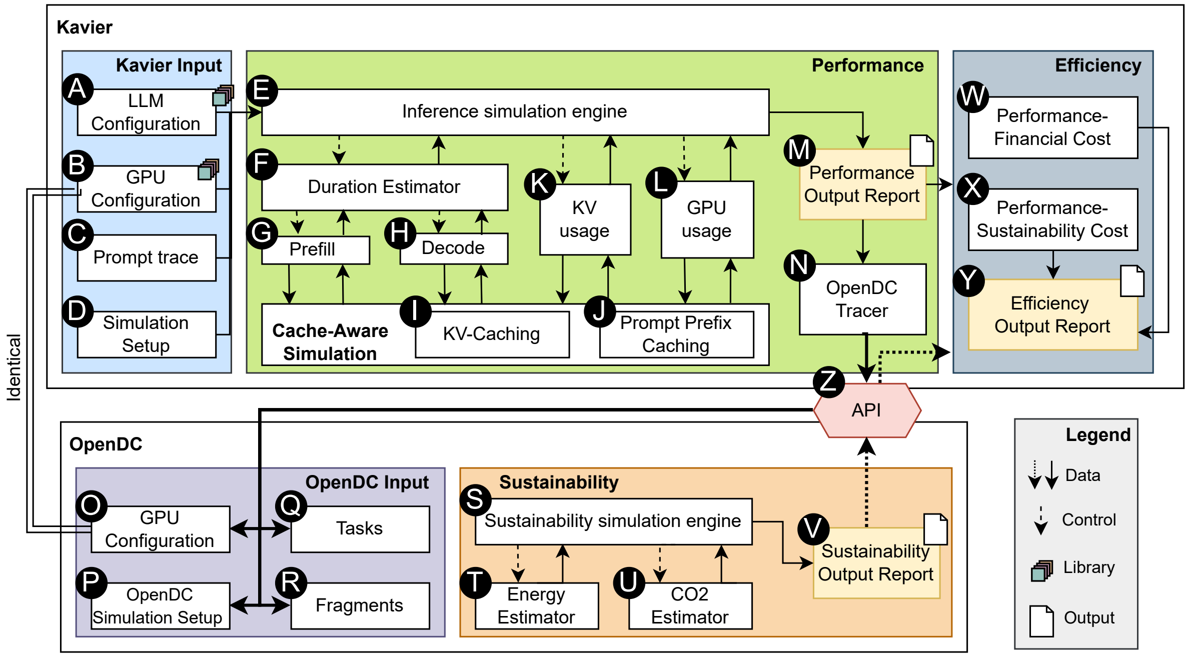

Figure 4.1: Overview of the high-level architecture of Kavier and OpenDC.

with a peer-reviewed datacenter simulator, evolve, and adapt to future engineering. Without (**NFR4**), Kavier's future development and maintenance would be unsustainable in the long run.

By meeting the above functional and non-functional requirements, Kavier would serve as a KV-Cache-aware LLM inference simulator, providing accuracy, efficiency, speed, and utility for various stakeholder groups, with a long software lifecycle and simplicity in expansion by future contributors.

4.3 Overview of Kavier

In this section, we present a high-level overview of Kavier as coupled with OpenDC, a state-of-the-art, peer-reviewed simulator. The holistic simulation infrastructure follows a discrete-event simulation model (**FR1**), and predicts performance (**FR3**), sustainability (**FR4**), and efficiency (**FR5**) of both small and massive-scale batches of LLM inference workloads. Kavier integrates with OpenDC, an open-source, peer-reviewed, and state-of-the-art simulation framework for datacenters, with over 8 years of development, operation, and constant contributions to the scientific community [7, 78, 68, 70, 69, 17]. Following the AtLarge Design Process [14], we design, implement, and validate Kavier iteratively; this process begins with bootstrapping the creative process (stage 3), then focusing on the high-level and low-level design (stage 4) [69, 14].

4.3.1 Design Choices

In this section, we analyze design choices of the high-level architecture of Kavier. We identify three main classes of analysis: type of simulation (e.g., discrete-event, continuous, or total), integration with other simulation tools, and simulation pipeline. We now discuss each alternative.

(DC1) Discrete-event simulation: We identify three main simulation models: discrete-event simulation, continous-simulation, and total simulation. We identify discrete-event simulation as the most suitable for addressing the **MFR**, as this simulation model enables both post- and during-prediction analysis, unlike total simulation, which offers only overall results without providing insights into how the ecosystem evolves [7, 60]. We also identify continuous simulation model which, however, does not align with LLM inference operational model, which involves discrete-events (e.g., token generation, cache hits/misses) occuring at a specific gran-

ularity. In contrast, discrete-event simulation matches LLM inference by matching the operational model of LLM inference, and simulating at a user-established granularity.

(DC2) Integration with other simulators: We identify three main simulation scopes (later modules) for Kavier: performance (**FR3**), sustainability (**FR4**), and efficiency (**FR5**). For each scope, we identify two design choices: we can either leverage existing, peer-reviewed work or design and implement from scratch. We analyzed, per scope, peer-reviewed literature and identified instruments for predicting the sustainability of (LLM) Ecosystems under workload, but no instrument capable of predicting performance or efficiency of LLM ecosystems, both cache-awarely and discrete-event. We thus choose to leverage a simulator with peer-reviewed capabilities for predicting sustainability, instead of building our own sustainability simulation module. We argue that, although adapting Kavier to an external simulator increases the engineering complexity compared to creating an in-Kavier module dedicated to simulating sustainability, adapting and using peer-reviewed functionality is more important. We thus choose to design and engineer only the modules that have never been explored before by our community (i.e., performance and efficiency), and leverage the peer-reviewed capabilities of a simulator for predicting sustainability.

(DC3) Simulation pipeline: We identify the MFR of predicting performance, sustainability, and efficiency of LLM ecosystems under inference. We identify four main pipeline architectures: 1) sequential pipeline (first performance simulation, then sustainability simulation, and lastly efficiency calculation), 2) parallel pipeline, where all modules run simultaneously with a final aggregation, 3) integrated pipeline, with a single monolithic simulator handling all the simulation and calculation aspects, and 4) hierarchical pipeline, adopting a multi-level simulation with different granularities. We identify a hybrid pipeline between the aforementioned pipelines, basing on a sequential design, where firstly the simulation system predicts performance (**FR3**), then sustainability (**FR4**), then efficiency (**FR5**). This pipeline enables per-module validation, which also allows for individual module adoption (thus, for leveraging the sustainability module from a peer-reviewed simulator, DC2). This pipeline, unlike the others, allows for a human-in-the-loop setup, who can manual verify and analyze between stages, orchestrate stages based on their needs, and make adjustments. Moreover, this architecture allows for failure tolerance, if a module fails (e.g., financial efficiency is functional without sustainability predictions, performance simulation is functional without sustainability simulation) (**FR6**) and simplified debugging. Lastly, this pipeline maximized modularity and validation rigor, while adhering to principles of software design and architecture [103, 150], and addressing (**NFR4**).

Throughout the design process, also matching the community-standard methodology on designing computer systems and ecosystems [14], we identify and analyze multiple design choices and select the option (usually, the tradeoff) that best aligns with the established requirements. For example, in Section 4.4, we analyze various designs of a system able to simulate performance with cache-awareness. In Section 4.5, we analyze two main design choices in simulating GPU performance, and compare simulation leveraging empirical-measurements and simulation leveraging mathematical and statistical approaches.

4.3.2 Kavier Input

The Kavier process begins in the input stage, where Kavier receives the experiment setup. Through *LLM Configuration* 🅐, the user can either select a prefab from the LLM Library or build their own by offering to the system the configuration parameters. Similarly, through *GPU Configuration* 🅑, the user can either select a prefab from the GPU Library or build their own. The *Prompt trace* contains two mandatory columns, the amount of input tokens and the amount of output tokens, and two optional columns, the tokenized input and the tokenized output; while the latter columns are optional for the overall simulation process, their presence allows for simulating with *Prompt Prefix Caching* policies 🅙. The *Simulation Setup* 🅓 allows the user to configure and customize the simulation based on their own needs and preferences, and provides configurable options such as snapshot granularity, simulation models, and output preferences.

4.3.3 Performance Simulator

Once the simulation setup is finalized, the simulation process begins. The *Performance Simulation Engine* 🅔 orchestrates and manages simulation processes which predict throughput and latency metrics (**FR3**). Firstly, the simulator predicts the duration of the prefill and decode stage, following a cache-aware simulation approach 🅘,🅙 (**FR2**). We identify caching as a central component of Kavier, with major and various impacts on the simulation time, highly dependent on the simulation policy.

Secondly, once Kavier predicted the amount of time per prefill and decode stage, it breaks the simulation time in simulation snapshots, based on the simulation granularity the user selected (**FR1**), following the formula $N_i = \lceil (T_p + T_d)/T_i \rceil$, where N_i is the number of intervals at which snapshots occur, T_p is the prefill time, T_d is the decode time, and T_i is the user-selected granularity at which the snapshoting occurs. For example, if the prefill time is 1.1, decode time is 9.0 seconds, and user selected a snapshotting interval is of 1 second, this would result in a total of 11 snapshots ($\lceil 1.1 + 9.0 \rceil$ 10 $= \lceil 10.1/1 \rceil = \lceil 10.1 \rceil = 11$ snapshots).

Thirdly, for each monitoring snapshot, Kavier simulates *KV usage* 🅚 and *GPU usage* 🅛, following trace-based simulation models which allow for versatile, stage-specific predictions (**FR1**), thus capturing the specific and distinct behaviours of prefill stage and decode stage. Both components 🅚 and 🅛 are linked to the caching system, which naturally influences the usage of the infrastructure based on the presence or absence of caching, or based on the caching policy (**FR2**).

Lastly, once the *Inference Simulation Engine* 🅔 finishes the simulation process, it transfers data to component 🅜, where a *Performance Report* is generated. This report contains Kavier's predictions on inference latency per prompt and system throughput (**FR3**). This data is ultimately transferred to the efficiency module ad is essential to compute the *Performance-Financial Cost* 🅦, and to 🅝, where Kavier's predictions are adjusted and made compatible with OpenDC input requirements. This results in a Kavier-output, OpenDC-input file. The transfer between Kavier and OpenDC happens through an internal *API* between the systems 🅩.

4.3.4 OpenDC Input

OpenDC input consists of specifications of hardware infrastructure, simulation setups, and workload traces. The *GPU configuration* from 🅑 coincides with the configuration from 🅞; this configuration can either be manually set up by the user or can be selected from a list of prefabs. Component 🅟 represents the simulation setups of OpenDC, partially coinciding yet not exhaustively with the simulation setup of Kavier. Component 🅠 is the workload trace OpenDC uses to predict energy consumption, while component 🅡 is the CO2 trace OpenDC uses to predict the amount of CO2 emitted for running the batch of inference tasks in a real-world environment. All the inputs are forwarded to the *Sustainability simulation engine* 🅢.

4.3.5 Sustainability Simulator

Following a similar approach of discrete-event simulation, OpenDC's simulation process is orchestrated by a *Sustainability simulation engine* 🅢. This firstly predicts the amount of energy the GPU infrastructure would consume in a real-life setup, through the *Energy Estimator* 🅣 (**FR4**).

Once energy predictions are completed, the *Sustainability simulation engine* redirects results for a *CO2 estimator* 🅤, a tool which leverages the given CO2 trace and the predicted amount of energy consumption and predicts the amount of CO2 consumed at every timestamp, following the granularity selected by the user (**FR4**), (**FR1**). The CO2 estimator component of OpenDC is proposed and detailed in depth by *Niewenhuis et al.* in [70].

Lastly, the sustainability predictions are aggregated into a *Sustainability Report* 🅥 (**FR4**). The sustainability report is transferred through the API interface 🅩 to Kavier, into the efficiency module, and is essential in computing *Performance-Sustainability Cost* 🅧.

4.3.6 Efficiency Simulator

Addressing (**FR5**), the efficiency component computes *Performance-Financial Cost* 🅦, represented in price per token per second, and the *Performance-Sustainability Cost* 🅧, expressed in watts per token per second.

The *Performance-Financial Cost*, component Ⓦ, simulates economic efficiency by combining a predefined, yet simple to modify and expand (**FR6**), financial model with the Kavier-simulated performance. Specifically, the simulator computes the cost of serving LLM inference under given hardware price, amortized over the number of tokens generated per second. Similarly, the *Performance-Sustainability Cost*, component Ⓧ, quantifies environmental costs. Environmental efficiency by combining power usage with token throughput. We identify power usage as a more robust metric than CO2 emissions, as CO2 emissions are location-dependent (different locations, especially countries, emit varying amounts of CO2 for energy production), whereas power usage is location-independent. We expand both efficiency models in Section 4.7.

4.4 Kavier Components for Simulating Key-Value and Prompt-Prefix Caching

In this section, we expand the caching abilities of Kavier, able to simulate key-value caching mechanisms in LLM inference (**FR2**). KV-Caching is a technique widely used in large-scale LLM deployments, which improves performance by reducing redundancy computation during the autoregressive decode phase [116, 10, 86, 84]; we expand KV-Caching simulation in Section 4.4.1. We also model prompt prefix caching, a technique which stores previously computed results in a caching system for a given period, and, if prompted again within the period, the system retrieves from memory instead of recomputing; we expand this caching technique in Section 4.4.2.

4.4.1 KV-Caching Simulation

Design Choices: We identify two main design choices for facilitating (KV-)cache-aware simulation (**FR2**). First, we consider predicting the average KV-Caching usage for the entire inference *(total-simulation model)*. Second, we consider simulating KV-Caching at each timestamp at a user-selected granularity *(discrete-event simulation model)*. We identify the latter approach to be better suited for Kavier, as it provides precise estimations of the KV-Cache usage at each timestamp and at adjustable granularity. Thus, discrete-event simulation model allows operators to analyze the ecosystem in finer detail and analyze how caching usage evolves over time. However, this discrete-event simulation approach comes with a higher performance cost ($O(n)$), compared to the total-simulation model, where KV-Caching is estimated only once, as an average over the simulation ($O(1)$).

We simulate following a discrete-event simulation approach [7]. During autoregressive generation, at each timestamp, the model takes as input the new token and the past keys/values from previous tokens' attention layers; then, instead of recomputing, the model caches the already computed states while generating tokens. For example, the model computes the attention layer for the first token from scratch. Then, for token 2, the model recomputes the attention only for token 2, and retrieves from the cache the layer for token 1. Then, for token 3, the model reuses the result for tokens 1 and 2. This process runs recursively until the last token is decoded.

KV-Caching reduces the time complexity from quadratic to linear. Specifically, KV-Caching reduces the time complexity from $O(n^2)$, where the model would process all n previous tokens for each of the n tokens, to $O(n)$, where the model only processes the new token and retrieves the past n computations from the cache. In our simulator, we assume KV-Caching as enabled by default, reflecting the current state-of-the-art in nowadays LLM serving frameworks (e.g., vLLM 0.9.1 [116]). However, the current design also allows disabling KV-Caching, improving Kavier's versatility for various scenario simulations.

The memory used by KV-Caching for each prompt is simulated using the community-vetted formula represented in Equation (4.1).

$$\mathrm{KV}_{\text{usage}} = 2 \times L \times H \times d \times N \times \mathit{sizeof}(\mathit{type}) \tag{4.1}$$

where L is the number of transformer layers in the model, H is the number of attention heads, d is the dimension per head, N is the number of tokens in the sequence, and sizeof(type) represents the size of the data type in bytes (e.g., Listing 4.1: KV-Caching, enabled and disabled.

```python
def get_decode_time(...):
    ...
    if kv_cache:
        return n_out * time_per_token
    else:
        return (n_out * (n_out + 1) / 2) * time_per_token
```

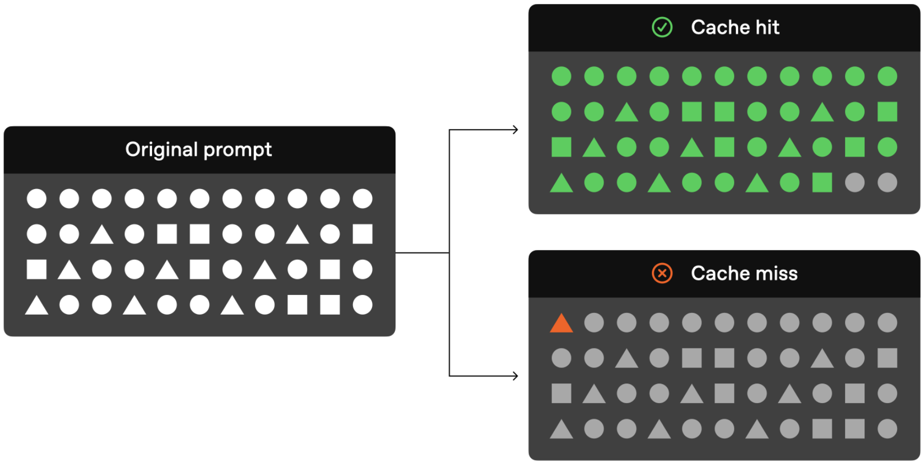

Figure 4.2: Prompt caching analogy used by OpenAI. Figure from [11].

float16 represents 2 bytes, float32 4 bytes). The factor of 2 represents storing two matrices, one for keys and one for values.

The logic of computing decoding time reflects the versatility of decoding with and without using KV-Caching, thereby illustrating the linear and quadratic behavior of the decoding stage. We represent this functionality in Listing 4.1, where n_{out} represents the number of output (decode) tokens.

4.4.2 Prompt Prefix Caching

Prefix caching is a system-level technique for performance improvement that caches new, unseen queries for a fixed amount of time. If a new query arrives with a matching prefix (e.g., the first 256 tokens), the results of the previous computations are retrieved from the cache instead of being recomputed [11]. While, to the best of our knowledge, OpenAI is the only company as of July 2025 to acknowledge using a similar caching technique, we believe it is an industry standard. However, it is still hidden under the curtains of closed-source codebases and inference pipelines.

OpenAI utilizes a prompt cache for very long prompts, exceeding 1,024 tokens, where they store/retrieve the prefill weights from the cache and do the decode stage independent of cache hit/cache miss; this approach is reported to have reduced latency by 80% and costs by 50%. In Figure 4.2 we showcase a figure taken from OpenAI's official blog on prefix caching [11]. On the left-hand side is a user prompt. If the first n tokens match and n exceeds the minimum threshold for the number of tokens in the matching prefix, then there is a cache hit (top right). However, if there is even one token in the prefix that doesn't match, the system gives a cache miss, even if the other tokens perfectly match (bottom right).

Design Choices: We identify two main design choices for designing a prompt prefix caching system. First, we consider the exact-match approach, which OpenAI uses, where any mismatch in the cached prefix results in a cache miss (i.e., if there is at least one token which does not match, there is a cache miss). Second, we consider

Listing 4.2: Prompt caching pseudocode.

```
PREFIX_CACHE = {}
PREFIX_CACHE_MIN_LEN = 256

for prompt in prompts:
    if len(prompt) > PREFIX_CACHE_MIN_LEN:
        prefix = prompt[:PREFIX_CACHE_MIN_LEN]
        if prefix in prompt_cache:
            handle_cache_hit()
            T_prefill = 0
            T_decode = simulate_decoding() # will be >0
            continue

    T_prefill, T_decode = simulate_decoding()
    if len(prompt) > PREFIX_CACHE_MIN_LEN:
        handle_cache_miss() # saves the prompt in the cache
```

an approximate-match approach that allows minor mismatches, configurable by a user (i.e., allows for cache hit even if at least one of the e.g., 1,024 prefix tokens does not match). We identify the exact-match approach as better suited for Kavier's initial design and better suited for simulation-driven experiments from this work, in which we evaluate various prefix caching policies against OpenAI system. Specifically, the exact-match approach allows for keeping a similar experimental setup with the real-world ecosystem used by OpenAI. Still, we envision future research into further designign flexible caching strategies with user-adjustable tolerance levels for prefix mismatches.

To simulate prefix caching in Kavier, we design a simple representation of a cache store. The user configures a minimum length parameter; if the prompt length is higher than the user-configured length (i.e., if the user-configured length is n, and the prompt length is at least $n + 1$), then the prompt input is cached. As Kavier iterates through the input trace of requests, we check for each prompt if the first n tokens from the respective prompt are stored in the cache. In the case of a cache hit, the real-world system would skip the redundant prefill phase and retrieve the post-prefill data from memory, thereby only performing the decode stage.

To improve simulator performance, the caching system only contains the input tokens and does not store the output tokens. Listing 4.2 shows a pseudocode of a system that simulates prompt caching:

We acknowledge the greedy yet powerful approach of this system. Although this design doesn't take into account the overhead of cache lookup, nor the overhead of retrieving the cache-stored response, we argue these actions are insignificant compared to the big model inference times, which would otherwise need to prefill and decode prompts of hundreds or thousands of tokens. However, albeit insignificant for individual prompts, in massive-scale operation scenarios, this overhead adds up. We leave the simulation of memory, memory levels, and networking for future research.

4.5 Kavier Module for Performance Analysis

In this section, we describe the performance models Kavier uses to simulate LLM inference (**FR3**). The inference process involves two core stages, prefill and decode, each with different performance particularities.

4.5.1 Performance simulation in the prefill stage

For the prefill stage, Kavier assumes a linear dependence between the number of input tokens and the decoding time, since the model does a full forward pass for each token in the user-given prompt [10]. Specifically, the simulator computes the prefill based on the total floating-point operations (FLOPs) required for the prompt, divided by the GPU's throughput in FLOPs per second. According to [151], the number of FLOPs per token is estimated at twice the number of parameters in the model. The authors explain the processing of one token as involving a forward pass through all the layers of the model, including both the attention and the feed-forward networks, which is approximately equivalent to twice the size of the model.

Kavier simulates the GPU's effective compute throughput by multiplying the amount of FLOPs per second by the efficiency factor, a hyperparameter that reflects the real-world limitations of GPUs; for example, empirical research conducted by *Recasens et al.* shows that LLMs achieve only up to 30-35% of the theoretical performance due to bottlenecks (e.g., memory, networking) or model (in)optimizations [152]. Similarly, systems may have overheads before each prompt execution; acknowledging this, we trace and measure real-world deployments and establish the prefill overhead as a hyperparameter, initially set to 25ms, but user-adjustable. Transforming the paragraphs above into a formula, Kavier simulates prefill time through the formula defined in Equation (4.2).

$$T_p = \frac{n_{\mathrm{i}} \times m_{\mathrm{p}} \times 2}{F \times C_e} + O \tag{4.2}$$

T_p is the simulated prefill time, n_i is the number of input tokens, m_p is the number of parameters in the model, F is the theoretical throughput of the GPU, measured in FLOPs per second. C_e, the compute efficiency, and O, the system overhead, are hyperparameters system-dependent.

4.5.2 Performance simulation in the decode stage

For the decode stage, Kavier models the time per output token and multiplies by the number of generated tokens, dependent on the presence or absence of KV-Caching.

KV on: If KV-Caching is enabled, then the real-world LLM inference process executes in $O(n)$ time complexity; thus, computation of each token requires roughly the same amount of computation, leading to a decode time which grows linearly with the number of output tokens [10, 116]. Kavier simulates the decode time of a model using KV-Caching using Equation (4.3).

$$T_{d,KV} = n_{\mathrm{o}} \times T_t \tag{4.3}$$

where $T_{d,KV}$ is the simulated decode time with KV-Caching enabled, n_o is the number of output tokens, T_t is the computed time per token.

KV off: If KV-Caching is disabled, the real-world LLM inference process takes $O(n^2)$ time complexity; thus, computation per token grows as the LLM traverses the decode stage, leading to quadratic time complexity. This time complexity is due to the need to recompute attention, from scratch, over an ever-growing sequence, without the possibility of caching the previous computations. Kavier simulates the decode time of a model not using KV-Caching using the Equation (4.4):

$$T_{d,KV} = (n_{\mathrm{o}} \times (n_{\mathrm{o}} + 1)/2) \times T_t \tag{4.4}$$

where T_d is the simulated decode time, n_o is the number of output tokens, T_t is the computed time per token.

Equations (4.3) and (4.4) introduce a new variable, T_t, the time required to compute one token. *Recasens et al.* empirically measure bottleneck in LLM inference, especially *"unveiling GPU bottlenecks in large-batch LLM inference"* [152]; the time per token is either compute-bound or memory-bound. In our simulation approach we simulate the latency for both compute-bound and memory-bound, then select the highest latency between the two. *"Minding the memory gap"*, and considering that *"no model exceeds 35% average (...) usage in either the prefill or decode phase"* [152], we consider the same hyperparameter for compute efficiency set at 30%. Similarly, the memory-read efficiency is empirically measured and reported in Table 1, [152], averaging at 57.6%, we thus implement a hyperparameter for memory-efficiency and set at 60%.

We synthesise the above paragraphs in formulas; Equation (4.5) shows the computation of compute-bound time per token, while Equation (4.6) shows the computation of memory-bound time per token.

$$C = \frac{f_{\text{tok}}}{F \times C_e} \tag{4.5}$$

where C is the compute-bound time per token, f_{tok} is the number of FLOPs per token (estimated as $2 \times m_p$), F is the theoretical throughput of the GPU in FLOPs per second, and C_e is the compute efficiency hyperparameter.

$$M = \frac{b \times m_{\text{p}}}{B \times M_e} \tag{4.6}$$

where M is the memory-bound time per token, b is the bytes per parameter, m_p is the number of parameters in the model, B is the memory bandwidth in bytes per second, and M_e is the memory efficiency hyperparameter.

Then, the final time of per-token computation is determined by taking the maximum between the compute-bound and the memory-bound, i.e., $max(C, M)$, computed in Equations (4.5), (4.6).

4.5.3 GPU Utilization

Simulating the GPU utilization of the ecosystem under LLM inference is crucial for simulating sustainability metrics. From the amount of GPU utilization, we can estimate the amount of power used by the GPUs and further simulate the amount of CO2 emitted for running the workload, addressing (**FR4**).

To the best of our knowledge, as of June 2025, there are no open-source traces showing the correlation between LLM inference and GPU utilization. Addressing this challenge, we decided to conduct our own ecosystem measurements. We deployed an LLM inference engine (vLLM 0.9.1, the latest version at the time of writing) and developed a tool for tracing LLM ecosystems, which we have released as open-source. We deployed the inference engine on clusters from two supercomputers: a cluster from SURF containing an NVIDIA A10 and a cluster from DAS-6 containing an NVIDIA A4000. We further detail and expand the tracing process in Section 6.2.

After empirical measurements, we observe an insignificant start-up time of ≈50-100 ms, when the GPU utilization grows from 4% to the user-established maximum utilization e.g., 98%; datacenter providers limit computing infrastructure to a certain cap, depending on the established SLOs and QoS. Then, throughout the inference process, the GPU utilization stays within the user-established cap, leading to an insignificant ≈50-100 ms when the GPU utilization decreases towards 0-10%.

Design choices: We identify two main design processes of simulating the GPU utilization, one observational-based, leveraging real-world traces, and one based on mathematical and statistical models. We identify the observation-based approach as superior, because GPU utilization remains largely constant throughout the inference, at the user-established maximum utilization, with negligible warm-up and cool-down periods. This design choice simplifies the computation complexity of the simulation process, while keeping a close-to-perfect simulation accuracy.

Hence, addressing the negligible variations in GPU utilization during inference, we simulate GPU utilization using the pseudocode presented in Listing 4.3.

4.6 Kavier Module for Sustainability Analysis

In this section, we detail the sustainability component of the Kavier-OpenDC system (**FR4**). Figure 4.3 illustrates the relationship between the sustainability models OpenDC provides. The input is processed by a power model, which predicts energy consumption and generates the output trace; this output trace is then further leveraged by a CO2 model, which predicts CO2 emissions. In Section 4.6.1, we expand the energy simulation (component ❸). In Section 4.6.2, we expand the simulation of CO2 emissions (component ❹).

Listing 4.3: Prompt caching pseudocode.

```
def get_gpu_utilization(t, t_prefill, t_decode, warm = 0.1, cool = 0.1):
    if t < warm: # if warming-up stage
        return 0.5 # i.e., 50% utilization

    if t < t_prefill + t_decode - cool: # if inference stage
        return MAX_GPU_UTILIZATION # i.e., user-established cap

    # if cooling stage
    return 0.5 # i.e., 50% utilization
```

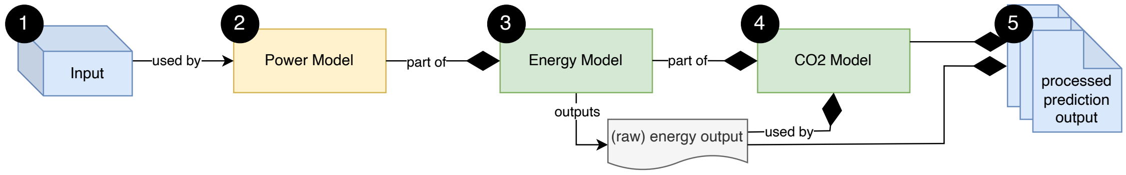

Figure 4.3: Relation between sustainability models in OpenDC.

4.6.1 Energy Simulation

To simulate energy usage, we leverage the capabilities of OpenDC, a peer-reviewed and top-tier simulator, with which we are coupling Kavier. We argue that, since OpenDC is already a vetted simulator through publications in many peer-reviewed venues [7, 78, 17, 48, 147], its models are reliable, accurate, and a robust model of reality. Furthermore, OpenDC implements capabilities of Multi- and Meta-Model simulation for energy models, which further increase the explainability and robustness of the simulation results [68, 47]. Table 4.1 shows the simulation models OpenDC-Kavier simulation system uses to predict power draw and, further, energy usage; each model is leveraged from peer-reviewed literature on datacenter (energy) simulation.

OpenDC simulates energy usage and outputs the results into a discrete-event output format, which shows, at the user-selected granularity, the amount of power drawn at a certain timestamp (represented in Watts and derived units), as well as the total energy usage which a real-world infrastructure would consume in a real-world experimentation setup (represented in Watt-hour and derived units). OpenDC exports the results in Parquet format, due to scalability, efficient storage, and cross-system compatibility (and thus portability) of this storage format [68, 153]

4.6.2 CO2 Emissions Simulation

Once the output of the energy predictions, OpenDC leverages the embedded carbon model to predict CO2 emissions. The CO2 model, specifically component ❹ from Figure 4.3, predicts at the user-established granularity (same granularity as the prediction of energy consumption), leveraging the amount of power drawn and the carbon intensity at the specific granularity, as represented in Equation (4.7). OpenDC implements the CO2 model explained in the peer-reviewed Footprinter [70].

Niewenhuis et al. show, using empirical measurements from ENTSO-E, the *"Europe's most ambitious electricity data platform"* [156], that the amount of CO2 emitted to produce one unit of energy varies significantly across location, time of the day, or weather conditions. In previous work, we showed that the same experiment run in locations with a high carbon footprint can emit up to 150-200 times more CO2 compared to low carbon footprint; for example, running the experiment in Germany emits a predicted 13.4 tCO2, while the same experiment run in Switzerland emits a predicted 0.081 tCO2 (Experiment 3 and Appendix C from [68]).

OpenDC receives as input a carbon trace, which shows the carbon intensity per timestamp, in a discrete-event

Name	Formula	Source
Sqrt	$P(u) = P_{\text{idle}} + (P_{\max} - P_{\text{idle}})\sqrt{u}$	[154, 9, 7]
Linear	$P(u) = P_{\text{idle}} + (P_{\max} - P_{\text{idle}})\, u$	[154, 9, 7]
Square	$P(u) = P_{\text{idle}} + (P_{\max} - P_{\text{idle}})\, u^2$	[154, 9, 7]
Cubic	$P(u) = P_{\text{idle}} + (P_{\max} - P_{\text{idle}})\, u^3$	[154, 9, 7]
MSE	$P(u) = P_{\text{idle}} + (P_{\max} - P_{\text{idle}})\, (2u - u^r)$	[7, 155]
Asymptotic	$P(u) = P_{\text{idle}} + \frac{P_{\max} - P_{\text{idle}}}{2}\left(1 + u - e^{-u/\alpha}\right)$	[7]
Asymptotic DVFS	$P(u) = P_{\text{idle}} + \frac{P_{\max} - P_{\text{idle}}}{2}\left(1 + u^3 - e^{-u^3/\alpha}\right)$	[7]

Table 4.1: Formulas of the power models the peer-reviewed OpenDC uses to predict energy usage. P_{idle}, $P_{\max}$ are the powers in idle and full-capacity states, u is device utilization, e is Euler's number, α is the utilization fraction at which the host becomes asymptotic, and r is a calibration parameter.

monitoring and reporting format. This trace is leveraged by ❹, together with the power drawn, to simulate the CO2 emissions. OpenDC outputs CO2 predictions in Parquet format, matching the format of predictions of energy usage.

$$C_{e,t=\alpha} = P_{t=\alpha} \times C_{i,t=\alpha} \tag{4.7}$$

Where $C_{e,t=\alpha}$ represents the CO2 emissions at timestamp alpha, $P_{t=\alpha}$ represents the power drawn at timestamp alpha, and $C_{i,t=\alpha}$ represents the carbon intensity at timestamp alpha. Alpha is identical for each variable.

4.7 Kavier Module for Efficiency Analysis

We now detail the efficiency models Kavier uses to compute the efficiency of the simulated experiment (**FR5**). We note that the accuracy of the efficiency module is equal to the simulation accuracy; in other words, mathematics is never wrong, but simulation can be. If the simulations have 100% accuracy, the efficiency predictions will also have 100% accuracy.

We further present two efficiency models, one for predicting financial efficiency and one for predicting sustainability efficiency. We argue that this component is crucial for datacenter and LLM operators in making informed decisions about potential deployments, as this component offers a homogeneous comparison metric for each type of efficiency, directly comparable, simple to understand, represent, and explain.

4.7.1 Financial efficiency

We express financial efficiency as the cost per token per second, essentially for the monetary aspect of running LLM ecosystems at scale, in profit-driven processes. Equation (4.8) shows the formula Kavier uses for computing the financial efficiency across LLM prompts, containing the total cost (set up by the user for their own specific financial model), the total amount of tokens (derived from the trace), from both the prefill and decode phase, and the total time needed for the prefill and decode phase (simulated by Kavier).

C from Equation (4.8) represents the cost; we identify C as a provider-dependent variable, as providers have distinct and highly diverse, some multi-dimensional financial models; for example, Microsoft Vidur shows a financial model in which users are charged at a GPU-hourly rate with ≈$10 per hour [12], while OpenAI API charges users by the amount of processed tokens [112]. Thus, we provide the financial model as embodying an abstract variable, yet simple to implement a specific financial cost.

The financial efficiency is, thus, represented in *currency per token per second* e.g., *€/t/s*, *LEU/t/s.*

$$E_f = \frac{C}{T} = \frac{C}{\frac{T_p + T_d}{\Delta T_P + \Delta T_D}} = \frac{C \times (\Delta T_P + \Delta T_D)}{T_P + T_D} \tag{4.8}$$

Where E_f represents the financial efficiency, C represents the operational cost, T_P and T_D represent the amount of prefill and decode tokens, respectively, and ΔT_P and ΔT_D represent the total inference time for prefill and decode stages, respectively.

We further identify Equation (4.9) for comparing two systems through financial efficiency ratio. We note that the efficiency of both systems from Equation (4.9), $s1$ and $s2$, should be quantified with the same metric.

$$R_f = \frac{E_{\mathrm{s1}}}{E_{\mathrm{s2}}} = \frac{\frac{\frac{P_{\mathrm{s1}}}{T}}{\Delta T_{\mathrm{s1}}}}{\frac{\frac{P_{\mathrm{s2}}}{T}}{\Delta T_{\mathrm{s2}}}} = \frac{P_{\mathrm{s1}} \times \Delta T_{\mathrm{s1}} \times T}{P_{\mathrm{s2}} \times \Delta T_{\mathrm{s2}} \times T} = \frac{(P_{\mathrm{A10}} \times \Delta T_{\mathrm{s1}}) \times \Delta T_{\mathrm{s1}}}{(P_{\mathrm{A10}} \times \Delta T_{\mathrm{s2}}) \times \Delta T_{\mathrm{s2}}} = \frac{\Delta T_{\mathrm{s1}}^2}{\Delta T_{\mathrm{s2}}^2} \tag{4.9}$$

Where R_f is the financial efficiency ratio between system $s1$ and $s2$, s1 refers to the first system, s2 refers to the second system, E is efficiency, P is price per hour, ΔT is time, T is the amount of processed tokens.

4.7.2 Sustainability efficiency

Following a similar approach as in Equation (4.8), we implement a sustainability model for computing the sustainability efficiency. Equation (4.10) shows the formula Kavier uses for computing the sustainability efficiency across LLM prompts; in essence, the only difference between the formula for financial efficiency and sustainability efficiency is the cost element; in the former, the cost is monetary, in the latter, the cost is of sustainability. The proposed equation for computing sustainability efficiency, Equation (4.10), contains the sustainability cost, either in Energy Usage or CO2 emissions (simulated by the sustainability module of Kavier-OpenDC setup), the total amount of tokens (derived from the trace), from both the prefill and decode phase, and the total time needed for the prefill and decode phase (simulated by Kavier).

The sustainability efficiency is, thus, represented in *Wh per token per second* (e.g., Wh/t/s) and *CO2 per token per second* (e.g., CO2/t/s), or both.

$$E_s = \frac{S}{T} = \frac{S}{\frac{T_p + T_d}{\Delta T_P + \Delta T_D}} = \frac{S \times (\Delta T_P + \Delta T_D)}{T_P + T_D} \tag{4.10}$$

Where E_s represents the sustainability efficiency, S represents the sustainability cost, T_P and T_D represent the amount of prefill and decode tokens, respectively, and ΔT_P and ΔT_D represent the total inference time for prefill and decode stages, respectively.

To compare the sustainability efficiency ratio of two systems, $s1$ and $s2$, we propose the formula represented in Equation (4.11). We note that, similarly to the financial efficiency ratio, both efficiencies must be quantified using the same sustainability metric.

$$R_s = \frac{E_{s1}}{E_{s2}} = \frac{\frac{S_1 \times (\Delta T_{P_1} + \Delta T_{D_1})}{T_{P_1} + T_{D_1}}}{\frac{S_2 \times (\Delta T_{P_2} + \Delta T_{D_2})}{T_{P_2} + T_{D_2}}} = \frac{S_1 \times (\Delta T_{P_1} + \Delta T_{D_1}) \times (T_{P_2} + T_{D_2})}{S_2 \times (\Delta T_{P_2} + \Delta T_{D_2}) \times (T_{P_1} + T_{D_1})} \tag{4.11}$$

Where R_s is the sustainability efficiency ratio between system $s1$ and $s2$, E_{s1} and E_{s2} are the sustainability efficiencies of system $s1$ and $s2$, respectively, S_{s1} and S_{s2} represent the sustainability cost (energy consumption or CO2 emissions), ΔTP and ΔT_D represent the total inference time for prefill and decode stages, respectively, and T_P and T_D represent the amount of prefill and decode tokens, respectively.

4.8 Requirement Validation

In this chapter, we presented a design of Kavier, a simulator for LLM ecosystems under inference, able to predict performance, sustainability, and multi-layer efficiency. We defined a set of requirements, both functional and non-functional, which guided our design process. We now evaluate the validity of our design against each requirement.

(**FR1**) **Support holistic simulation of the LLM inference process.**
Kavier models both inference stages of the LLM inference process, specifically the prefill stage and the decode stage, each with their own distinct characteristics and specific behaviours. Besides, Kavier follows a discrete event simulation paradigm, where the simulator predicts and exports predictions at a user-established granularity. Moreover, the Kavier-OpenDC system is designed to follow the same discrete event simulation model, reflecting real-world LLM performance and supporting detailed performance, sustainability, and efficiency reports at a user-set tradeoff between export granularity (with impact on performance) and available information (with impact of report detail).

(**FR2**) **Simulate with cache awareness.**
We design Kavier as a cache-aware simulator, capable of predicting LLM ecosystems under inference with KV-Caching enabled or disabled, and under conditions where prefix matching follows various cache store and cache hit policies. Thus, Kavier allows for versatility in experimentation and exploration of the impact of various caching policies on system performance, environmental sustainability, and efficiency. Kavier models the different impacts of caching on the different execution stages within the inference process.

(**FR3**) **Predict the performance of LLM ecosystems under workload.**
Kavier predicts system performance using community-vetted simulation models or models derived from these models, which the system then uses to simulate cache-awarely. Kavier predicts latency by determining the total amount of time required to answer a prompt, obtained by summing up the inference time for the prefill phase and the inference time of the decode phase. Throughput is further derived from the simulated latency (e.g., if a prompt contains, in total, n tokens, and the simulated latency for that prompt is of m seconds, then the throughput is n/m tokens per second). Lastly, the results are exported in both task-based and fragment-based traces, ensuring compatibility with OpenDC, a top-tier datacenter simulation framework, also adhering to (**FR6**).

(**FR4**) **Predict the sustainability of LLM ecosystems under workload.**
Kavier predicts sustainability as part of the Kavier-OpenDC simulation system. Leveraging the peer-reviewed simulation capabilities of OpenDC [7, 70, 78], Kavier simulates power draw, measured in Watts, essential for discrete event simulation (**FR1**), energy usage, measured in Watt-Hours, essential for overall system predictions (**FR5**), and, resulting from these, CO2 emissions, simulated in both discrete-event format (**FR1**) and overall system sustainability (**FR4**). For this, the simulation system uses real-world traces and user-defined granularity to provide accurate and detailed sustainability predictions. We design Kavier as modular and integrable with a peer-reviewed simulation framework. We select OpenDC and propose a detailed design of Kavier-OpenDC integration, emphasizing each party's role and how these two communicate with each other. Furthermore, the current design ensures a high degree of modularity and extensibility, allowing for a long software lifecycle and enabling the implementation of new functionality, modification of existing functionality, or deactivation of functionality.

(**FR5**) **Predict the efficiency of LLM ecosystems under workload.**
Kavier predicts efficiency as of LLM ecosystems through the Kavier Efficiency module, thus addressing (**FR5**). Kavier can predict financial efficiency, represented in cost per token per second, and environmental efficiency, represented in CO2/Wh per token per second. This module of Kavier is crucial for operators to differentiate and compare systems with ease, by simply analysing one or more efficiency metrics of the ecosystem.

(**FR6**) **Design Kavier compatible with other simulators and extensible.** We design Kavier as compatible with OpenDC, a peer-reviewed and state-of-the-art datacenter simulator. Kavier leverages the sustainability module of OpenDC to predict the energy consumption and CO2 emissions of LLM inference. We further validate this design component in Chapter 5, where we integrate an engineered prototype of Kavier with OpenDC. Addressing the extensibility aspect, we design Kavier as modular, where each core functionality can be modified, removed, or expanded, thus ensuring long lifetime of the simulator and also aligning with (**NFR4**).

(**NFR1**) **Provide in-meeting, near-interactive, same-day simulation results.**
We design the architecture of Kavier to minimize redundancies and enable the engineering of the simulator according to best software engineering practices. In Section 6.4, we analyze Kavier's performance through trace-based experimentation and observe that Kavier can simulate, at second-granularity, 500 GPU hours in under 10 seconds. Even more, Kavier meets (**NFR1**) even when simulating at milisecond-granularity, and is able to simulate 500 GPU hours in about 150 minutes.

(**NFR2**) **Aim to provide adequate simulation accuracy.**
We design Kavier as modular, easily modifiable. Thus, every simulation model can be modified as "plug-and-play." We validate the accuracy of Kavier in predicting performance of LLM ecosystems using real-world traces in Section 6.4. We obtain a MAPE error ratio of 7.39% for prefill and 4.00% for decode. Further addressing (**NFR2**), we argue that the prediction accuracy of the sustainability module of Kavier is already validated, by design, since Kavier leverages functionality from OpenDC, a peer-reviewed simulator within numerous venues [70, 7, 78, 17, 48] and used in national [1] and international scale projects [157]. Lastly, we note that the validity of the efficiency component of Kavier is directly dependent on the validity of the sustainability component and the performance component.

(**NFR3**) **Facilitate reproducibility and open science.**
Addressing the major reproducibility and closed-science challenge in computer systems research, we release all the designs, prototypes, engineered tools and instruments, traces, experiments, and information to the community. In short, we perfectly adhere to concepts of open-(real-)science. We ensure experiment reproducibility by releasing a reproducibility capsule (expanded in 6) which strengthens our experiments, claims, and findings.

(**NFR4**) **Adhere to modern software design and development standards.**
We present, in Chapter 5, the engineering process of Kavier, where we follow state-of-the-art standards of software architecture, design, development, and integration, some (also as) described in [103]. Matching the first stage of modern software design and development standards, we propose in this section high-level and detailed design of the Kavier simulation process, and detail each simulation model and component, and how they interact.

4.9 Discussion

We now summarize the contributions of this chapter, envision future research, and discuss potential threats to validity.

Summary: In this chapter, we propose and validate a design for a discrete and cache-aware simulator for LLM ecosystems under inference. Leveraging the reference architecture proposed and validated in Chapter 3, which matches a well-defined set of requirements, we provide a high-level overview of such a simulation instrument. We detail each simulation module: performance, sustainability, and efficiency. We also emphasize the simulation mechanisms our design uses for differentiating the specific behavior of the prefill and decode stages, and run cache-aware simulations.

Future Research: We envision a future design of a simulator that predicts the performance, sustainability, and efficiency of LLM training, modeling specifics of the training process, similarly to how Kavier models the specifics of the inference process.

We also envision integrating the Kavier-OpenDC simulation system into the first digital twin for datacenters, with a specific focus on measuring, simulating, and dynamically adjusting LLM ecosystems.

Threats to Validity: The design of Kavier, albeit robust for a first-of-its-kind tool, poses several limitations. Kavier assumes zero latency in processes of searching in and retrieving from the caching system, which is not applicable in real-world ecosystems.

Furthermore, while the sustainability module of Kavier-OpenDC employs Multi-Model simulation, the performance component simulates using a single model, trained for general-purpose scenarios, and thus prone to errors when encountering edge-cases. We envision future research in simulating the performance component through Multi-Model simulation.

5

Prototype and integration of Kavier

Designing an instrument for simulating LLM ecosystems and adhering to a robust reference architecture of the LLM inference Compute Continuum is a critical yet non-trivial challenge for the community. Addressing this challenge, in Chapter 4, we proposed a design for Kavier, a simulator for LLM inference that can predict the performance, sustainability, and efficiency of LLM ecosystems under inference. To fully validate this design and simulate LLM ecosystems, implementing a prototype is (also) a critical yet non-trivial challenge. We envision such a prototype as suitable for simulating LLM performance independently of other instruments (no such tools exist at the time of publication), suitable for simulating LLM sustainability when coupled with a peer-reviewed datacenter simulator, and suitable for predicting the efficiency of LLM ecosystems. This raises the research question: *(RQ3) How to implement and integrate Kavier within a peer-reviewed, discrete-event datacenter simulator?*

In this chapter, we implement Kavier following state-of-the-art software engineering principles, aiming for performance, long-term codebase sustainability, and open science. Then, we integrate Kavier with OpenDC and release the instrument as open-source. Lastly, we evaluate our implementation against non-functional requirements established in Chapter 4.

5.1 Overview

We implement Kavier, matching the state-of-the-art AtLarge design, implementation, and valuation process of researching computer systems and ecosystems [14]. Our contribution in this chapter is five-fold:

1. We implement, in Section 5.2, a working prototype of Kavier engineering the core functionality of the simulator, and adhering to the design proposed in Chapter 4. Kavier would, thus, be the first instrument for predicting performance, sustainability, and efficiency of LLM ecosystems under inference, following a discrete-event and cache-aware simulation approach.
2. We integrate Kavier in OpenDC, thereby leveraging the peer-reviewed capabilities of OpenDC for predicting the sustainability of datacenters (Section 5.3).
3. We showcase the GPU and LLM library, as well as the input interface of Kavier in Section 5.4.
4. We analyze, in Section 5.5, the engineered prototype against the requirements established in Chapter 4.
5. We reflect on Kavier, its limitations, and envision future engineering work in Section 5.6.

5.2 Implementation of a Kavier Software Prototype

In this section, we discuss the elements of Kavier that we implemented in our engineered prototype. Figure 5.1 represents the high-level design of Kavier and distinguishes the components of this design that we implemented

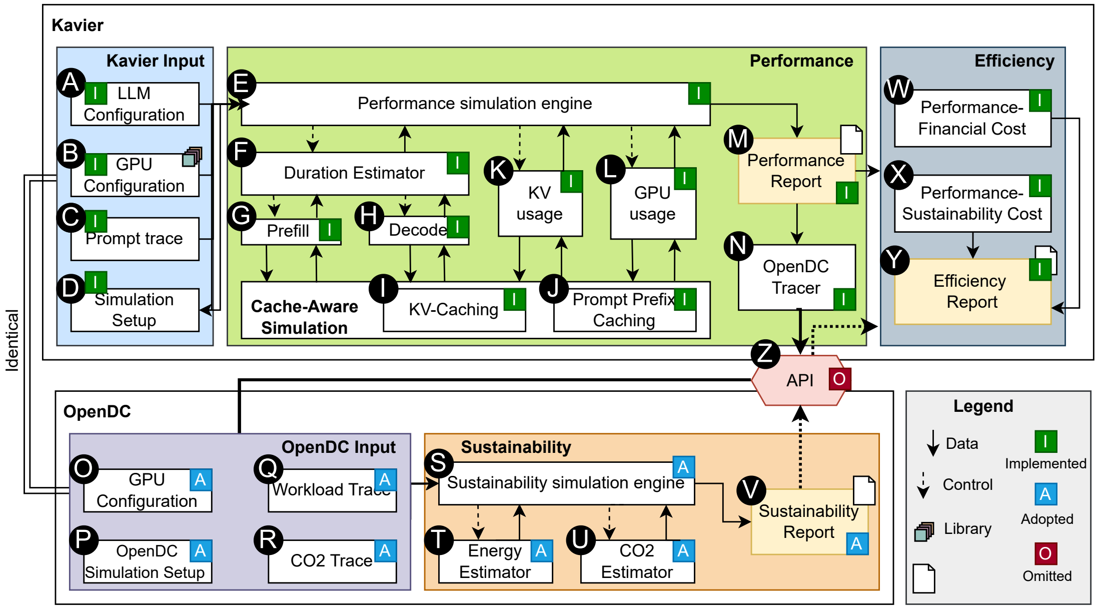

Figure 5.1: Kavier design from Chapter 4 showcasing specific components we implemented, adopted, or omitted in the engineered Kavier prototype.

in this work, adopted from the peer-reviewed OpenDC, or omitted.

(Implemented) Kavier input: We implement and release to the community a prototype of Kavier, which receives input through CLI arguments, thus making Kavier easy to operate and easy to adopt in a simulation conglomerate, where external tools leverage the functionality of our prototype. We identify multiple ways to configure the experiment, including through an external configuration file, codebase tweaks, or visual interfaces. We argue that the CLI approach is the most versatile option, as it is simple to implement, operate, and adapt, and adheres best to our functional requirements, especially (**FR6**), (**NFR4**), and (**NFR3**). Kavier can thus be configured either using default values or using one or more CLI arguments. We expand the input interface and how to set up experiments with Kavier in Section 5.4.

(Implemented) Performance: We fully implement the performance component of Kavier, ensuring the prototype's ability to simulate discrete-event and KV-Cache aware. Kavier can mimic the distinct performance behavior of the prefill and decode stage, and can simulate various caching policies, such as autoregressive KV-Caching or Prompt-Prefix Caching. KV-Caching can be enabled or disabled through user input. Prompt-Prefix Caching can be configured by prefix length, where the minimum prefix length can be specified through user input and considered in the simulation. For example, the minimum sequence length should be at least n tokens to be either cached or searched in the cache. This module exports a performance report, which is formatted to match the OpenDC input format.

(Leveraged) OpenDC: We leverage OpenDC "as-is," and, thus, leverage its status of community-reviewed and vetted simulator, which strengthens the validity of sustainability predictions. OpenDC outputs a sustainability report.

(Implemented) Efficiency: We tailor the input format for the efficiency component of Kavier such that it matches the output format of OpenDC. *(1) Performance-Financial Cost:* we implement a static financial model, where we consider a static price per hour for running LLM inference on a single GPU; we implement the default rate of 1.2$ per hour which is consistent with prices of renting a GPU as NVIDIA A10 in July 2025 [158]. We use this specific GPU because it is the machine to which we have access, and we also utilize it in our tracing and experiments. The price per hour is simply adjustable by the user. The financial model, albeit not "one-command-away-adjustable," is relatively simple to modify without breaking external functionality

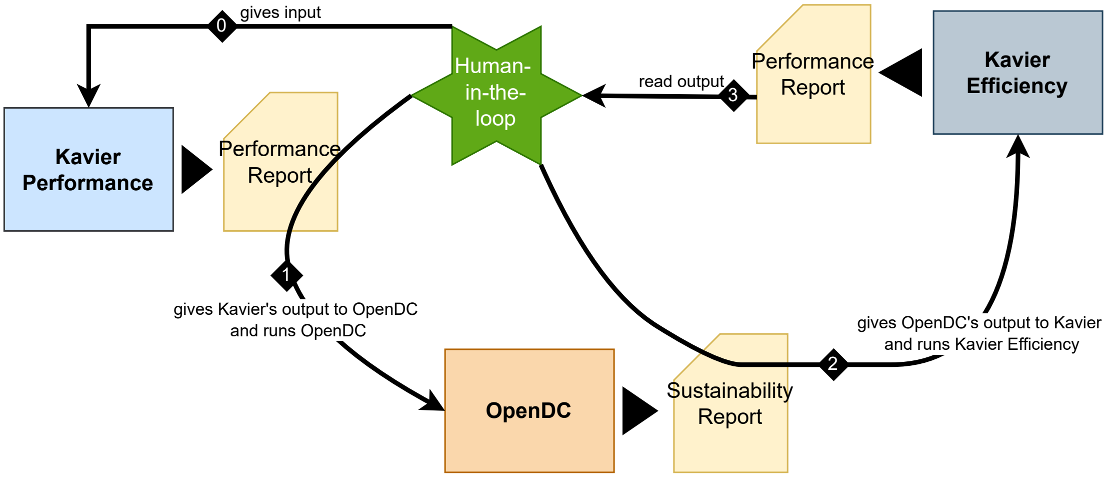

Figure 5.2: Kavier-OpenDC interaction with a human in the loop, who sets up the experiments, reads and analyzes outputs, and manipulates reports between simulators.

of the Kavier, thanks to the modularity of the designed and engineered prototype (**FR6**). The output is reflected in the amount of money per million tokens. *(2) Performance-Sustainability Cost:* the sustainability efficiency module computes the total amount of CO2, the total amount of tokens, and computes the amount of CO2 per million tokens. The results are ultimately packaged into a brief efficiency report.

5.3 Integration of Kavier with OpenDC

In this section, we focus on component Ⓩ from Figure 5.1, the *(omitted) API*, then present a high-level overview of the software engineering processes used in the prototyping process.

5.3.1 The Human-in-the-loop

(Omitted) API: We envision the API component as crucial for a digital-twinning system, with a human in the loop only for decision-making processes. However, in this prototype, Kavier acts as an LLM inference simulator decoupled from a digital twin (in fact, currently, there doesn't exist a digital twin for ICT ecosystems).

Figure 5.2 showcases the role of the "human-in-the-loop" in the simulation step. While a fully autonomous system would involve only steps ⓿ and ❸, our prototype involves two additional steps. The human gives input to Kavier ⓿ and waits until Kavier outputs a performance report; in Section 6.4 we show that this waiting period is usually a matter of seconds, and showcase that Kavier can simulate workloads of 500-GPU hours within 10 seconds, at second-granularity export rates. Then, in ❶, the human retrieves Kavier's input and gives it to OpenDC, then runs the OpenDC sustainability simulation part. After OpenDC's sustainability report ❷, the human forwards the predictions to Kavier to compute efficiency; the efficiency computation happens in a matter of milliseconds. Lastly, the user reads Kavier's prediction and analyzes the results.

Future work: In this work, we identify the human-in-the-loop as necessary because many experiments are pioneering and exploratory in nature. We envision much of the human-in-the-loop's work could be automated once these experiments become de facto standards in the community. However, we regard this extra step as beyond the purpose this work, where we prioritize conceptual, experimental, and trace-based contributions over engineering contributions. We therefore prioritize the core engineering features (those without which a simulator would be unable to simulate and thus meet functional and non-functional requirements) and reserve the API component for future work.

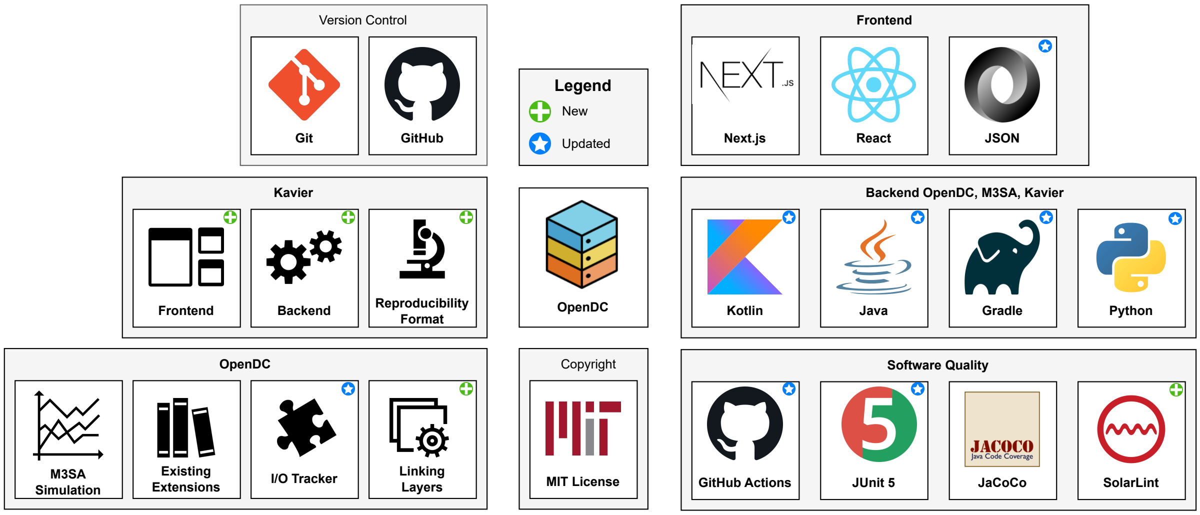

Figure 5.3: Technologies Kavier and OpenDC use for simulating performance, sustainability, and efficiency.

5.3.2 Software Engineering Processes

We employ industry-best software development practices and technologies in the engineering process of Kavier. The main codebase of Kavier is written in Python, the second most used programming language, due to its extensive support for various libraries and frameworks [159, 47]. The main codebase of OpenDC, which represents the sustainability component of Kavier, is written in Kotlin, a modern and fast-growing programming language, fully interoperable with Java, and already widely adopted by large companies [160, 69, 47]. Figure 5.3 shows the technologies Kavier prototype uses to simulate performance and efficiency of LLM ecosystems, and OpenDC and M3SA use to (multi-model-) simulate sustainability of such ecosystems.

We develop and envision future development of our simulator through industry-standard version control; we use Git and GitHub. In the Kavier repository, development happens through branches and the main branch can be modified only through pull requests (in the future, assuming a large-scale adoption of the tool, pull requests would be reviewed and merged by authorized repository maintainers and techical leads). Commits messages and pull requests follow industry-standard formats employed in large-tech companies, such as Google [161, 162].

We promote software quality through GitHub Actions, which run Continuous Integration (CI) [163] pipelines for each pull request. The CI pipeline consists of running automated test suites to spot code errors in functionality and simulation logic, and linting, to mandate adherence to best engineering/coding practices.

Although these software engineering processes increase the overall burden of engineering and maintenance, they ensure high-quality implementations and integrations of simulation instruments, cross-component compatibility, and the long life of the system as Kavier and OpenDC evolve [150, 69, 47].

5.4 Kavier Interface

In this section, we detail the LLM and GPU library that Kavier provides, as well as the experimental setup and versatility that Kavier enables. Then, after establishing this operational background, we give two input examples to Kavier, the most simplistic and the most complex.

Kavier receives input through a CLI where the user can configure the specifics of the simulation, or use default setups. Unlike file-based setups, which add extra complexity to experiment configuration, or visual interfaces, which decrease the ease of adaptation or integration with other instruments, the CLI is simple to implement and maintain, easy to operate, and easy to integrate with third-party tools.

LLM and GPU library: To simplify the input process and setup of LLM and GPU properties, we provide

Flag	Default	Description
`--llm`	`Llama-3-8B`	LLM prefab to simulate.
`--gpu`	`A10`	GPU prefab to simulate.
`--trace`	`N/A`	LLM workload trace to simulate.
`--output_folder`	`data/output_traces`	Output folder to save Kavier's predictions.
`--kv_cache`	`on`	Toggles vLLM-style KV reuse.
`--prefix_len`	`256`	Only prompts more than this many tokens populate the prefix cache (0 disables).
`--export_rate`	`0.1`	Sets the simulation granularity, in seconds.
`--flush_size`	`10,000`	Granularity of exporting to the output file, e.g., after 10,000 simulated prompts.

Table 5.1: Command-line flags to set up experiments for Kavier.

Listing 5.1: Simplest input to Kavier.

```
python -m kavier.main --trace name-of-the-trace.csv
```

Listing 5.2: Most detailed input to Kavier.

```
python -m kavier.main \
  --llm Llama-3-8B \                    % LLM simulated in experiments
  --gpu A10 \                           % GPU simulated in experiments
  --trace input-workload.csv \          % the workload trace
  --outputfolder output-folder \        % the output folder
  --kv_cache on \                       % enable KV-Caching
  --prefix_cache_min_len 512 \          % prefix caching 512 tokens
  --export_rate 0.01 \                  % second-granularity predictions
  --flush-size 100                      % flush granularity of 100 prompts
```

an LLM library, from which the user would only select a specific LLM or GPU, instead of configuring from scratch. We include in the library 8 LLMs and 8 GPUs widely used in real-world LLM inference, where the LLMs vary in parameter size (e.g., 8B, 30B, 176B), architecture (e.g., LLama, OPT, Bloom), etc, and the GPUs vary in tensor core performance (e.g., 312 teraflops, 2,040 teraflops, 4,800 teraflops), memory (e.g., 24 GB, 80 GB, 141 GB), etc. However, if the user doesn't find the needed LLM or GPU in the library, they can append to this library by simply adding a new entry to the array.

Flags for setting up Kavier: Table 5.1 describes the flags Kavier takes. For each flag, Kavier has a default value, which Kavier uses if the user leaves the field empty (e.g., the user doesn't need a specific simulation-driven experiment, but only wants to test the technical functionality of the setup).

We showcase in Listing 5.1 the simplest command Kavier takes, where the user only gives the workload trace. In contrast, we showcase in Listing 5.2 the most complex command Kavier takes, where the user exhaustively configures the simulation.

5.5 Requirement Validation

We now evaluate our prototype against the requirements we established in Section 4.8.

5.5.1 Functional Requirements

In this chapter, we implemented and integrated a prototype of Kavier that strictly matches the design proposed in Chapter 4. The implemented prototype of Kavier (hereafter referred to as Kavier) is a discrete event simulator of LLM ecosystems under inference, with a user-configurable export rate (**FR1**), and can model the inference process based on the presence and absence of KV-Caching (**FR2**). Kavier successfully predicts the performance (**FR3**), sustainability (**FR4**), and efficiency (**FR5**) of LLM ecosystems under inference, as we successfully validate in Chapter 6. Not only do we design Kavier as extensible and compatible with a peer-reviewed datacenter simulation framework, but we also implement and integrate Kavier with OpenDC (**FR6**), where a human-in-the-loop provides inputs, analyzes outputs, and manipulates intermediate files.

5.5.2 Non-Functional Requirements

Establishing and addressing FRs on Kavier, we answer *"what it does"* [46]. Now, with an engineered prototype of Kavier, we can successfully validate the design against non-functional requirements, and answer the *"how well it does"* [46]. In Section 6.4, we measure the accuracy of the engineered prototype and observe an error rate (MAPE) of 7.39% for the prefill stage and MAPE of 4.00% for the decode stage, well below the NFR-established bar of "under 10.00%," thus successfully validating (**NFR2**). Also in Section 6.4, Kavier proves its efficiency by stimulating 500 GPU hours in a matter of seconds, and at second-granularity, thus meeting (**NFR1**).

In this chapter, we engineered a prototype of Kavier, adhering to modern software design, development standards, and principles of open science (**NFR3**),(**NFR4**). We follow industry-standard technology, development, and version control pipelines, as well as software modularity. Matching (**NFR4**), we ensure the long life potential of Kavier, an envisioned state-of-the-art component of a future digital twin for LLM ecosystems under inference.

5.6 Discussion

We now summarize the contributions of this chapter and envision future developments.

Summary: In this chapter, we engineered Kavier, the first instrument capable of predicting the performance, sustainability, and efficiency of LLM ecosystems under inference. This aligns with the fifth step of the vetted AtLarge Design Process [14] and addresses RQ3.

Future work We envision future work in maintaining and growing Kavier, from the current prototype, which serves core and basic functionality for simulating LLM ecosystems, to a tool able to mimic exhaustively end-to-end, planet-scale LLM ecosystems, simulating multi-user workloads, geo-distributed datacenters, heterogenous accelerators, multi-level caching systems, adapting scheduling, workload carbon-aware scheduling, migration, and distribution, and various caching policies.

Multi-Prompt, Multi-GPUs: Currently, Kavier assumes one prompt running per GPU and simulates ecosystems with a single GPU. This is still valid for a prototype; inference engines such as vLLM keep the GPU usage close to maximum (≈ 95-96%) during the inference. However, for future versions, we envision scheduling as a crucial component of Kavier. This scheduler would enable operators of LLM ecosystems to analyze the impacts of different scheduling techniques (e.g., prioritizing jobs first, or batching small jobs on the same GPU) on performance, sustainability, and efficiency.

Multi-Level Caching: Currently, Kavier assumes zero latency in the case of a prefill cache hit. This is valid for a prototype, as the process of cache searching and cache retrieval takes only tens of milliseconds, which is insignificant compared to the seconds, sometimes even minutes, taken by the LLM inference. However, we envision future work in exploring multi-level caching and exploring the tradeoffs between searching in the cache and running the inference workload. For example, if the latency of retrieving from the deepest cache level (e.g., a different datacenter) would take two seconds, while the inference itself takes one second, the system would choose inference instead of cache retrieval.

Parallelism: Currently, Kavier simulates sequentially, one prompt at a time. While this is already sufficient for a prototype, we envision future engineering research where Kavier would parallelize the simulation process. While simulating 500 GPU hours within 10 seconds, at second granularity, on a regular-user machine (current performance), it is even more impressive to simulate 5 GPU years within 10 seconds at second granularity, or 500 GPU hours within 10 seconds at millisecond granularity.

6

Trace-Based Experiments with Kavier

Anticipating LLM ecosystems under inference is a critical, yet non-trivial, simulation challenge. In Chapter 4, we design Kavier, a KV-Caching-aware simulator, capable of predicting the performance, sustainability, and efficiency of LLM ecosystems under inference. Then, in Chapter 5 we propose an engineered prototype of Kavier, which we implemented and integrated with a state-of-the-art datacenter simulator. This build-up raises the research question: *(RQ4) How to evaluate a Kavier prototype with trace-based realistic scenarios?*

In this chapter, we address RQ4 by evaluating the engineered Kavier prototype against the requirements defined in Chapter 4. Then, we use Kavier's capabilities, many of which are novelties for the field, and analyze the impact of various caching policies on LLM inference performance, sustainability, and efficiency.

6.1 Overview

We evaluate Kavier matching the seventh stage of the state-of-the-art AtLarge design, implementation, and valuation process of researching computer systems and ecosystems [14]. Our contribution in this chapter is five-fold:

1. We deploy a state-of-the-art inference engine on real-world infrastructure and engineer a tracing instrument, which we subsequently use to trace the real-world infrastructure. We leverage traces that map the relationship between the amount of prefill and decode tokens and the time required for prefill and decode across various infrastructures. We release all the obtained traces, as well as the tracing instrument, as open-source and open-science. (Section 6.2).
2. We present the experimental setup in Section 6.3. We run experiments through discrete-event simulation, where we use Kavier for simulating real-world setups.
3. We analyze the impact of the prefill and decode length on the ecosystem performance. Then, we successfully validate Kavier's performance module against real-world measurements (Section 6.4).
4. We analyze how the presence and absence of KV-Caching affects the performance of various-sized, state-of-the-art models (Section 6.5).
5. Lastly, we analyze the impact of different prefix matching and caching policies on performance, and compare our simulation-driven, trace-based results with performance reports from OpenAI (Section 6.6).

6.2 Deploying and tracing LLM ecosystems

In this section, we present our approach to measuring LLM ecosystems deployed on real-world clusters.

Ecosystem Setup	NPT	NDT	Prefill Time	Decode Time	Latency	Throughput
s1	npt1	ndt2	t11	t12	l1	t1
s2	npt2	ndt2	t21	t22	l2	t2
...	...	...	...	...	...	...

Table 6.1: Sample chunk of a trace containing the needed data for validating the performance tier of Kavier. NPT is the number of prefill tokens and NDT is the number of decode tokens.

Timestamp	ContextTokens	GeneratedTokens
2024-05-10 00:00:00.009930+00:00	2,162	5
2024-05-10 00:00:00.017335+00:00	2,399	6
2024-05-10 00:00:00.022314+00:00	76	15

Table 6.2: Sample chunk from the *Azure LLM inference trace 2024* showing context and generated token counts per request.

6.2.1 Context: what traces do we need

To validate Kavier's accuracy in simulating the performance of LLM ecosystems, we need traces showing, for a given infrastructure setup, the relationship between the prefill length (i.e., the number of tokens) and the time required by the ecosystem to perform the decode (i.e., the number of seconds). We need a similar trace for the decode phase. These measurements enable us to derive performance metrics, including latency and throughput.

We present in Table 6.1 an example of a trace that matches the content needs for our experiments and for validating Kavier's predictions. While the format structure is flexible, and columns such as *latency* and *throughput* are optional (they can be derived from the rest of the data), the trace should contain information on ecosystem setup, number of prefill and decode tokens, and the time needed for prefill and decode stage.

6.2.2 Context: existent traces

We identify a large bank of traces released as open science and with significant contributions to the community. However, none of these traces match the information setup we described in Section 6.2.1.

Stojkovic et al. release the *Azure LLM inference trace 2024*, which contains three fields: timestamp, ContextTokens, equivalent to *NPT* from Table 6.1, and GeneratedTokens, equivalent to *NDT*. We present a sample from the Azure trace in Table 6.2. We identify that this trace does not contain information about the amount of time needed per inference phase. While useful for their work published in HPCA 2025 [164], this trace fails to present performance-related details.

Wang et al. open the trace used in BurstGPT [165] to the public. The released traces contain six columns: timestamp, model (e.g., GPT-4), request tokens (i.e., prefill tokens), response tokens (i.e., decode tokens), total tokens, and log type (e.g., conversation log, API log). Their trace also reveals failures in the LLM inference process, which are caused by various operational phenomena. We present a sample from the BurstGPT trace in Table 6.3. However, similarly to the Azure trace, their trace does not present performance-related details.

Pan et al. researched prefix caching techniques and released the traces they use in the Marconi paper [166]; in their work, the authors leveraged traces from peer-reviewed articles [167, 168, 169, 170], used for the experiment, and then released them as part of their reproducibility capsule. These traces contain crucial information for prefix matching, which we utilize in our experimentation as input traces to evaluate the multi-tier impacts of prefix caching. We present a sample from the Marconi trace in Table 6.4. However, while this trace contains useful information for prefix caching, it does not contain performance-related tracing.

Timestamp	Model	Request Tokens	Response Tokens	Total Tokens	Log Type
5	ChatGPT	472	18	480	Conversation log
825735	ChatGPT	94	11	105	API log
825731	ChatGPT	3,090	160	3,250	API log

Table 6.3: Sample chunk from the *BurstGPT* trace with request/response token counts and log-type metadata.

session_id	turn_id	ts	num_in_t	num_out_t	input_tokens	output_tokens
0	0	0.0	158	528	[1, 8853, 3051, 1115, 376, 12148, 6773, 445, 5828, ... 1792, 9092]	[1, 8853, 3051, 1115, 376, 2887, 27085, 29918, 29896, ... 22137, 9092]
1	0	4.0	99	189	[1, 8853, 3051, 1115, 376, 22550, 278, 1494, 2323, ... 1792, 9092]	[1, 8853, 3051, 1115, 376, 1576, 1959, 1234, 338, ... 22137, 9092]
2	0	8.0	22	137	[1, 8853, 3051, 1115, 376, 5816, 338, 278, 19087, ... 1792, 9092]	[1, 8853, 3051, 1115, 376, 1576, 19087, 4234, 491, ... 22137, 9092]

Table 6.4: Excerpt of a token-level trace capturing session and turn identifiers together with the full input and output token sequences. num_in_t is the number of input tokens, num_out_t is the number of output tokens.

6.2.3 Deploying on real-world clusters

After analyzing existing traces, we conclude that, as of May 2025, no publicly available trace contains the necessary information for validation, as summarized in Table 6.1.

Thus, we conduct our own tracings.

SURF: We obtain access to real-world infrastructure from SURF, the largest datacenter provider in the Netherlands[1]. Specifically, the offered infrastructure comprises a cluster with an NVIDIA GPU A10 [171], on which we deploy the latest version of vLLM at the time of tracing (v0.9.0), serving Llama-3-8B [172], and maintain the default settings [116].

DAS-6: Further, we obtain access to real-world infrastructure from the DAS-6 [173], the set of clusters at Vrije Universiteit Amsterdam, which contains 32 nodes, and provides access to GPUs as NVIDIA A4000 [174], NVIDIA A6000 [175], and NVIDIA A100 [176]. We deployed vLLM (v0.9.0) and kept the settings default [116].

6.2.4 Tracer and the tracing process

We then engineer Tracer, an instrument for tracing real-world LLM deployments, tailored to the LLM serving infrastructure we deployed in Section 6.2.3. Similarly to the rest of our contributions, we release Tracer as open science.

Tracer is a utility instrument that we use for automating the tracing process. Three main threads run in parallel: one for starting the inference engine, one for monitoring the cluster, particularly the GPU, and one for sending the input and receiving the output. However, the "order of operations" matters: first, we need to run thread 1, then thread 2, then thread 3. The human (me!) runs thread 1, and Tracer runs threads 2 and 3. We illustrate in Figure 6.1 the time frame of running and starting the threads.

[1] Many thanks to my team from the Network Institute, especially to Radu Apșan and Ivano Malavolta, who helped gain access to the SURF infrastructure.

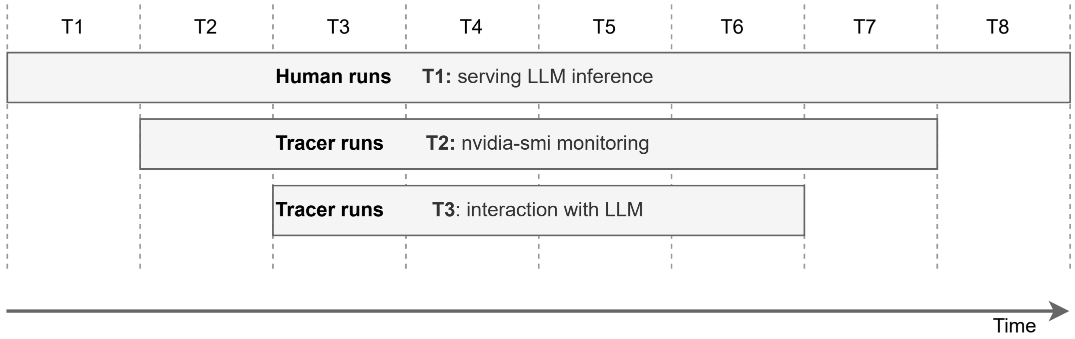
T1
T2
T3
T4
T5
T6
T7
T8
Human runs T1: serving LLM inference
Tracer runs T2: nvidia-smi monitoring
Tracer runs T3: interaction with LLM
Time

Figure 6.1: Threads for starting the inference engine (T1), running the measurement with NVIDIA-SMI (T2), and sending the prompt/receiving the answer (T3). Time progresses horizontally to the right, and the intervals between timestamps are considered equal, for the sake of exemplification and clarity.

Listing 6.1: Thread 1 – set-up commands on DAS-6.

```
1 srun -p defq --gres=gpu:A6000:1 --time=00:15:00 --pty bash -i
2 source /var/scratch/$USER/conda/etc/profile.d/conda.sh
3 conda activate vllm
4 module load cuda12.3/toolkit
5 export HF_HOME=/var/scratch/$USER/hf
```

Listing 6.2: Thread 1 command to serve LLama-3.1-8B with vLLM.

```
vllm serve meta-llama/Llama-3.1-8B
```

Thread 1: Running the inference

Firstly, the infrastructure needs to run vLLM. To connect on DAS-6, for example, we run the following set of commands from Listing 6.1:

This set of commands from Listing 6.1 selects a node by GPU, in this case, for exemplification purposes, an A6000, then reserves the node for 15 minutes (line 1). Then, the script loads Conda (line 2), vLLM (line 3), and the NVIDIA CUDA toolkit drivers (line 4). Then, to serve, e.g., LLama-3.1-8B, we run the command from Listing 6.2; we note that this command assumes the models are already loaded on the cluster, e.g., from HuggingFace [172].

Thread 2: NVIDIA-SMI sampler

Tracer starts a remote sampler, through which it connects to the cluster and analyzes GPU utilization using NVIDIA-SMI [177]. Tracer runs the command shown in Listing 6.3. This command starts just before thread 3 begins, and ends just after thread 3 completes; in other words, NVIDIA-SMI measurements span the period between sending the prompt and receiving the system response.

Thread 3: Sending the prompt and receiving the system's output

Lastly, Tracer sends the prompt to the system using the function presented in Listing 6.4. This function returns the system's text response. Immediately after that, the monitoring with NVIDIA-SMI is closed, and the results are stored in files.

Listing 6.3: Thread 2 – NVIDIA-SMI sampler.

```python
cmd = [
        "ssh",
        "-i", SURF_KEY_PATH,
        "-o", "StrictHostKeyChecking=no",
        f"{SURF_USER}@{SURF_HOST}",
        (
            "nvidia-smi "
            "--query-gpu=timestamp,utilization.gpu,utilization.memory "
            "--format=csv,noheader,nounits "
            f"--loop-ms={loop_ms}"
        )
    ]
```

Listing 6.4: Thread 3 – sending prompt and collecting reply.

```python
HEADERS = {
    "Content-Type": "application/json",
    **({"Authorization": f"Bearer {SURF_API_KEY}"})
}

def send_surf_prompt(model: str, prompt: str, max_tokens, temperature) -> str:
    payload = {
        "model": model,
        "prompt": prompt,
        "max_tokens": max_tokens,
        "temperature": temperature,
    }
    resp = requests.post(SURF_URL, headers=HEADERS, json=payload)
    ...
```

6.2.5 Traces

In this section, we present the traces we obtained. To ensure consistency and minimize system-dependent performance biases, we ran each measurement 10 times, then selected the median value. After each run, caches were deleted; between runs, the inference setup was kept identical. All measurements were run on the SURF infrastructure, which contains a cluster with an NVIDIA A10, serving vLLM, with a temperature of 0.8 and KV-Caching enabled.

We first measured the system's performance for the prefill phase. We sent prompts growing logarithmically in length, each of them asking *”Which is the most common word in the following text? Answer in exactly _1 word_: LOREM IPSUM DOLOR SIT AMET...”*. We used Lorem Ipsum text, generated using [178]. We present results in Table 6.5.

Then, we measured the system's performance for the decode stage. We send prompts requesting increasingly large responses *”Generate an exactly {size} word story about computers”*. We present results Table 6.6.

6.2.6 The LLM Trace Archive

We release all the traces used in this research as FAIR dataset [72], which includes both traces leveraged from peer-reviewed scientific articles and the traces we obtained in this work, by deploying and measuring real-world LLM ecosystems. We name this archive *the LLM Trace Archive.*

Setup	PS	ML [s]	MT [tokens/s]
SURF, A10, LLama-3.1-8B, vLLM default	64	0.054	1,192
SURF, A10, LLama-3.1-8B, vLLM default	128	0.072	1,776
SURF, A10, LLama-3.1-8B, vLLM default	256	0.123	2,095
SURF, A10, LLama-3.1-8B, vLLM default	512	0.213	2,408
SURF, A10, LLama-3.1-8B, vLLM default	1,024	0.436	2,349
SURF, A10, LLama-3.1-8B, vLLM default	2,048	0.819	2,501
SURF, A10, LLama-3.1-8B, vLLM default	4,096	1.749	2,354
SURF, A10, LLama-3.1-8B, vLLM default	8,192	3.860	2,127
SURF, A10, LLama-3.1-8B, vLLM default	16,384	7.347	2,230

Table 6.5: Prefill performance trace. PS represents the prompt size, in tokens, ML represents the median latency, and MT represents the median throughput.

Setup	RRS	MRS	ML [s]	MT [tokens/s]
SURF, A10, LLama-3.1-8B, vLLM default	64	53	2.3	22.5
SURF, A10, LLama-3.1-8B, vLLM default	128	106	4.5	23.1
SURF, A10, LLama-3.1-8B, vLLM default	256	206	9.0	22.8
SURF, A10, LLama-3.1-8B, vLLM default	512	409	18.1	23.0
SURF, A10, LLama-3.1-8B, vLLM default	1,024	769	36.3	21.9
SURF, A10, LLama-3.1-8B, vLLM default	2,048	1,838	73.0	25.1
SURF, A10, LLama-3.1-8B, vLLM default	4,096	3,109	147.2	20.7
SURF, A10, LLama-3.1-8B, vLLM default	8,192	6,585	299.5	21.9
SURF, A10, LLama-3.1-8B, vLLM default	16,384	13,940	617.6	22.5

Table 6.6: Decode performance trace. RRS represents the requested response size, MRS represents the median response size, ML represents the median latency, and MT represents the median throughput.

Societal impact: We envision the LLM Trace Archive as a main contribution of our work; this FAIR dataset can significantly alleviate future research efforts, otherwise spent on data collection or system measurement. Furthermore, the LLM Trace Archive contains unique tracing, the first FAIR dataset in the community to map the relationship between the amount of tokens (in both prefill and decode phases) and the corresponding execution times. These traces are essential for accurately simulating performance and for validating predictions, as we show in this chapter. Lastly, the archive enables researches who don't have direct access to datacenter infrastructure to conduct experiments, thereby making a step towards equal scientific opportunities for everybody.

Future work: Albeit already highly impactful on the community, we envision future work on the LLM Trace Archive, aided by Tracer, which would add traces for various ecosystems configurations run on various ecosystems deployments. These new traces could map the relationship between the system workload and system performance of e.g., various models (e.g., Llama, Granite), of different sizes (e.g., 8B parameter, 32B parameter), run on different GPUs (e.g., A10, A4000, A6000, A100), and with different vLLM configurations (e.g., KV enabled/disabled, different temperature varying from 0.0 to 1.0).

Model	Source	P	L	H	d_h	d_m	B
Llama-3-8B	Meta [172]	8	32	32	128	4,096	2
Llama-2-13B	Meta [179]	13	40	40	128	5,120	2
Granite-20B	IBM [180]	20	52	48	128	6,144	2
MPT-30B	Mosaic [181]	30	48	64	112	7,168	2

Table 6.7: Configuration of the LLMs used in our experiments. P = parameters (billions), L = Transformer layers, H = attention heads, d_h = dimension per head, d_m = hidden dimension, B = precision in bytes (2 = FP16).

GPU	Vendor	M	B	FP16	C	F	$P_{\min}$	$P_{\max}$
A10-24GB	NVIDIA [171]	24	600	125	9,216	1,695	20	150
A100-80GB	NVIDIA [176]	80	2,039	312	6,912	1,410	50	400

Table 6.8: Configuration of the GPUs used in our experiments. M = memory (GB), B = bandwidth (GB/s), FP16 = tensor-core throughput (TFLOPS/s), C = CUDA cores, F = boost frequency (MHz), $P_{\min}/P_{\max}$ = power draw (W).

6.3 Experimental setup

We validate and run experiments matching step seven of the state-of-the-art and community-vetted AtLarge methodology on design and validation of computer ecosystems [14]. To facilitate *reproducibility* and *consistency* among results, we run all experiments on the same physical infrastructure: an off-the-shelf MacBook Pro M3 Max, without other user programs running in the background. We run each non-deterministic experiment 10 times (e.g., performance validation) and report the standard deviation where applicable.

Marconi traces (public): We simulate using traces from Marconi [166], which leverages a set of traces from various peer-reviewed publicaitons, each of them containing real-world data anonymized. Marconi [166] release a set of traces which we aggregate into a singular, very-large LLM trace. We use the trace obtained in Section 6.2.5 as ground truth. Marconi used these traces in their peer-reviewed paper on prefix caching on hybrid LLMs; thus, we regard this trace useful also for our work when evaluating various prefix caching policies. Furthermore, we select the Marconi trace for its volume of data, which is crucial for simulating operation of LLM ecosystems at scale. Specifically, we aggregate all traces from [166], into a large trace which contains 96,870 entries, where each entry includes the user prompt and the system's response, as tokenized, the session ID, the turn ID, and the timestamp. With 3,000 sessions (i.e., 3,000 organizations, matching the terminology from [11]), we compute an average of 32.29 prompt-response pairs per session.

CO2 trace (public): To simulate CO2 emissions, we use a trace from ENTSO-E, leveraged and used also in our previous work [68]. This trace was collected from ENTSO-E Transparency Platform [88], *"an association representing 40 electricity transmission system operators from 36 countries across Europe"* [88]. In this work, we use a trace from July 2023 monitoring the amount of CO2 emissions per Wh of energy, at 15-minute intervals, in the Netherlands.

LLM Models: We validate Kavier against real-world measurements and traces, with an identical experimental setup: Llama-3-8B, vLLM (default settings, v0.9.0), A10. Then, throughout the experimentation process, we consider that the LLMs are deployed via vLLM. For the experimentation process, we select four state-of-the-art LLMs from Kavier's LLM library, from different industry leaders, and with various configurations. We represent the specifications of each LLM we use in Table 6.7.

GPU Units: Throughout the experimentation process, we consider GPUs running vLLM, keeping default settings, as in v0.9.0. Throughout the experimentation process, we use A10 and A100, matching the real-world configurations, also represented in Table 6.8.

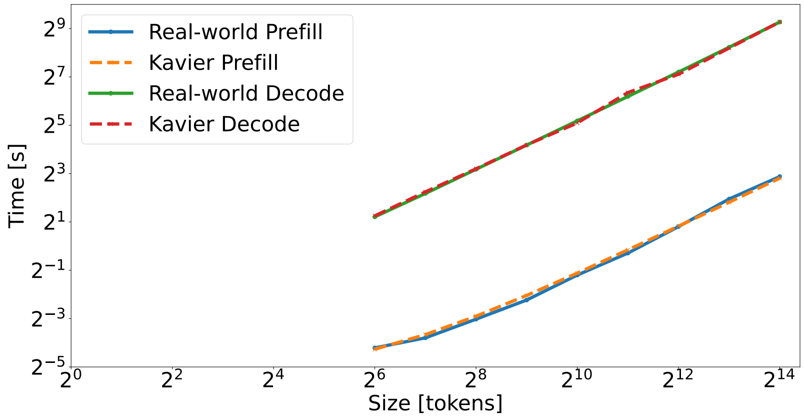

Figure 6.2: Kavier's predictions on prefill and decode time compared to the measured reality. The vertical axis depicts time, while the horizontal axis depicts the size/amount of tokens; specifically, the horizontal axis shows, for prefill, the amount of prefill tokens and, for decode, the amount of decode tokens. The MAPE for prefill time is 7.39%, and the MAPE for decode time is 4.00%.

6.4 Exploring Kavier accuracy and performance when simulating real-world LLM-inference processes

In this experiment, we investigate through discrete-event simulation the impact of input length (prompt size) and output length (LLM response) on performance. We first analyze the exponentially growing prefill and decode sizes and compare them with tracings of real-world deployments from Section 6.2.5, thus successfully validating against the established accuracy (**NFR2**). Then, we explore the performance of Kavier through large-trace experiments (**NFR1**), and showcase the efficiency superiority of the simulation approach compared to running real-world experimentation.

We run simulations of various prompts, which vary in prefill size exponentially between 2^6 and 2^{14}, and range in decode size logarithmically between 2^6 and 2^{14}. Figure 6.2 shows, on logarithmic scales (both vertical and horizontal), the simulated prefill and decode time against real-world measurements (MF1).

We identify a constant gap of 1-2 orders of magnitude between the prefill time and decode time, thus emphasizing the heavy computation involved in the decoding stage and the lightweight computation from the prefill stage. Even for very-large prefill lengths of 16,384 tokens (i.e., 2^{14}), the elapsed prefill time is under 10 seconds; in contrast, for the same very-large length, the elapsed decode time is over 500 seconds, approximately 9-10 minutes.

Addressing (**NFR2**), we quantify the accuracy of our simulation instrument by measuring the MAPE error ratio against ground-truth. According to (**NFR2**), Kavier should model reality with an error rate of at most 10%. However, in this experiment, Kavier achieves an MAPE of 7.39% for the prefill time and a MAPE of 4.00% for the decode time, successfully fulfilling (**NFR2**), and leading to MF2.

MF1	Kavier can simulate both stages of inference and model-specific behaviour.
MF2	Kavier simulates prefill with an error rate of 7.39 % and decoding with an error rate of 4.00 %.

Addressing (**NFR1**), we quantify Kavier's performance through a real-world trace, which aggregates all the traces released from Marconi [166]. This trace contains 96,869 tasks spanning over 502.1 GPU hours. According to (**NFR1**), Kavier should simulate in less than 1% of the equivalent of a real-world experiment; in this case, Kavier should simulate 502.1 GPU-hours in less than 5 hours, on a regular user machine. We

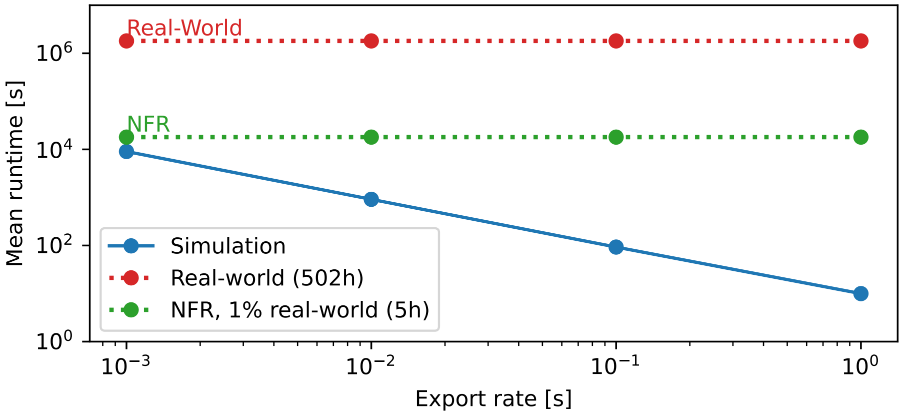

Figure 6.3: Measurements of Kavier's performance across various export rates, compared to (**NFR1**) requirements, and to the equivalent of running the experiment in a real-world setup. The vertical axis depicts time, and the horizontal axis depicts the export rate set for Kavier.

Export Rate [s]	Mean Time [s]	σ [s]	σ [%]
1	9.9	0.1	1.0
0.1	92.8	0.8	0.9
0.01	914.8	4.8	0.5
0.001	9,039.7	21.7	0.2

Table 6.9: Raw data from to Figure 6.3; σ is the standard deviation over 10 runs, s represents seconds.

identify a trade-off between simulation granularity and performance; the higher the granularity, the longer it takes to simulate (and vice versa). Thus, we evaluate Kavier's performance for export rates of 1 second, 100 ms, 10 ms, and 1 ms, and present the results in Figure 6.3. We observe that Kavier simulates the workload in under 10 seconds, at second-granularity (MF3), and even meets the established (**NFR1**), for millisecond granularity (MF4).

MF3 Kavier can simulate 500 GPU hours in 10 seconds, at second-granularity.

MF4 Kavier can simulate at millisecond granularity (2.5 hours), and still run in under 1% of the real-world equivalent (500 GPU-hours)

6.5 Analyzing the Impact of KV-Caching on LLM-Inference Performance

In this experiment, we investigate the impact of KV-Caching presence and absence on ecosystem performance through simulation aided by Kavier, and analyze how KV on/off affects various 8B-parameter LLMs.

1,000× Marconi Trace: For this experiment, we use the Marconi trace, one thousand times, to simulate massive-scale, real-world operation. While the original Marconi trace contains 3,000 sessions (i.e., 3,000 users each with one session), and an average of 32.29 prompt-response pairs per session, in this experiment, we up-scale the input trace to 3 million sessions, each with the same average of 32.29 prompt-response pairs per session.

Caching: In this experiment, we equip Kavier with a no-prompt prefix caching policy. We focus only on the impacts of token KV-Caching, where *"computation of a new token depends on interactions between its embedding and the previously stored intermediate KV-Cache tensors"* [2]. Since *Vaswani et al.* introduced

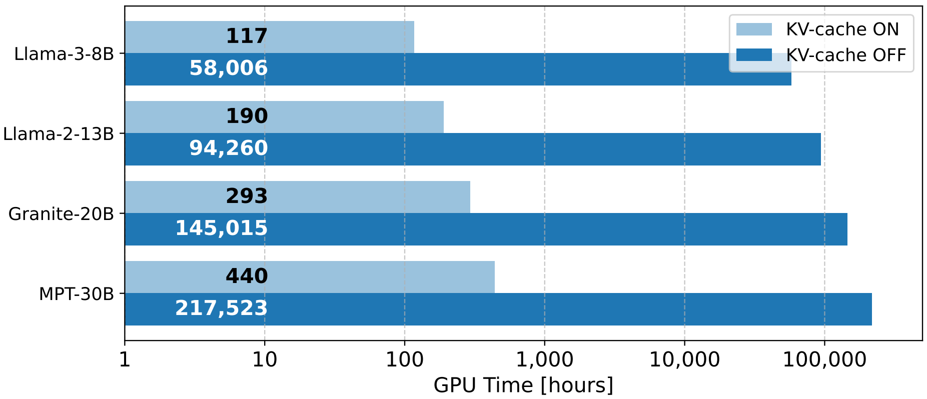

Figure 6.4: Impact of the presence and absence of KV-Caching on decode performance on industry state-of-the-art models.

KV-Caching in *"Attention Is All You Need"*, in 2017 [10], KV-Caching became an industry standard and is widely used in LLM ecosystems [84, 10, 2]. While KV-Caching has little impact on the prefill time, it reduces the time complexity for the decode phase from quadratic to linear; thus, in this experiment, we evaluate only the decode phase.

Simulated Infrastructure: We evaluate the impacts of KV-Caching on and off policy as run on an NVIDIA A100-80GB [176]. We simulate state-of-the-art LLMs, widely used in real-world setups, and growing in parameter sizes; we simulate Meta's LLama-3-8B [172], Meta's Llama-2-13B [179], IBM's Granite-20B [180], and Databricks' (MosaicML's) MPT-30B [181], all of them open-source and included in the LLM Library of Kavier. All the experiments run in this section total approximately 58 GPU years, as run on an NVIDIA A100. Although we do not have access, nor the physical time, to run these massive-scale experiments on a real-world NVIDIA A100-80GB, Kavier aids in predicting and anticipating how such real-world ecosystems would operate, in an availability-, time-, and cost-efficient way, in under 1 hour.

Figure 6.4 shows the performance of the four different models, using and not using KV-Caching. On the vertical axis, we represent the used models, of 8, 13, 20, and 30 billion parameters, and the presence or absence of KV-Caching. On the horizontal axis, we represent the total GPU time, represented on a logarithmic scale, required to run the given workload. We identify a 2 to 3 order magnitude gap between the presence and the absence of KV-Caching (MF5). Thus, while the absence of KV-Caching would lead to a total of 58.76 GPU years, the adoption of KV-Caching reduces the computation time by a factor of 497x, to only 0.11 GPU years (MF5). Moreover, we observe a direct relationship, converging to a linear growth, between the model size (number of parameters) and the decode time, for both policies of using and not using KV-Caching.

We identify the 2 to 3 orders of magnitude difference between KV-Caching enabled and disabled as a direct consequence of their fundamentally different time complexities. Specifically, the absence of KV-Caching in autoregressive token generation (decode phase) has a time complexity of $O(n^2)$ (n is the number of tokens in the decode sequence), and each new token needs to recompute attention over the entire sequence generated so far [10]; to recompute attention, the attention mechanism repeatedly performs computations over a continuously growing set of previously generated tokens. However, the presence of KV-Caching allows for keeping cached previously conducted computations (i.e., token matrices), and computing only new, unseen, and not caches token multiplications. The presence of KV-Caching reduces time complexity from quadratic ($O(n^2)$) to linear ($O(n)$).

This experiment thus emphasizes the need for a detailed analysis of the impacts of caches on performance. In general, caches offer significant performance benefits when workloads grow; for example, processing units (e.g., CPU, GPU) use multi-level caches to reduce latency when accessing frequently-used instructions and data, and adopt principles of temporal and spatial locality. However, caching can affect performance if workloads exhibit low locality, such as in cache-trashing scenarios in these processing units where frequent and incorrect cache misses leads to worsen performance, instead of improved performance.

We argue that caching in LLM ecosystems can reflect similar behaviour to caching in processing units. However, there is currently a gap in understanding the degree, sometimes magnitude, to which caching can help or "dishelp". We envision significant future work, both by us and by the community, in analyzing the impact of caches on the performance and subsequent aspects of LLM ecosystems.

MF5	KV-Caching can improve ecosystem performance by 2 to 3 orders of magnitude; in this experiment, KV-Caching improves performance by 497x.
MF6	Simulation enables prediction of 59 GPU years in under 1 hour.

6.6 Analyzing the Impact of Prompt-Prefix Caching Policies on LLM-Inference Performance, Sustainability, and Efficiency

In this experiment, we analyze the impact of prefix caching on prefill performance through discrete-event simulation aided by Kavier.

Prompt prefix matching - experiment setup: In this experiment, we equip Kavier with a `Least Recently Used (LRU)` cache eviction policy. We simulate *session caches* where users can benefit only from their own prefill caches, and caches are not shared between users. We also identify the existence of *global caches*, which are a large, global, and shared pool of caches among all users; in this experiment, we do not simulate global caches. Lastly, in our experiment, we set various maximum capacities of this cache, between 2 and 64 prompts, and we analyze cache hit ratios.

Prompt prefix matching - OpenAI setup: OpenAI acknowledges they use a prompt-prefix caching technique, available for 1,024 tokens or more, where *"only the prompt itself is cached, while the actual response is computed anew each time based on the cached prompt"* [11]. OpenAI uses a `system-load`-based eviction policy, where cached prefixes remain active for 5-10 minutes or up to one hour during off-peak hours [11]. OpenAI claims to be using the equivalent of what we define as *session caches*, which helps them reduce latency by up to 80% and costs by up to 75% [11].

6.6.1 Exploring matching prompt prefix length and size caches

We analyze various prefix caching policies, specifically disabling prefix caching and setting the minimum matching tokens to 1,024 (used by OpenAI [11]), 2,048, and 4,096. We select the baseline at 1,024, and consider this number as the industry standard, and the minimum matching size of the prefix matching for which the cache hits are still helpful for accuracy; we select twice and four times higher prefix matching sizes, thus higher caching strictness, which in theory should ensure better accuracy when cache hits occur.

We evaluated with caches of up to 8 prompts and 16 prompts. Considering the in-session scope of caches, caches of 8 and 16 prompts should be already sufficient, as the average conversation with an LLM has an average of *"8.95 turns (n.b., prompts) per dialogue"* [182] and *"65.5% of conversations finish within 10 turns"* [183]. We measured the impact of these setups on the Cache-Hit Ratio and Prefill time and show the results in Figure 6.5 and Figure 6.6.

In Figure 6.5, we represent the impact of the size of prefix matching on the cache-hit ratio and observe that a cache size of 16 prompts has approximately a double cache-hit ratio compared to a cache size of 8 prompts. We also identify a slight decreasing trend in the cache hit ratio as the prefix tokens increase and, thus, the caching policies become stricter. For 1,024 prefix tokens, the industry-standard (OpenAI) prefix caching policy yields a cache hit rate of 5.14% and 11.21%, for caches of maximum 8 and 16 prompts, respectively (MF7, MF8).

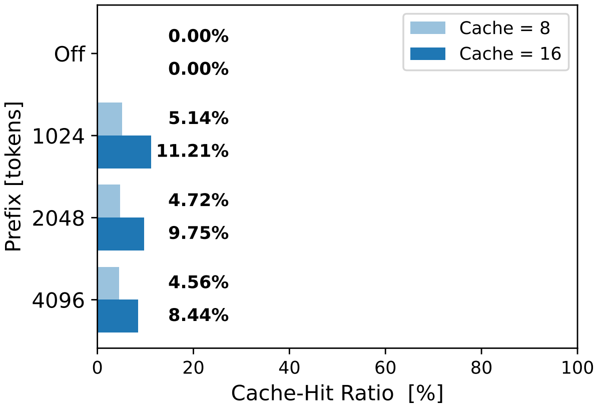

Figure 6.5: Prefix matching of various sizes against cache hit ratio. We measure with a cache size of 8 and 16 prompts, and an `LRU` eviction policy.

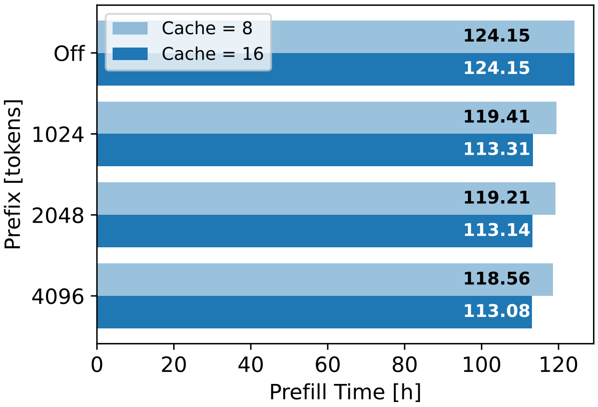

Figure 6.6: Prefill latency vs. prefix matching sizes. We measure with a cache size of 8 and 16 prompts, and an `LRU` eviction policy.

MF7 In this experiment, we identify cache sizes of 16 prompts as having a twice higher cache-hit ratio than cache sizes of 8 prompts).

MF8 For minimum prefix caching size of 1,024 tokens the cache-hit ratio is 5.14% for caches of 8 prompts and 11.21% for caches of 16 prompts.

In Figure 6.6, we illustrate the equivalent real-world prefill time for running the Marconi aggregated trace, which complements Figure 6.5. Simulating the `no-caching` policy, we observe a total prefill time of 124.15 GPU-hours, while caching can reduce latency by up to 11 GPU-hours. For cache sizes of up to 8 prompts, we observe an average improvement of approximately 5.1 hours, equivalent to a 4.0% improvement relative to no caching (MF9). For a cache size of up to 16 prompts, the average relative improvement is 8.8% (MF10), which is already substantial for SLOs and QoS when running LLM ecosystems at a societal scale. Lastly, we identify a relative improvement for the 1,024 tokens policy of 8.7%.

MF9 Prefix caching can reduce latency by 4.0%, over caching of 8 prompts.

MF10 Prefix caching can reduce latency by 8.8%, over caching of 16 prompts.

OpenAI reports that, using prompt prefix caching of 1,024 tokens, and a cache eviction policy based on system load, *"can reduce latency by up to 80% and cost by up to 75%"* [11]. We observe a one-order-of-magnitude gap between our findings in this experiment and the improvements reported by OpenAI [11]. We identify three main possible causes for this finding.

Potential cause 1: OpenAI's experimental setup and our experimental setup differ in the cache eviction policy (we use `LRU`, while OpenAI uses `system-load`), in cache size (we use various sizes for the cache, while OpenAI does not report the size), and in the input trace (our trace and their trace most probably don't coincide). Despite attempting to reproduce their experiments, OpenAI does not release the experimental setup or the used traces as open-source. Thus, we cannot investigate the eviction policy or the impact of the input trace further (we can further explore the effect of the cache size, which we do in Section 6.6.2). However, it seems unlikely that this specific difference in setup leads to such a large performance gap as identified in MF10.

Potential cause 2: OpenAI could be using *global caches*, instead of *session caches*. This is a large difference that goes beyond our setup and, because of the much higher potential to optimize when using the much larger global cache and its superior oversight on all prompts, it appears a likely explanation of the performance gap between our MF8 and OpenAI's reported performance. However, due to the closed-source nature of OpenAI's operational pipelines, we are unable to investigate this aspect further.

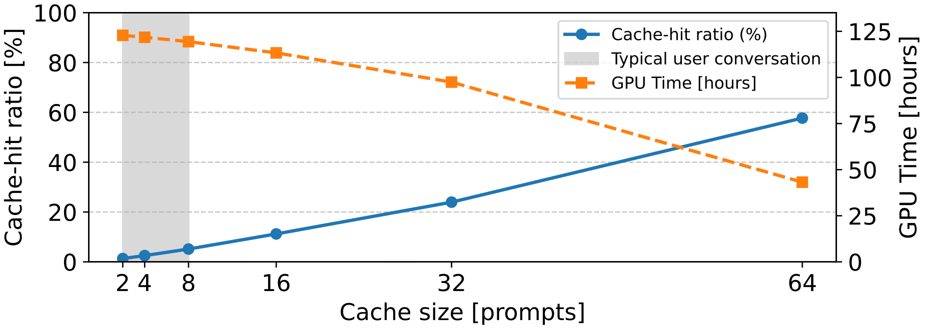

Figure 6.7: Impact of the size of in-session caches on cache hit ratio and total GPU time.

Potential cause 3: Our measurements and reports, or OpenAI's measurements and reports, or both, may contain core errors that would affect the final reported results. For potential external validation of results and future research, we release all traces and codebase as open science. However, we cannot investigate the experimental process of OpenAI's measurements since this information is not publicly available.

6.6.2 Exploring the implications of cache size

Further exploring *potential cause 1*, we analyze how the size of the cache, measured by the number of prompts it can hold, impacts the cache-hit ratio and the total GPU time.

Figure 6.7 illustrates our findings. On the horizontal axis, we represent the size of the cache, growing exponentially from 2 to 64 prompts; on the left vertical axis and with the blue straight line, we illustrate the cache-hit ratio; on the right vertical axis and with the orange discontinuous line, we illustrate the GPU time measured in hours.

We identify an increasing trend in the cache-hit ratio and a decreasing trend in GPU time, both of which are expected trends since the size of the cache increases and, thus, more prompts can be cached. However, the growing and diminishing trends are unexpectedly large, where the cache hit rate increases from 1.3% for caches of 2 prompts to 57.7% for caches of 64 prompts (MF11), and the GPU total time (i.e., total latency) decreases from 122.83 hours to 43.21 hours.

Focusing on the cache size of 64 prompts, we observe a 65.2% improvement in latency (relative to the absence of caching and 124.15 GPU hours), and expect a similar scale improvement in costs, both financial and environmental. While these findings match, at least in scale, the number reported by OpenAI [11], we argue that caches of 64 prompts are challenging to operate from both computational and usability perspectives.

From a computational perspective, storing individual caches of 64 prompts for each session would rapidly overwhelm the ecosystem's resources as the number of users scales to millions. LMSYS-Chat-1M, a large dataset containing one million real-world conversations from 25 LLMs, reports an average of 69.5 tokens per prompt [184]; for only 1 million concurrent users, all using GPT-4o-mini (an 8 billion parameter model, the smallest LLM OpenAI provides), 64 prompts per session, and 69.5 tokens per prompt, results in 2.33 GB per user, and 2.33 PB for hosting 1 million users at once. (6.1)-(6.6) show our computations.

$$T = M_t \times M_p \times M_s \times U \tag{6.1}$$

$$M_t = 2 \times 2 \times 32 \times 4096 = 524,288 \text{ bytes} = 0.52 \text{ MB per token} \tag{6.2}$$

$$M_p = 524,288 \times 69.5 = 36,438,016 \text{ bytes} = 36.44 \text{ MB per prompt} \tag{6.3}$$

$$M_s = 36.44 \times 64 = 2,332.16 \text{ MB} = 2.33 \text{ GB per user (session)} \tag{6.4}$$

$$U = 10^6 \text{ users (sessions)} \tag{6.5}$$

$$T = 2.33 \times 10^6 \text{ GB} = 2.33 \text{ PB} \tag{6.6}$$

where T=total memory, M_t=memory per token, M_p=memory per prompt, M_s=memory per session, U=users.

From a usability perspective, the average user session does not even reach 64 interactions; according to [185, 183, 182], the average chat contains less than 10 interactions ([185] reports an average of 3.5 interactions, [183] *"reports that over 65.5% of conversations finish within 10 turns"*, [182] report an average of *"8.95 turns per dialogue"*). This means that, usability-wise, the expected cache-hit ratio would be that corresponding to between 3.5 and 10 prompts; so, at most 10% cache-hit ratio (see shaded area in Figure 6.7), far off the maximum of above 65% for 64 or more prompts per dialogue.

We thus conclude that, given our experimental setup, storing in-session caches and reducing latency by costs by 75%, respectively 80%, is computationally- and usability- wise challenging (MF12).

MF11 Session caches of 64 prompts can lead to 57% cache hit ratios, and improve lantecy by 65%.

MF12 Session caches of 64 prompts, with small models (8B parameters, e.g., GPT 4o-mini), and 1 million concurrent users, would constantly occupy 2.33 PB of caches. This is computationally- and usability- wise challenging.

6.6.3 Performance, Sustainability, Efficiency

Throughout this experiment, we leveraged the capabilities of the Kavier prototype for predicting performance (**FR3**), sustainability (**FR4**), and efficiency (**FR5**). In total, Section 6.6 contains measurements spanning over 2,500 GPU (A10) hours in a real-world setup, and only 0.6 simulation hours on a regular-user machine (i.e., not a supercomputer) (**FR3**), (MF14). Similarly, the simulation approach consumed approximately 7,075x less energy than the real-world simulation equivalent (**FR4**) (MF15).

Lastly, through the Kavier performance module, we computed the financial efficiency and ratio between real-world-based and simulation-based experimentation (MF16). We consider the hourly running cost of the personal machine to be equal to the cost of renting an A10.

$$R = \frac{E_{\text{sim}}}{E_{\text{real}}} = \frac{\frac{P_{\text{sim}}}{\frac{T}{\Delta T_{\text{sim}}}}}{\frac{P_{\text{real}}}{\frac{T}{\Delta T_{\text{real}}}}} = \frac{P_{\text{sim}} \times \Delta T_{\text{sim}} \times T}{P_{\text{real}} \times \Delta T_{\text{real}} \times T} = \frac{(P_{\text{A10}} \times \Delta T_{\text{sim}}) \times \Delta T_{\text{sim}}}{(P_{\text{A10}} \times \Delta T_{\text{real}}) \times \Delta T_{\text{real}}} = \frac{\Delta T_{\text{sim}}^2}{\Delta T_{\text{real}}^2} = \frac{0.6^2}{2500^2} \approx 1 : 173,000,000 \tag{6.7}$$

where R is the financial efficiency ratio between simulation and reality, sim refers to simulation-driven experimentation, real refers to reality-driven experimentation, E is efficiency, P is price, ΔT is time, T is the amount of processed tokens.

MF13 - MF16 successfully validate the main function requirement *MFR: "Simulate performance, sustainability, and efficiency of LLM ecosystems under inference"*, and prove the superiority of our proposed approach, simulation-based experiments, over real-world-based experiments MF16.

MF13 Kavier enables conducting real-world experiments in a time and cost-efficient way, through discrete-event simulation.

MF14 Performance – Kavier: 0.6 h, Reality: 2,500 GPU h (via Kavier performance).

MF15 Energy – Kavier: 0.054 KWh. Reality: 375 KWh (via Kavier Sustainability).

MF16 Financial efficiency improvement – 1:173,000,000 Kavier : reality (via Kavier Efficiency).

6.7 Discussion

We now summarize the contributions of this chapter, the final content chapter of this thesis, and envision future experimentation. Many thanks, reader, if you have reached the page of our work, we hope you enjoyed the journey, 'cause we surely did!

Summary: In this chapter, we conducted the first open-science tracings of LLM ecosystems, which show the relationship between the prefill length, the decode length, and the time required to run the prefill and decode stages. We engineered Tracer, a utility tool for tracing LLM infrastructure. We release Tracer as open-science, and we release all the traces as an open-science LLM Trace Archive.

Then, using the ground-truth measurements, we validated Kavier against the established non-functional requirements for accuracy and performance, and validated Kavier against the remaining requirements through experimentation. We explored the massive impact (Token) KV-Caching has on LLM ecosystem performance, and identified differences of 2-3 orders of magnitude between its KV-on and KV-off. Lastly, we explored prompt prefix caching and reproduced results from OpenAI through experimentation aided by Kavier; we identified discrepancies between our findings and their reports, and identified three possible causes. We then explored in depth one of these causes (the other two could not be explored because OpenAI's operational pipelines are closed source). We then validated Kavier-aided experimentation versus real-world experimentation and identified orders of magnitude improvements of the simulation approach, the largest being of 1:173,000,000 in financial efficiency improvement.

Future exploration: Kavier enables the exploration of large-scale systems in a time- and cost-efficient manner, without requiring access to real-world ecosystems or incurring the financial, time, and configuration burdens. We envision Kavier as aiding in exploring future aspects of LLM inference, with the current prototype.

We envision future research and exploration on how different prefix caching policies and cache eviction policies (e.g., least recently used, least frequently used, random) can impact system metrics. We also envision future research in exploring the most energy-efficient configurations while still meeting performance-sustainability real or synthetic SLOs.

7

Conclusion and Future Work

In this chapter, we summarize the contributions of our work and envision future research and exploration of LLM ecosystems through simulation.

7.1 Conclusion

We investigated in this work *how to enable analysis of LLM ecosystems through discrete-event simulation (MRQ)*. We identified and addressed three research questions, methodologically matching the state-of-the-art AtLarge vision on design of distributed systems and ecosystems [14]. In Chapter 1, we described the societal impact of LLM ecosystems and described the potential benefits of a simulation instrument of LLM ecosystems under inference. We identified two main problems: the lack of such a scientific instrument and the lack of a (robust) reference architecture of LLM ecosystems under inference, on which the simulator would map against. In Chapter 2, we provided relevant background and analyzed existing reference architectures, prior to this work. In Chapter 3, we designed a reference architecture for LLM ecosystems under inference and validated the architecture against real-world ecosystems. In Chapter 4, we proposed Kavier, a scientific instrument for simulating the performance, sustainability, and efficiency of LLM ecosystems under inference. We then prototyped Kavier Chapter 5. In the absence of real-world traces necessary for validating Kavier, we deployed LLM ecosystems and deployed real-world infrastructure. We then successfully validated Kavier in Chapter 6, and analyzed the impact of various caching policies on ecosystem, performance, sustainability, and efficiency. We now answer each research question punctually:

RQ1 **How to synthesize and validate a reference architecture of LLM ecosystems?**
In Chapter 3, we have conducted a literature review and analyzed existent reference architectures of LLM ecosystems under inference. We detailed positives and negatives and identified that none of the existent reference architectures are sufficient to map an LLM ecosystems simulator upon. Existent architectures are either incomplete [61], non-inference oriented [61, 62, 63], assume a (too) high degree of homogeneity of LLM ecosystems [62], vetted, following state-of-the-art approaches in distributed systems, but too universal [16], or do not follow a distributed systems approach [62]. To design a robust reference architecture for LLM ecosystems under inference, we defined a set of design requirements and design principles which guide our design process. We then proposed a reference architecture of the current continuum of LLM ecosystems, mapping to real-world deployments. To validate our reference architecture, we explicitly mapped our model to four real-world ecosystems, out of which two in-detail (IBM, OpenAI) and two high-level (Ubicloud, Databricks), and against a state-of-the-art reference architecture from the scientific community.

RQ2 How to design Kavier, a scientific instrument for cache-aware simulation analysis of the performance, sustainability, and efficiency of LLM ecosystems under inference?
In Chapter 4, we have designed Kavier adhering to stage 1 of AtLarge Design Process [14]. We established a set of requirements which guide our design process, then propose a high-level design of Kavier, the first scientific instrument for predicting the performance, sustainability, and efficiency of LLM ecosystems under inference, through discrete-event simulation and cache-awareness. We designed Kavier as modular and leveraging peer-reviewed capabilities of predicting sustainability of OpenDC [7, 70]. We also designed Kavier as able to predict the continuum cache-aware, and model the impacts various caching policies (e.g., prompt prefix caching, KV-Caching) have on LLM ecosystems. We then detailed each main module of Kavier, specifically the performance module, the sustainability module, and the efficiency module. Lastly, we systematically evaluated our proposed design against established functional and non-functional requirements.

RQ3 How to implement and integrate Kavier within a peer-reviewed, discrete-event datacenter simulator?
In Chapter 5, we implemented a prototype of Kavier and integrated with OpenDC. We developed Kavier following state-of-the-art software engineering practices and so industry-standard software engineering processes. We integrated Kavier and OpenDC, thus allowing Kavier to leverage the peer-reviewed capabilities of OpenDC to simulate sustainability [7, 70].

RQ4 How to evaluate a Kavier prototype with trace-based realistic scenarios?
In Chapter 6, we collected traces for validation of Kavier and simulation-driven experimentation aided by Kavier. We identify a gap in open-source traces, as none of them was revealing the relationship between the prefill/decode size and the prefill/decode performance. To address this challenge, we deployed LLM ecosystems on real world infrastructure from SURF, engineered a utility tracing tool, and obtained traces matching the validation needs for Kavier. We then validated Kavier and identify its ability to simulate hundreds of GPU hours within seconds, with at-second-granularity predictions, and with error rates of less than 8%. With a validated prototype, we analyzed impacts of the presence and absence of KV-Caching on massive-scale LLM inference. Laslty, we analyzed impacts of prompt prefix caching on system performance and contrasted our findings with real-world reports from OpenAI.

We released all instruments, tools, and traces as open-source and open-science. Specifically, we release a parent-repository containing:

1. Kavier, the scientific instrument we designed, engineered, and validated in this work.
2. The LLM Trace Archive, containing all the traces used in this work, both leveraged from peer-reviewed articles and obtained by us, by tracing real-world deployments.
3. Tracer, the utility tool we built for tracing LLM ecosystems.
4. A reproducibility capsule of all our experiments, with a guide on how to reproduce our findings.
5. This thesis.

The parent-repository can be found on GitHub, via
`https://github.com/Radu-Nicolae/On-Simulating-LLM-Ecosystems-under-Inference`.

7.2 Future work

We envision four main areas of future research, building upon our contributions from this work.

1. *Simulation of heterogeneous, highly-distributed LLM ecosystems:* We plan to broaden the fidelity of the models and expand the current prototype of Kavier to predicting high-heterogeneity infrastructure: multi-GPU, TPU, and NPU parallelism, network, memory contention, geo-distributed and multi-layered caches, and multi-level metric report card. By supporting simulation of highly heterogeneous and

distributed infrastructure, Kavier can evolve from the current simulator status to an LLM ecosystem simulation twin, able to close to perfectly mimic reality.

2. *The LLM ecosystem Digital Twin:* Digital twins are simulation ecosystems, where simulation instruments (e.g., Kavier) are connected to the operational ecosystem. There is a continuous feedback loop between *1) the LLM ecosystem* running LLM workloads (e.g., inference, fine-tuning, training), *2) the metrics reported by the ecosystem* (operational data analytics, live telemetry), which are transmitted to the simulator, *3) the simulator's predictions* to adjust the infrastructure such that the ecosystem operates as efficient and performant, while still meeting the SLOs and QoS, predictions which are transmitted to the LLM ecosystem and *4) the LLM ecosystem* which reacts based on the simulator's predictions. There is currently no such digital twin for LLM ecosystems, nor for ICT infrastructure. We envision Kavier as taking the role of the simulator within a potential digital twin of LLM ecosystems under inference.

3. *LLM ecosystems under training workloads:* We identify simulation of the training stage as potential future work and future capabilities of Kavier. While inference represents the largest proportion of an LLM's lifetime, the training of *large* language models also raises high performance, sustainability, and efficiency challenges. We envision a future version of Kavier as capable of simulating the holistic lifecycle of an LLM ecosystem, from the initial deployment in the training pipeline until the last prompt of the last user interacting with the respective LLM ecosystem. Similarly, to the inference process, we envision digital twins as crucial also for LLM training.

4. *Educating future generations:* We plan to develop educative material around Kavier and OpenDC, and deliver as a series of interactive workshops, seminars, and assignments to educate future generation of scientists and engineers, on how to responsibly use, deploy, and monitor LLM ecosystems, focusing on the model inference aspect. Thanks to the open-source nature of all our contributions, such material can be developed both by us or by other researchers and educators from the community, and can be in-depth explored by the students who would engage in these educational activities. We envision various difficulty educational materials, matching various academic ages, from highschool, to Bachelor's, Master's, and Doctorate levels. Such educational activities, both emerging from this work and from other work, are essential for training future generations on systematically, in-depth, and ethically exploring, researching, and engineering LLM ecosystems, and, overall, on responsibly *Massivizing Computer Systems.*

Bibliography

[1] A. Iosup, F. Kuipers, A. L. Varbanescu, P. Grosso, A. Trivedi, J. S. Rellermeyer, L. Wang, A. Uta, and F. Regazzoni, "Future computer systems and networking research in the netherlands: A manifesto," *CoRR*, vol. abs/2206.03259, 2022.

[2] C. Zhang, K. Du, S. Liu, W. Kwon, X. Mo, Y. Wang, X. Liu, K. You, Z. Li, M. Long, *et al.*, "Jenga: Effective memory management for serving llm with heterogeneity," *arXiv preprint arXiv:2503.18292*.

[3] A. A. Chien, "Genai: Giga$$$, terawatt-hours, and gigatons of co$_2$," *Commun. ACM*, vol. 66, no. 8, p. 5, 2023.

[4] C. Wu, R. Raghavendra, U. Gupta, B. Acun, N. Ardalani, K. Maeng, G. Chang, F. A. Behram, J. Huang, C. Bai, M. Gschwind, A. Gupta, M. Ott, A. Melnikov, S. Candido, D. Brooks, G. Chauhan, B. Lee, H. S. Lee, B. Akyildiz, M. Balandat, J. Spisak, R. Jain, M. Rabbat, and K. M. Hazelwood, "Sustainable AI: environmental implications, challenges and opportunities," in *Proceedings of the Fifth Conference on Machine Learning and Systems, MLSys 2022, Santa Clara, CA, USA, August 29 - September 1, 2022* (D. Marculescu, Y. Chi, and C. Wu, eds.), mlsys.org, 2022.

[5] B. Cottier, R. Rahman, L. Fattorini, N. Maslej, and D. Owen, "The rising costs of training frontier AI models," *CoRR*, vol. abs/2405.21015, 2024.

[6] J. Simon, "Large language models: A new moore's law?." `https://huggingface.co/blog/large-language-models`, 2024. Accessed: 2025.

[7] F. Mastenbroek, G. Andreadis, S. Jounaid, W. Lai, J. Burley, J. Bosch, E. V. Eyk, L. Versluis, V. van Beek, and A. Iosup, "Opendc 2.0: Convenient modeling and simulation of emerging technologies in cloud datacenters," in *21st IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGrid 2021, Melbourne, Australia, May 10-13, 2021* (L. Lefèvre, S. Patterson, Y. C. Lee, H. Shen, S. Ilager, M. Goudarzi, A. N. Toosi, and R. Buyya, eds.), pp. 455–464, IEEE, 2021.

[8] S. K. Gupta, R. R. Gilbert, A. Banerjee, Z. Abbasi, T. Mukherjee, and G. Varsamopoulos, "Gdcsim: A tool for analyzing green data center design and resource management techniques," in *2011 International Green Computing Conference and Workshops*, pp. 1–8, IEEE, 2011.

[9] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exp.*, vol. 41, no. 1, pp. 23–50, 2011.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[11] OpenAI, "Prompt caching." `https://platform.openai.com/docs/guides/prompt-caching`, 2025. Accessed: 2025.

[12] A. Agrawal, N. Kedia, J. Mohan, A. Panwar, N. Kwatra, B. S. Gulavani, R. Ramjee, and A. Tumanov, "Vidur: A large-scale simulation framework for llm inference," *Proceedings of Machine Learning and Systems*, vol. 6, pp. 351–366, 2024.

[13] J. Cho, M. Kim, H. Choi, G. Heo, and J. Park, "Llmservingsim: A hw/sw co-simulation infrastructure for llm inference serving at scale," in *2024 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 15–29, IEEE, 2024.

[14] A. Iosup, L. Versluis, A. Trivedi, E. V. Eyk, L. Toader, V. van Beek, G. Frascaria, A. Musaafir, and S. Talluri, “The atlarge vision on the design of distributed systems and ecosystems,” in *39th IEEE International Conference on Distributed Computing Systems, ICDCS 2019, Dallas, TX, USA, July 7-10, 2019*, pp. 1765–1776, IEEE, 2019.

[15] G. Andreadis, L. Versluis, F. Mastenbroek, and A. Iosup, “A reference architecture for datacenter scheduling: design, validation, and experiments,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018, Dallas, TX, USA, November 11-16, 2018*, pp. 37:1–37:15, IEEE / ACM, 2018.

[16] M. Jansen, A. Al-Dulaimy, A. V. Papadopoulos, A. Trivedi, and A. Iosup, “The SPEC-RG reference architecture for the compute continuum,” in *23rd IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGrid 2023, Bangalore, India, May 1-4, 2023* (Y. Simmhan, I. Altintas, A. L. Varbanescu, P. Balaji, A. S. Prasad, and L. Carnevale, eds.), pp. 469–484, IEEE, 2023.

[17] G. Andreadis, F. Mastenbroek, V. van Beek, and A. Iosup, “Capelin: Data-driven compute capacity procurement for cloud datacenters using portfolios of scenarios,” *IEEE Trans. Parallel Distributed Syst.*, vol. 33, no. 1, pp. 26–39, 2022.

[18] V. Agrawal, “Energy efficient large language models: Advancements and challenges,” *INTERANTIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING AND MANAGEMENT*, 2025.

[19] S. Moon, J. Kim, J. Kim, S. Hong, J. Cha, M. Kim, S. Lim, G. Choi, D. Seo, J. Kim, H. Lee, H. Park, R. Ko, S. Choi, J. Park, J. Lee, and J. Kim, “LPU: A latency-optimized and highly scalable processor for large language model inference,” *CoRR*, vol. abs/2408.07326, 2024.

[20] S. Ilager, L. F. Briem, and I. Brandic, “GREEN-CODE: optimizing energy efficiency in large language models for code generation,” *CoRR*, vol. abs/2501.11006, 2025.

[21] A. AlZaabi, A. ALamri, H. Albalushi, R. Aljabri, and A. AalAbdulsalam, “Chatgpt applications in academic research: a review of benefits, concerns, and recommendations,” *Biorxiv*, pp. 2023–08, 2023.

[22] J. K. Kim, M. Chua, M. Rickard, and A. Lorenzo, “Chatgpt and large language model (llm) chatbots: The current state of acceptability and a proposal for guidelines on utilization in academic medicine,” *Journal of Pediatric Urology*, vol. 19, no. 5, pp. 598–604, 2023.

[23] J. G. Meyer, R. J. Urbanowicz, P. C. Martin, K. O’Connor, R. Li, P.-C. Peng, T. J. Bright, N. Tatonetti, K. J. Won, G. Gonzalez-Hernandez, *et al.*, “Chatgpt and large language models in academia: opportunities and challenges,” *BioData mining*, vol. 16, no. 1, p. 20, 2023.

[24] W. Liang, Y. Zhang, Z. Wu, H. Lepp, W. Ji, X. Zhao, H. Cao, S. Liu, S. He, Z. Huang, *et al.*, “Mapping the increasing use of llms in scientific papers,” *arXiv preprint arXiv:2404.01268*, 2024.

[25] H. Qin and Z. Li, “A study on enhancing government efficiency and public trust: The transformative role of artificial intelligence and large language models,” *International Journal of Engineering and Management Research*, vol. 14, no. 3, pp. 57–61, 2024.

[26] M. Safaei and J. Longo, “The end of the policy analyst? testing the capability of artificial intelligence to generate plausible, persuasive, and useful policy analysis,” *Digital Government: Research and Practice*, vol. 5, no. 1, pp. 1–35, 2024.

[27] Z. Dai, “Applications and challenges of large language models in smart government-from technological advances to regulated applications,” in *Proceedings of the 2024 3rd International Conference on Frontiers of Artificial Intelligence and Machine Learning*, pp. 275–280, 2024.

[28] N. Corporation, “Nvidia h100 tensor core gpu.” `https://www.nvidia.com/en-us/data-center/h100/`, 2023. Accessed: 2025.

[29] I. E. Agency, “Energy efficiency in data centers and transmission networks,” 2023.

[30] D. Patterson, J. Gonzalez, Q. Le, *et al.*, “Carbon emissions and large neural network training,” *arXiv preprint arXiv:2104.10350*, 2022.

[31] E. P. Agency, “Greenhouse gas emissions from transportation,” 2023.

[32] I. E. Agency, “Netherlands 2023 energy policy review,” 2023.

[33] E. Masanet, A. Shehabi, N. Lei, *et al.*, “Recalibrating global data center energy-use estimates,” *Science*, vol. 367, no. 6481, pp. 984–986, 2020.

[34] A. S. Luccioni, S. Viguier, and A. Ligozat, “Estimating the carbon footprint of bloom, a 176b parameter language model,” *J. Mach. Learn. Res.*, vol. 24, pp. 253:1–253:15, 2023.

[35] A. S. Andrae and T. Edler, “On global electricity usage of communication technology: trends to 2030,” *Challenges*, vol. 6, no. 1, pp. 117–157, 2015.

[36] Y. Shoham, “Why language models became large language models and the hurdles in developing llm-based applications.” `https://www.ai21.com/blog/long-context-yoav-shoham/`, 2024. Accessed: 2025.

[37] K. Chow, Y. Tang, Z. Lyu, A. Rajput, and K. Ban, “Performance optimization in the LLM world 2024,” in *Companion of the 15th ACM/SPEC International Conference on Performance Engineering, ICPE 2024, London, United Kingdom, May 7-11, 2024* (S. Balsamo, W. J. Knottenbelt, C. L. Abad, and W. Shang, eds.), pp. 156–157, ACM, 2024.

[38] A. Biswas, “The long context conundrum: Challenges and innovations in scaling llm memory.” \{https://www.semanticscholar.org/paper/613706b04c5cd3f3cb1b35aba73977ae2c5b6f64, 2025.

[39] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, *et al.*, “Gpt-4 technical report,” *arXiv preprint arXiv:2303.08774*, 2023.

[40] S. Xu, Z. Huang, Y. Zeng, S. Yan, X. Ning, Q. Zhang, H. Ye, S. Gu, C. Shui, Z. Lin, *et al.*, “Hethub: A distributed training system with heterogeneous cluster for large-scale models,” *arXiv preprint arXiv:2405.16256*, 2024.

[41] H. Cheng, R. L. Edwards, W. S. Broecker, G. H. Denton, X. Kong, Y. Wang, R. Zhang, and X. Wang, “Ice age terminations,” *science*, vol. 326, no. 5950, pp. 248–252, 2009.

[42] K. Spence, “Ancient egyptian chronology and the astronomical orientation of pyramids,” *Nature*, vol. 408, no. 6810, pp. 320–324, 2000.

[43] R. Cho, “AI’s Growing Carbon Footprint,” 2023. Accessed 2025-07-06.

[44] D. Patterson, “Good News About the Carbon Footprint of Machine Learning Training,” 2022. Accessed 2025-07-06.

[45] A. A. Chien, L. Lin, H. Nguyen, V. Rao, T. Sharma, and R. Wijayawardana, “Reducing the carbon impact of generative ai inference (today and in 2035),” in *Proceedings of the 2nd workshop on sustainable computer systems*, pp. 1–7, 2023.

[46] A. Iosup, A. Trivedi, J. Donkervliet, L. Versluis, and S. Talluri, *Distributed Systems: Lecture Notes 2019–2020.* 2019. Lecture notes, compiled 3 Dec 2019.

[47] R. Nicolae, “M3sa: Exploring the performance and climate impact of datacenters by multi-model simulation and analysis,” honours program thesis, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands, 2024. Submitted in partial fulfillment of the requirements for the Honors Program.

[48] F. Mastenbroek, T. D. Matteis, V. van Beek, and A. Iosup, “Radice: A risk analysis framework for data centers,” *Future Gener. Comput. Syst.*, vol. 166, p. 107702, 2025.

[49] A. Iosup, “Massivizing computer systems.” Keynote Presentation, February 4, 2021.

[50] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, *et al.*, "On the opportunities and risks of foundation models," *arXiv preprint arXiv:2108.07258*, 2021.

[51] M. Lazuka, A. Anghel, and T. Parnell, "Llm-pilot: Characterize and optimize performance of your llm inference services," in *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–18, IEEE, 2024.

[52] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, *et al.*, "Efficient large-scale language model training on gpu clusters using megatron-lm," in *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pp. 1–15, 2021.

[53] M. Ramponi, "Why language models became large language models and the hurdles in developing llm-based applications." `https://www.assemblyai.com/blog/why-language-models-became-large-language-models`, 2024. Accessed: 2025.

[54] S. Samsi, D. Zhao, J. McDonald, B. Li, A. Michaleas, M. Jones, W. Bergeron, J. Kepner, D. Tiwari, and V. Gadepally, "From words to watts: Benchmarking the energy costs of large language model inference," in *IEEE High Performance Extreme Computing Conference, HPEC 2023, Boston, MA, USA, September 25-29, 2023*, pp. 1–9, IEEE, 2023.

[55] Q.ai, "Microsoft confirms its $10 billion investment into ChatGPT, changing how microsoft competes with google, apple and other tech giants," *Forbes*, January 2023. Accessed: 2025.

[56] T. Guardian, "Three Mile Island nuclear plant to reopen under Microsoft initiative," *The Guardian*, September 2024. Accessed: 2025.

[57] R. Andreoli, J. Zhao, T. Cucinotta, and R. Buyya, "Cloudsim 7g: An integrated toolkit for modeling and simulation of future generation cloud computing environments," *CoRR*, vol. abs/2408.13386, 2024.

[58] S. N. A. Jawaddi and A. B. Ismail, "Integrating openai gym and cloudsim plus: A simulation environment for DRL agent training in energy-driven cloud scaling," *Simul. Model. Pract. Theory*, vol. 130, p. 102858, 2024.

[59] S. K. S. Gupta, A. Banerjee, Z. Abbasi, G. Varsamopoulos, M. Jonas, J. Ferguson, R. R. Gilbert, and T. Mukherjee, "Gdcsim: A simulator for green data center design and analysis," *ACM Trans. Model. Comput. Simul.*, vol. 24, no. 1, pp. 3:1–3:27, 2014.

[60] J. Banks, *Discrete event system simulation.* Pearson Education India, 2005.

[61] Q. Lu, L. Zhu, X. Xu, Z. Xing, and J. Whittle, "Toward responsible AI in the era of generative AI: A reference architecture for designing foundation model-based systems," *IEEE Softw.*, vol. 41, no. 6, pp. 91–100, 2024.

[62] A. Bucaioni, M. Weyssow, J. He, Y. Lyu, and D. Lo, "A functional software reference architecture for llm-integrated systems," *arXiv preprint arXiv:2501.12904*, 2025.

[63] F. Mahr, G. Angeli, T. Sindel, K. Schmidt, and J. Franke, "A reference architecture for deploying large language model applications in industrial environments," in *2024 IEEE 30th International Symposium for Design and Technology in Electronic Packaging (SIITME)*, pp. 19–23, IEEE, 2024.

[64] D. Yang, X. Han, Y. Gao, Y. Hu, S. Zhang, and H. Zhao, "Pyramidinfer: Pyramid KV cache compression for high-throughput LLM inference," in *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024* (L. Ku, A. Martins, and V. Srikumar, eds.), pp. 3258–3270, Association for Computational Linguistics, 2024.

[65] T. A. Team, J. Shan, V. Gupta, L. Xu, H. Shi, J. Zhang, N. Wang, L. Xu, R. Kang, T. Liu, *et al.*, "Aibrix: Towards scalable, cost-effective large language model inference infrastructure," *arXiv preprint arXiv:2504.03648*, 2025.

[66] X. A. Harrison, L. Donaldson, M. E. Correa-Cano, J. Evans, D. N. Fisher, C. E. Goodwin, B. S. Robinson, D. J. Hodgson, and R. Inger, "A brief introduction to mixed effects modelling and multi-model inference in ecology," *PeerJ*, vol. 6, p. e4794, 2018.

[67] G. Myhre, W. Aas, R. Cherian, W. Collins, G. Faluvegi, M. Flanner, P. Forster, Hodnebrog, Z. Klimont, M. T. Lund, *et al.*, "Multi-model simulations of aerosol and ozone radiative forcing due to anthropogenic emission changes during the period 1990–2015," *Atmospheric Chemistry and Physics*, vol. 17, no. 4, p. 2709–2720, 2017.

[68] R. Nicolae, D. Niewenhuis, S. Talluri, and A. Iosup, "On datacenter performance and climate-impact with multi-model simulation and analysis," *Pre-print*.

[69] F. Mastenbroek, T. D. Matteis, V. van Beek, and A. Iosup, "Radice: A risk analysis framework for datacenters," *IEEE Transactions on Cloud Computing*, 2023.

[70] D. Niewenhuis, S. Talluri, A. Iosup, and T. de Matteis, "Footprinter: Quantifying data center carbon footprint," 2024.

[71] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Addison-Wesley Professional, 2021.

[72] "Fair principles," 2016.

[73] M. Flinders and I. Smalley, "What is ai inference?." `https://www.ibm.com/think/topics/ai-inference`, urldate = 2025-07-03, 2024.

[74] IBM, "Tokens and tokenization." `https://www.ibm.com/docs/en/watsonx/saas?topic=solutions-tokens`. IBM watsonx documentation, accessed 2025-07-04.

[75] J. Li, "Life of an inference request (vllm v1): How llms are served efficiently at scale," 2025.

[76] Y. T. Lee, C. R. McLean, and G. Shao, "Neutral information structure for manufacturing simulations: a neutral information model for simulating machine shop operations," in *Proceedings of the 35th Winter Simulation Conference: Driving Innovation, New Orleans, Louisiana, USA, December 7-10, 2003* (S. E. Chick, P. J. Sanchez, D. M. Ferrin, and D. J. Morrice, eds.), pp. 1296–1304, IEEE Computer Society, 2003.

[77] F. Zarai and P. Nicopolitidis, eds., *Modeling and Simulation of Computer Networks and Systems*. Elsevier, 2015.

[78] A. Iosup, G. Andreadis, V. van Beek, M. Bijman, E. V. Eyk, M. Neacsu, L. Overweel, S. Talluri, L. Versluis, and M. Visser, "The opendc vision: Towards collaborative datacenter simulation and exploration for everybody," in *16th International Symposium on Parallel and Distributed Computing, ISPDC 2017, Innsbruck, Austria, July 3-6, 2017* (R. Prodan, F. Pop, and R. Mundani, eds.), pp. 85–94, IEEE, 2017.

[79] International Energy Agency, "Data centres and networks." `[https://www.iea.org/fuels-and-technologies/data-centres-networks](https://www.iea.org/fuels-and-technologies/data-centres-networks)`. Accessed: 2023.

[80] D. Reinsel, J. Gantz, and J. Rydning, "Data age 2025: The evolution of data to life-critical. don't focus on big data," *2*, 2017.

[81] H. He, "Modelling energy consumption in the opendc datacenter simulator for analyzing energy-aware cloud infrastructure," 5 2021. Honours Programme, Research Thesis.

[82] M. Satyanarayanan, W. Gao, and B. Lucia, "The computing landscape of the 21st century," in *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications, HotMobile 2019, Santa Cruz, CA, USA, February 27-28, 2019* (A. Wolman and L. Zhong, eds.), pp. 45–50, ACM, 2019.

[83] S. H. Mortazavi, M. Salehe, C. S. Gomes, C. Phillips, and E. de Lara, "Cloudpath: a multi-tier cloud computing framework," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing, San*

Jose / Silicon Valley, SEC 2017, CA, USA, October 12-14, 2017 (J. Zhang, M. Chiang, and B. M. Maggs, eds.), pp. 20:1–20:13, ACM, 2017.

[84] H. Face, "Kv cache strategies." https://huggingface.co/docs/transformers/en/kv_cache, 2024. Accessed: 2025.

[85] L. Wu, X. Liu, and Q. Liu, "Centroid transformers: Learning to abstract with attention," *CoRR*, vol. abs/2102.08606, 2021.

[86] E. NLP, "The kv cache: Memory usage in transformers." YouTube video, July 2023. Available at: https://www.youtube.com/watch?v=Jt8Xc3pG6cg.

[87] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, *et al.*, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.

[88] European Network of Transmission System Operators for Electricity (ENTSO-E), "Entso-e transparency platform." Official Website, https://transparency.entsoe.eu/, 2025.

[89] C. Malone and C. Belady, "Proceedings of 2006 digital power forum richardson tx," *Metrics to Characterize Data Center IT Equipment Energy Use*, 2006.

[90] V. Avelar, D. Azevedo, A. French, and E. N. Power, "Pue: a comprehensive examination of the metric," *White paper*, vol. 49, 2012.

[91] Climate Neutral Data Centre, "Home." https://www.climateneutraldatacentre.net/, 2023. Accessed: 2024.

[92] Google, "Data center efficiency." https://www.google.com/about/datacenters/efficiency/, 2023. Accessed: 2024.

[93] H. M. Ljungqvist, M. Risberg, A. Toffolo, and M. Vesterlund, "A realistic view on heat reuse from direct free air-cooled data centres," *Energy Conversion and Management: X*, vol. 20, p. 100473, 2023.

[94] J. Summers, A. Kozma, *et al.*, "Holistic cooling at the world's most efficient data center," *Data Center Dynamics*, Oct 2019.

[95] Statista, "Data center average annual pue worldwide." https://www.statista.com/statistics/1229367/data-center-average-annual-pue-worldwide/, 2023. Accessed: 2024.

[96] Statista, "Electricity prices in selected countries." https://www.statista.com/statistics/263492/electricity-prices-in-selected-countries/, 2024. Accessed: 2024.

[97] R. Zhou, Y. Shi, and C. Zhu, "Axpue: Application level metrics for power usage effectiveness in data centers," in *2013 IEEE International Conference on Big Data*, pp. 110–117, IEEE, 2013.

[98] E. E. Agency, "Share of energy consumption from renewable sources," 2023. Accessed: 2024.

[99] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, "Operating systems: Three easy pieces," 2018.

[100] A. De Myttenaere, B. Golden, B. Le Grand, and F. Rossi, "Mean absolute percentage error for regression models," *Neurocomputing*, vol. 192, pp. 38–48, 2016.

[101] Oracle, "Mape (mean absolute percentage error) documentation." Oracle Cloud Infrastructure Documentation, https://docs.oracle.com/en/cloud/saas/planning-budgeting-cloud/pfusu/insights_metrics_MAPE.html, 2024. Accessed: 2025.

[102] J. J. M. Moreno *et al.*, "Using the r-mape index as a resistant measure of forecast accuracy," *Psicothema*, 2013.

[103] N. Rozanski and E. Woods, *Software systems architecture: working with stakeholders using viewpoints and perspectives.* Addison-Wesley, 2012.

[104] IBM, "Milvus overview." IBM Documentation for watsonx.data, https://www.ibm.com/docs/en/watsonx/watsonxdata/2.0.x?topic=overview-milvus, 2024. IBM watsonx.data version 2.0.x documentation.

[105] IBM, "Ibm cloud databases for elasticsearch." IBM Product Documentation https://www.ibm.com/products/databases-for-elasticsearch, 2024. Fully managed Elasticsearch Service offering on IBM Cloud.

[106] OpenAI, "Chatgpt." https://chatgpt.com/, 2025. Accessed: 2025.

[107] J. Dai, X. Pan, R. Sun, J. Ji, X. Xu, M. Liu, Y. Wang, and Y. Yang, "Safe rlhf: Safe reinforcement learning from human feedback," *arXiv preprint arXiv:2310.12773*, 2023.

[108] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, *et al.*, "Training language models to follow instructions with human feedback," *Advances in neural information processing systems*, vol. 35, pp. 27730–27744, 2022.

[109] H. Dong, W. Xiong, B. Pang, H. Wang, H. Zhao, Y. Zhou, N. Jiang, D. Sahoo, C. Xiong, and T. Zhang, "Rlhf workflow: From reward modeling to online rlhf," *arXiv preprint arXiv:2405.07863*, 2024.

[110] O. Erdogan, "Eurogpt: Open source and privacy-conscious alternative to chatgpt enterprise." https://www.ubicloud.com/blog/eurogpt-open-source-and-privacy-conscious-alternative-to-chatgpt-enterprise, 2024. Accessed: 2025.

[111] Databricks, "Implementing llm guardrails for safe and responsible generative ai deployment on databricks," 2025. Accessed 2025.

[112] OpenAI, "Openai API reference." https://platform.openai.com/docs/api-reference/introduction, 2024. Online documentation for the OpenAI API.

[113] OpenAI, "Infrastructure for deep learning." https://openai.com/index/infrastructure-for-deep-learning/, 2016. Technical blog post detailing infrastructure approaches.

[114] Microsoft, "Azure openai on your data." https://learn.microsoft.com/en-us/azure/ai-services/openai/concepts/use-your-data, 2025. Accessed: 2025.

[115] A. Patke, D. Reddy, S. Jha, C. Narayanaswami, Z. Kalbarczyk, and R. Iyer, "Hierarchical autoscaling for large language model serving with chiron," *arXiv preprint arXiv:2501.08090*, 2025.

[116] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

[117] NVIDIA, "Nvidia tensorrt." https://developer.nvidia.com/tensorrt, 2023. Accessed: 2025.

[118] Z. Zhou, X. Ning, K. Hong, T. Fu, J. Xu, S. Li, Y. Lou, L. Wang, Z. Yuan, X. Li, S. Yan, G. Dai, X. Zhang, Y. Dong, and Y. Wang, "A survey on efficient inference for large language models," *CoRR*, vol. abs/2404.14294, 2024.

[119] Y. He, J. Fang, F. R. Yu, and V. C. M. Leung, "Large language models (llms) inference offloading and resource allocation in cloud-edge computing: An active inference approach," *IEEE Trans. Mob. Comput.*, vol. 23, no. 12, pp. 11253–11264, 2024.

[120] OpenAI, "Scaling Kubernetes to 7,500 nodes." https://openai.com/index/scaling-kubernetes-to-7500-nodes/, 2021. Accessed: 2025.

[121] OpenAI, "Announcing the Stargate project." https://openai.com/index/announcing-the-stargate-project/, 2025. Accessed: 2025.

[122] OpenAI, "New funding to build towards AGI." https://openai.com/index/march-funding-updates/, 2025. Accessed: 2025.

[123] Microsoft, "Data source - azure cosmos db for mongodb vcore." https://learn.microsoft.com/en-us/azure/ai-services/openai/references/cosmos-db?tabs=python, 2024. Accessed: 2025.

[124] IBM, "Ibm watson." https://www.ibm.com/watson, 2025. [Online; accessed 6 July 2025].

[125] IBM, “Getting Started with watsonx Assistant - IBM Cloud Docs.” `https://cloud.ibm.com/docs/watson-assistant`, 2025. Accessed: 2025. Official IBM Cloud documentation for Watson Assistant.

[126] IBM, “Watsonx apis.” `https://www.ibm.com/docs/en/watsonx/saas?topic=tutorials-watsonx-apis`, 2025. Accessed: 2025.

[127] IBM, “Governing assets with watsonx.governance.” `https://www.ibm.com/docs/en/watsonx/saas?topic=governing-ai`, 2023. Accessed: 2025.

[128] IBM, “Watson machine learning.” `https://www.ibm.com/docs/en/software-hub/5.1.x?topic=services-watson-machine-learning`, 2023. Version: 5.1.2; Accessed: 2025.

[129] IBM Cloud Documentation, “Gen ai pattern for watsonx on ibm cloud.” `https://cloud.ibm.com/docs/pattern-genai-rag?topic=pattern-genai-rag-genai-pattern`, 2024. Accessed: 2025.

[130] IBM Cloud Documentation, “Workload placement planning for ibm cloud pak for integration.” `https://www.ibm.com/docs/en/cloud-paks/cp-integration/16.1.0?topic=planning-workload-placement`, 2025. Accessed: 2025.

[131] H. Malik, S. Chou, and E. Saydam, “Ibm partners with elasticsearch to deliver conversational search with watsonx assistant.” `https://www.elastic.co/blog/ibm-elasticsearch-partnership-conversational-search-watsonx-assistant`, 2024. Accessed: 2025.

[132] IBM Cloud Documentation, “Milvus overview for ibm watsonx.data.” `https://www.ibm.com/docs/en/watsonx/watsonxdata/2.0.x?topic=overview-milvus`, 2025. Accessed: 2025.

[133] IBM Cloud Documentation, “What is milvus? - ibm.” `https://www.ibm.com/think/topics/milvus`, 2025. Accessed: 2025.

[134] X. Yan and Y. Ding, “Are we there yet? a measurement study of efficiency for llm applications on mobile devices,” *arXiv preprint arXiv:2504.00002*, 2025.

[135] M. Xu, D. Niyato, and C. G. Brinton, “Serving long-context llms at the mobile edge: Test-time reinforcement learning-based model caching and inference offloading,” *CoRR*, vol. abs/2501.14205, 2025.

[136] Google, “Gemini AI.” `https://gemini.google.com/`, 2025. Accessed: 2025.

[137] Databricks Documentation, “Introduction to databricks notebooks,” 2025. Accessed 2025.

[138] Databricks Documentation, “Deploy models using mosaic ai model serving,” 2025. Accessed 2025.

[139] U. Cubukcu, “Lantern on ubicloud: Build ai applications with postgresql.” `https://www.ubicloud.com/blog/build-ai-apps-with-postgresql`, 2024. Accessed: 2025.

[140] J. Li, “Life of an inference request (vllm v1): How llms are served efficiently at scale.” `https://www.ubicloud.com/blog/life-of-an-inference-request-vllm-v1`, 2025. Accessed: 2025.

[141] Databricks, “Scalable kubernetes upgrade using operators,” 2024. Accessed 2025.

[142] Databricks Documentation, “Model inference using tensorflow and tensorrt,” 2025. Accessed 2025.

[143] Databricks, “Vector search,” 2025. Accessed 2025.

[144] Databricks Documentation, “Databricks documentation,” 2025. Accessed 2025.

[145] Meta AI, “Introducing llama 3.1: Our most capable models to date.” `https://ai.meta.com/blog/meta-llama-3-1/`, 2024. Accessed 2025.

[146] Databricks Documentation, “Mosaic ai capabilities for generative ai apps,” 2025. Accessed 2025.

[147] A. M. Lasa, S. Talluri, and A. Iosup, “A reference architecture for datacenter scheduler programming abstractions: Design and experiments (work in progress paper),” in *Proceedings of the International Conference on Performance Engineering, Coimbra, Portugal, April, 2023*, 2023.

[148] V. A. Korthikanti, J. Casper, S. Lym, L. McAfee, M. Andersch, M. Shoeybi, and B. Catanzaro, "Reducing activation recomputation in large transformer models," in *Proceedings of the Sixth Conference on Machine Learning and Systems, MLSys 2023, Miami, FL, USA, June 4-8, 2023* (D. Song, M. Carbin, and T. Chen, eds.), mlsys.org, 2023.

[149] OpenAI, "Introducing deep research." `https://openai.com/index/introducing-deep-research/`, Feb 2025. Accessed: 2025.

[150] J. Ousterhout, *A Philosophy of Software Design.* Yaknyam Press, 1 ed., 2018.

[151] Baseten, "A guide to llm inference and performance." `https://www.baseten.co/blog/llm-transformer-inference-guide/`.

[152] P. G. Recasens, F. Agullo, Y. Zhu, C. Wang, E. K. Lee, O. Tardieu, J. Torres, and J. L. Berral, "Mind the memory gap: Unveiling gpu bottlenecks in large-batch llm inference," *arXiv preprint arXiv:2503.08311*, 2025.

[153] The Apache Software Foundation, "Apache parquet." `https://parquet.apache.org/`, 2024.

[154] R. F. da Silva, A. Orgerie, H. Casanova, R. Tanaka, E. Deelman, and F. Suter, "Accurately simulating energy consumption of i/o-intensive scientific workflows," in *Computational Science - ICCS 2019 - 19th International Conference, Faro, Portugal, June 12-14, 2019, Proceedings, Part I* (J. M. F. Rodrigues, P. J. S. Cardoso, J. M. Monteiro, R. Lam, V. V. Krzhizhanovskaya, M. H. Lees, J. J. Dongarra, and P. M. A. Sloot, eds.), vol. 11536 of *Lecture Notes in Computer Science*, pp. 138–152, Springer, 2019.

[155] X. Fan, W. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *34th International Symposium on Computer Architecture (ISCA 2007), June 9-13, 2007, San Diego, California, USA* (D. M. Tullsen and B. Calder, eds.), pp. 13–23, ACM, 2007.

[156] L. Hirth, J. Mühlenpfordt, and M. Bulkeley, "The entso-e transparency platform–a review of europe's most ambitious electricity data platform," *Applied energy*, vol. 225, pp. 1054–1067, 2018.

[157] N. de Lama Sanchez, P. Haase, D. Roman, and R. Prodan, "Boosting the impact of extreme and sustainable graph processing for urgent societal challenges in europe graph-massivizer: A horizon europe project," in *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*, pp. 233–238, 2023.

[158] C. Dilmegani, "Cloud gpus for deep learning: Availability & price / performance," *AIMultiple Research*, July 2025. Accessed: 2025.

[159] Statista, "Most used programming languages among developers worldwide." `https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/`, 2025. Accessed: 2025.

[160] JetBrains, "Kotlin programming language." `https://www.jetbrains.com/opensource/kotlin/`, 2024. Accessed: 2025.

[161] Google Developers, "Contributing to blockly: Getting started with commits." `https://developers.google.com/blockly/guides/contribute/get-started/commits`, 2024. Accessed: 2025.

[162] Google Developers, "Contributing to blockly: Writing a good pull request." `https://developers.google.com/blockly/guides/contribute/get-started/write_a_good_pr`, 2024. Accessed: 2025.

[163] M. Fowler, "Continuous integration." `https://martinfowler.com/articles/continuousIntegration.html`, 2006. Accessed: 2025.

[164] J. Stojkovic, C. Zhang, Í. Goiri, J. Torrellas, and E. Choukse, "Dynamollm: Designing llm inference clusters for performance and energy efficiency," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1348–1362, IEEE, 2025.

[165] Y. Wang, Y. Chen, Z. Li, X. Kang, Z. Tang, X. He, R. Guo, X. Wang, Q. Wang, A. C. Zhou, and X. Chu, "Burstgpt: A real-world workload dataset to optimize llm serving systems," 2024.

[166] R. Pan, Z. Wang, Z. Jia, C. Karakus, L. Zancato, T. Dao, R. Netravali, and Y. Wang, "Marconi: Prefix caching for the era of hybrid llms," *arXiv preprint arXiv:2411.19379*, 2024.

[167] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica, "Judging llm-as-a-judge with mt-bench and chatbot arena," in *Advances in Neural Information Processing Systems*, vol. 36, pp. 46595–46623, 2023.

[168] ShareGPT Team, "Sharegpt: Share your wildest chatgpt conversations with one click." `https://sharegpt.com`, 2024. Accessed: 2025.

[169] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press, "Swe-agent: Agent-computer interfaces enable automated software engineering," *arXiv preprint arXiv:2405.15793*, 2024.

[170] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. R. Narasimhan, "Swe-bench: Can language models resolve real-world github issues?," in *International Conference on Learning Representations*, 2024.

[171] NVIDIA Corporation, "Nvidia a10 tensor core gpu." `https://www.nvidia.com/en-us/data-center/products/a10-gpu/`, 2025. Accelerated graphics and video with AI for mainstream enterprise servers. Accessed 2025-07-03.

[172] M. AI, "Meta llama 3.1 8b." `https://huggingface.co/meta-llama/Llama-3.1-8B`, 2024. Llama 3.1 is licensed under the Llama 3.1 Community License, Copyright © Meta Platforms, Inc. All Rights Reserved. Model release date: July 23, 2024.

[173] H. Bal, D. Epema, C. De Laat, R. Van Nieuwpoort, J. Romein, F. Seinstra, C. Snoek, and H. Wijshoff, "A medium-scale distributed system for computer science research: Infrastructure for the long term," *Computer*, vol. 49, no. 5, pp. 54–63, 2016.

[174] NVIDIA Corporation, "Nvidia rtx a4000 graphics card." `https://www.nvidia.com/en-us/products/workstations/rtx-a4000/`, 2025. Single-slot professional GPU with real-time ray tracing and AI acceleration. Accessed 2025-07-03.

[175] NVIDIA Corporation, "Nvidia rtx a6000 graphics card." `https://www.nvidia.com/en-us/products/workstations/rtx-a6000/`, 2025. 48 GB GDDR6, third-generation NVLink, for advanced visualization AI workloads. Accessed 2025-07-03.

[176] NVIDIA Corporation, "Nvidia a100 tensor core gpu." `https://www.nvidia.com/en-us/data-center/a100/`, 2025. Ampere-architecture accelerator for AI, HPC, and data analytics. Accessed 2025-07-03.

[177] NVIDIA Corporation, "Nvidia system management interface (nvidia-smi) documentation." `https://docs.nvidia.com/deploy/nvidia-smi/index.html`, 2025. Accessed 2025-07-03.

[178] L. D. Vitis, "loremipsum: A lorem ipsum text generator." `https://loremipsum.readthedocs.io/`, 2011–2014. Version 1.0.4. GNU General Public License v3 or later.

[179] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, S. Edunov, T. Scialom, and et al., "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[180] IBM Research, "Granite-20b foundation model." `https://huggingface.co/ibm-granite/granite-20b-code-base-8k`, 2024. Version granite-20b-code-base-8k.

[181] MosaicML, "MPT-30B: A 30-billion-parameter open-source transformer." `https://huggingface.co/mosaicml/mpt-30b`, 2023.

[182] J. Shim, G. Seo, C. Lim, and Y. Jo, "Tooldial: Multi-turn dialogue generation method for tool-augmented language models," *arXiv preprint arXiv:2503.00564*, 2025.

[183] Y. Deng, N. Zhao, and X. Huang, "Early chatgpt user portrait through the lens of data," in *2023 IEEE International Conference on Big Data (BigData)*, pp. 4770–4775, IEEE, 2023.

[184] L. Zheng, W.-L. Chiang, Y. Sheng, T. Li, S. Zhuang, Z. Wu, Y. Zhuang, Z. Li, Z. Lin, E. P. Xing, *et al.*, “Lmsys-chat-1m: A large-scale real-world llm conversation dataset,” *arXiv preprint arXiv:2309.11998*, 2023.

[185] H. McNichols and A. Lan, “The studychat dataset: Student dialogues with chatgpt in an artificial intelligence course,” *arXiv preprint arXiv:2503.07928*, 2025.