International Journal of Science and Research Archive

eISSN: 2582-8185

Cross Ref DOI: 10.30574/ijsra

Journal homepage: https://ijsra.net/

(REVIEW ARTICLE)

Check for updates

Optimizing AI Model Inference Performance with Dynamic Profiling

Ankush Jitendrakumar Tyagi *

University of Texas at Arlington, Texas, USA.

International Journal of Science and Research Archive, 2025, 16(01), 2266-2275

Publication history: Received on 31 May 2025; revised on 18 July 2025; accepted on 27 July 2025

Article DOI: https://doi.org/10.30574/ijsra.2025.16.1.2066

Abstract

Deep neural networks and Artificial Intelligence (AI) models have shown great success in areas that include computer vision, natural language processing, and autonomous systems. Yet, their application in real-world tasks is typically limited by inference performance drawbacks, in particular, when the specialized cutting-edge devices are needed to complete such tasks in real time and on resource-constrained devices. The main issue with the requirement to scale, efficient, and responsive AI systems is the key attention paid to the inference performance optimisation. Dynamic profiling, or the process of analysing AI models and system performance in real-time as they execute, has become a critical technique not only as a means to detect locations where performance is impeded but to inform the process of performance optimisation at runtime. In contrast to static profiling which performs an analysis before execution of specific and prepared traces of the executable (static profiling uses the pre-execution information about the program to perform an analysis of it), dynamic profiling allows a more dynamic and fine-grained inspection of problems like inefficiencies in memory access, imbalances in compute utilisation, layer-resolution latency, and power consumption. Dynamic performance tracing, profiling, and tools and frameworks such as TensorRT, Intel VTune, NVIDIA Nsight, and PyTorch Profiler enjoy wide support across the diverse hardware platforms, including CPU, GPU, and edge accelerator, with full support across platforms. These tools can offer useful information to guide fine-grained optimisations like operator fusion, quantisation, memory and computation scheduling, and replication strategies. Notably, with the seamless coupling of dynamic profiling to automated deployment pipelines, AI systems can dynamically optimise themselves at runtime and respond well to variations in workloads and system constraints. It helps achieve intelligent self-optimising AI applications that are also able to be kept at production level performance. As dynamic profiling is integrated into the AI model lifecycle, it allows continuous performance tracking and a sign-and-iterate cycle, hence facilitating the delivery of scalable, energy-efficient, and high-throughput AI approaches at scale in the cloud as well as at the edge. This paper will demonstrate that dynamic profiling is a very important technique to overcome the performance issues and drive the best future of AI deployment.

Keywords: Dynamic Profiling; Inference Optimisation; Real-Time Performance; Edge AI; Profiling Frameworks; Adaptive Deployment

1. Introduction

The use of Artificial Intelligence (AI) and, mainly, deep learning has revolutionized many fields, such as the classification of images, speech recognition, natural language processing, and autonomous control systems. These have led to more dense, complex models with heavy computation and memory demands, particularly driven by the use of deep neural networks (DNNs). Nevertheless, the move into deployment, particularly in a real-time or resource-limited setting, introduces many difficulties, mainly in terms of inference performance [1][2].

* Corresponding author: Ankush Jitendrakumar Tyagi

The AI lifecycle involves the step of inference, in which a trained AI model can be applied to new data. Although the training may be a computationally heavy process and may therefore attract much attention in publications, the inference stage is what determines the user experience of a production system. Real-time surveillance applications, voice assistants, self-driving cars, and mobile apps require high-performance model inference, as such solutions require fast, efficient, and accurate model inference. Such requirements need to be achieved in the existence of limitations on latency, power consumption, memory bandwidth, and compute resources [3][4]. In this respect, optimising inference performance comes to the fore. Nevertheless, static profiling will naturally be restricted by its inability to respond to dynamic conditions that are characteristic of real-world deployments. The solution can be presented by dynamic profiling, which enables one to profile the performance at run times, and also trace real-time system behaviour and reveal performance bottlenecks that would otherwise go undetected using a static approach [5]. Dynamic profiling allows developers to acquire a view into how AI models operate at deployment on different hardware backends: whether on widespread, wide-purpose CPUs, high-power GPUs, or low-power edge accelerators. Using the real-time information on compute usage, memory bandwidth, power draw, and kernel-level latencies, the practitioner is enabled to detect inefficiencies directly affecting inference speed and accuracy [6].

In addition, tools such as NVIDIA Nsight, Intel VTune, TensorRT, and PyTorch Profiler have been introduced to facilitate such an analysis. They can be used to instrument specific runtime in detail, can be integrated with multiple platforms, and to automatically suggest optimisation steps [7][8]. The knowledge gained by these profiling mechanisms enables the developers to make optimisations like operator fusion, precision reduction (e.g., quantisation), memory prefetching, and model pruning, which can be used to speed up and make efficient inference [7]. Dynamic profiling can be integrated into automated pipelines, especially in the case of continuous integration and deployment (CI/CD) of AI pipelines. This enables adaptive tuning and can make AI models more responsive to variations in distributions of input data, equipment, or other environmental variables without having to involve manual tuning. By that, dynamic profiling not only provides performance boosting but leads to the creation of robust, self-sustaining AI systems [8].

This paper investigates one solution to these issues in terms of how to consider dynamic profiling in AI inference workflows and what the benefits and tool support look like, its practical applications, and future possibilities.

2. Challenges in AI Model Inference Across Diverse Deployment Scenarios

Time and again, deep learning models have recorded amazing performance in laboratory environments and benchmark tests; however, when applied to real-world environments, the performance tends to be uneven, dependent on a myriad of architectural, environmental, and system-level limitations. The process of AI inference, in which the models are fed on real-time inputs and make predictions, is subject to various intrinsic variables such as the constraints of hardware, the diversity of the climate, and platform variance, which may hamper latency, efficiency, and accuracy [9]. Real-time latency requirements are one of the hottest deployment concerns. Programs like autonomous vehicles, medical imaging, and voice interaction systems require inference times measured nearly instantaneously, in many cases in milliseconds. However, deeper and larger models inherently consume more computation, and this trades off latency and model complexity [10][11]. This problem is complicated further in this production environment, where single-inference instances are run in real time as opposed to the more production-associated batch-inference optimisations of training environments. Without dynamic system feedback, developers cannot easily come up with the best balance of accuracy versus latency [12]. AI model inference deployment faces multiple challenges, ranging from hardware constraints and latency issues to environment variability and model drift. Addressing these complexities is critical for ensuring robust, scalable, and secure AI solutions across diverse application domains, as shown in Figure 1.

The performance of inference is further made tricky by hardware diversity. Trained models are leveraged over a wide variety of platforms, including GPUs, TPUs, and CPUs in the cloud, DSPs and ASICs on mobile and embedded devices. These devices possess distinctive computing power, memory structure, and bandwidth ability and drawbacks, which seriously affect the performance of AI solutions [13]. What is an optimisation that benefits one platform can harm another, especially when multiple compilers and backends are in use and have differing tools and requirements [14]. The most common hindrances will be CPU resource limitations, such as memory and bandwidth, particularly in edge-bound products such as Raspberry Pi or the NVIDIA Jetson Nano. These devices tend to have cache thrashing or paging that significantly raises the latency and drops throughput [15]. In addition, bottlenecks in memory further compound the energy cost, a crucial consideration in battery-powered systems and mobile applications; therefore, memory allocation during inference is in need of comprehension and optimisation [16].

Energy efficiency is another important parameter, particularly in mobile, IoT, and embedded systems in applications where power budgets are constrained. The number of power/energy spent during inference will vary with the computational workload of a model and the frequency of accesses to the model parameters, and state-of-the-art

profiling techniques (especially at run time) usually do not have sufficient resolution to capture such dynamics [17]. Although the application of strategies such as quantisation, pruning, and application of low-power accelerators has been embraced to lower the cost of energy, the optimisation is, in most cases, sensitive to real-time profiling data to be efficiently applied without witnessing a decline in precision [18]. The complexity of models is also a burden on inference. As more types of advanced architectures come into use, such as transformers, models are now made up of billions of parameters, including models like BERT and GPT-3, and require a lot of memory bandwidth and compute capability to infer their performance. Their complex layer arrangements and attention are not optimised as easily as across platforms with different compute strengths [19]. Intricate networks without the dynamic profiling, minor inefficiencies can balloon into big performance upheavals, e.g., tensor movement or the poor-scheduling of layers [20].

Moreover, there is a variable distribution of input, as well as data dependence, which presents another aspect of uncertainty. Performance of the inference is extremely susceptible to the input size and complexity; e.g., vision models can be slow when presented with high-resolution images, and NLP models can be slower with longer or more ambiguous sentences. Data unpredictability may be a concern in production systems operating in open environments, where inconsistent latency or failure mode in open systems may require real-time remedial mechanisms acting under the direction of profiling [21][22]. One more systematic challenge stems in the nature of the siloed view through the AI model lifecycle. Performance testing is usually restricted to the development environment or the staging environment, where controlled samples and conditions can be employed. This creates a blind spot post-deployment of the models, which is difficult to debug and commonly leads to unexpected performance problems only observed when the system is under a demanding production workload [23]. Dynamic profiling can be used to bridge this gap by providing in-situ monitoring and diagnosis of the inference behavior, with adaptive changes being implemented on the fly by the developer of the program, such as modification of kernel launch parameters or re-ordering execution graphs in response to conditions [24].

Finally, the profiling tools ecosystem is very fragmented. Various AI frameworks, such as TensorFlow, PyTorch, and ONNX Runtime, have different toolchains and APIs, and hardware vendors have their own respective profiling tools that are frequently incompatible. This division poses a large learning curve to the developers and makes it considerably costly to achieve cross-platform performance parity. In order to address this, unified cross-platform profiling frameworks that promise simplified interfaces and a standard method of output are becoming a crucial part of scalable AI deployment. In their absence, sophisticated profiling and optimisation methods are still confined, particularly in large-scale and multi-device production systems.

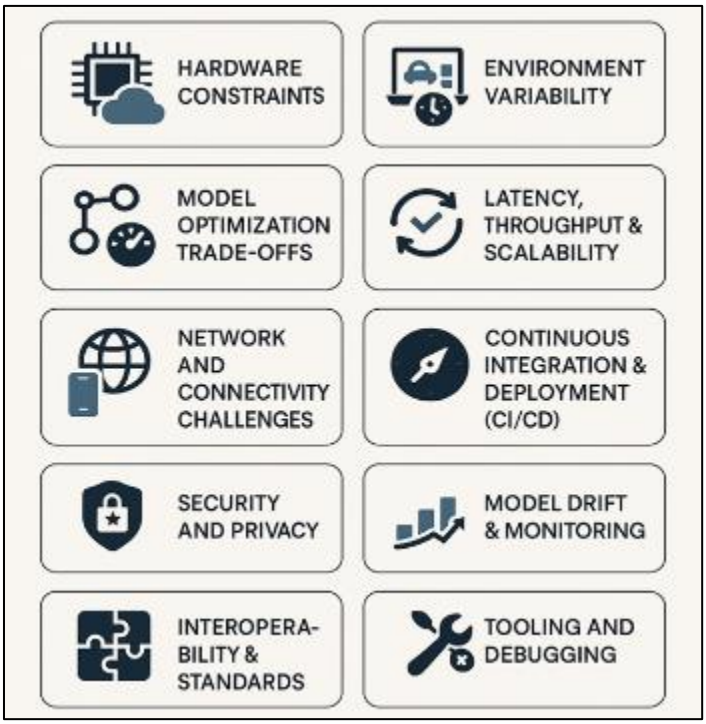

Figure 1 Challenges in AI Model Inference Across Diverse Deployment Scenarios

3. Understanding Dynamic Profiling and Its Advantages Over Static Methods

AI inference models tend to exhibit significant shifts in behaviour when exposed to runtime variations in data inputs, hardware limitations, and software stack performance. Consequently, their performance becomes highly sensitive to these factors, rendering static optimisation techniques inadequate for ensuring consistent effectiveness in real-world deployments. Meanwhile, the limitations of static profiling, which tests models based on pre-configured inputs or offline conditions, prevent it from being useful to capture dynamic loads such as memory contention, CPU throttling, or thermal constraints, and in general make it less suitable in operational environments. Conversely, dynamic profiling is a run-time performance tracking method that can collect real-time execution information, e.g., instruction throughput, memory access latency, cache performance, power consumption, and related thermal information, that provides a more realistic and invokable picture of the model behaviour when deployed [25]. Built into AI runtimes and inference engines, dynamic profiling techniques monitor layer-by-layer performance and reveal bottlenecks in data transfer, kernel performance, and care and utilisation of compute units, allowing fine-grained tuning in real-time to atypical situations like variable batch size, design constraints, or changes in input data volumes [26]. This makes static profiling fundamentally inflexible; it provides no feedback loop after a model is deployed, nor has the propensity to be structured around idealised test cases not reflecting operational workloads and thus yield optimisation strategies that are not effective at generalising in production [27]. By contrast, dynamic profiling facilitates ongoing system observation, which enables inference workloads to be contextually adaptive and able to sustain performance across a variety of conditions. Such a strategy has important benefits, e.g., real-time bottleneck identification, where latency-intensive operations are immediately detected, and corrective measures such as kernel fusion or memory layout refactoring can be undertaken [28]. In addition, dynamic profiling can provide cross-layer insight by displaying interdependencies and data flow patterns between layers in architectures with more complex pathways, such as residual networks or attention mechanisms, which is essential to the detection of cascading latency or execution time bottlenecks [29]. Dynamic profiling also supports hardware-aware optimisation; e.g., a profiler could determine whether a program is fully utilising available features such as Tensor Cores or AVX-512 instructions, and suggest ways to change tensor shapes or data types to exploit the additional compute resources. Dynamic profiling in energy-constrained environments is also able to track thermal load and power consumption to actively regulate system performance parameters before incurring any thermal throttling, enabled by tools like Intel Power Gadget or NVIDIA NVML [30]. Dynamic profiling can support runtime adaptability, an especially powerful feature in which models dynamically adjust inference paths in real time (e.g., by executing lower-precision operations or smaller batch size) to maintain performance within power or resource limits [31]. This has been found helpful in many disciplines: autonomous vehicles use profiling feedback to optimise model complexity in response to environmental density or temperature; medical devices trade off inference speed with diagnostic accuracy; edge devices of the IoT customise inference to limited compute resources; and cloud systems such as AWS Lambda or Google Cloud Run use profiling to optimise autoscaling and resource allocation [32][33]. Notably, dynamic profiling can be used not only to analyse but also to automate, with profiling output being used to drive model retraining or graph rewriting, or compilation optimisations by automation [34]. These strategies are already being used in benchmarks such as the MLPerf Inference Suite and inference engines TensorRT and OpenVINO, which also have profilers built in to guide deployment-time graph transformations based on signals at runtime as opposed to heuristics. In conclusion, though profiling, of the static variety, is pertinent to the early stage of the development cycle, it cannot adapt well to the circumstances of deploying inferences. Dynamic profiling tends to fill the gap between model design and operational robustness by enabling real-time, hardware-aware, precise optimisation, thus enabling large and responsive AI systems to operate in production.

4. Profiling Tools and Frameworks for Real-Time AI Performance Monitoring

The complexity of AI workloads and the rise in diversity of deployment environments have seen the emergence of an ecosystem of real-time profiling tools and frameworks. These profiling tools are core in dynamic profiling with in-depth insight into the inference performance on CPUs, GPUs, and edge accelerators. The most frequently deployed are the framework-integrated profilers, like those in PyTorch and TensorFlow. As an example, the PyTorch Profiler provides strong functionality to trace compute times per operator, monitor the CPU/GPU usage and memory consumption, and graphically represent the run timeline via TensorBoard and the Chrome Trace. It is also quite efficient with NVIDIA GPUs, where it can point to CUDA kernel bottlenecks, tensor misalignments, and inefficiencies in transfer between the host and the device. In a parallel manner, the TensorFlow profiler, managed through TensorBoard, provides support to explore kernel timing charts, input pipeline debugging and cross-device profiling. It supports a Performance Dashboard where efficiency of inferences is rated, and tuning proposals are given [35,36]. Effective real-time performance monitoring of AI models is critical for maintaining accuracy, efficiency, and responsiveness in deployment. Tools like NVIDIA Nsight, TensorBoard, TensorFlow Profiler, PyTorch Profiler, TensorRT Inference Server, and Prometheus provide valuable insights into system bottlenecks, GPU usage, and inference latency as shown in Figure 2.

In addition to framework-specific tools, there are also profilers available through hardware vendors, which provide low-resolution information at a system level, at the cost of revealing quite low-level platform behaviour, e.g., cache performance, voltage, and thermal behaviour. Nsight Systems and Nsight Compute, made by NVIDIA, are a good example. Nsight Systems provides an overview of GPU-CPU interaction and which synchronisation delays, whereas Nsight Compute considers individual CUDA kernel behaviour separately with warp-by-warp execution, instruction throughput, and memory access efficiency [37]. The Intel VTune Profiler, in its turn, is popular for the profiling of inference on the CPU, and it can monitor instruction-per-cycle (IPC) metrics, memory latency, thread core affinities, and integrates with the OpenVINO to support tuning AI-specific behaviours like INT8 quantisation and the fusion of operators. This gives rather similar features to the AMD hardware, such as performance counters and power estimation, but also cache analysis of Ryzen and EPYC processors, so, like with ROCm inference, it is surplus to requirements [38].

As AI models deploy more frequently at the edge, profiling tools that work well in an embedded environment are also starting to play more of a role. Arm Streamline has been developed specifically to profile on mobile and edge SoCs, including the understanding of real-time latencies, thermal and power systems, and integration with models compiled using Arm NN and Ethos-N accelerators. It offers the possibility to correlate CPU and GPU metrics, therefore, a competitive option to profile compact inference engines on mobile or embedded systems. More crudely, Google also offers its Coral Profiler tool, which specifically supports Edge TPU devices, but still has valuable runtime statistics like tensor latency and memory consumption during inference [39].

In clouds, large platforms like AWS, Google Cloud, and Azure support such services as managed profiling and observability via components that can be seamlessly integrated into inference endpoints. AWS SageMaker Debugger, as an example, gathers such metrics as GPU usage and execution latency and displays them in CloudWatch dashboards to track operations. Profiler by Google Cloud and Machine Learning Profiler by Azure provide comparable functionality where it monitor CPU/GPU utilization and make it possible to detect anomalies using telemetry-based feedback loops. These cloud-native profilers can be used to make intelligent auto-scaling and load-balancing choices using the inference real-time metrics.

Benchmarking suites like MLPerf Inference have been developed to facilitate standardised assessment, for example, by utilising dynamic profiling across various tasks (including image classification and speech recognition). Such benchmarks give a comparison perspective on performance in a production-like environment. The AI Benchmark suite is especially prevalent on Android, where it profiles models running on TensorFlow Lite or ONNX runtimes, logging time, the amount of power, and allocated memory [40].

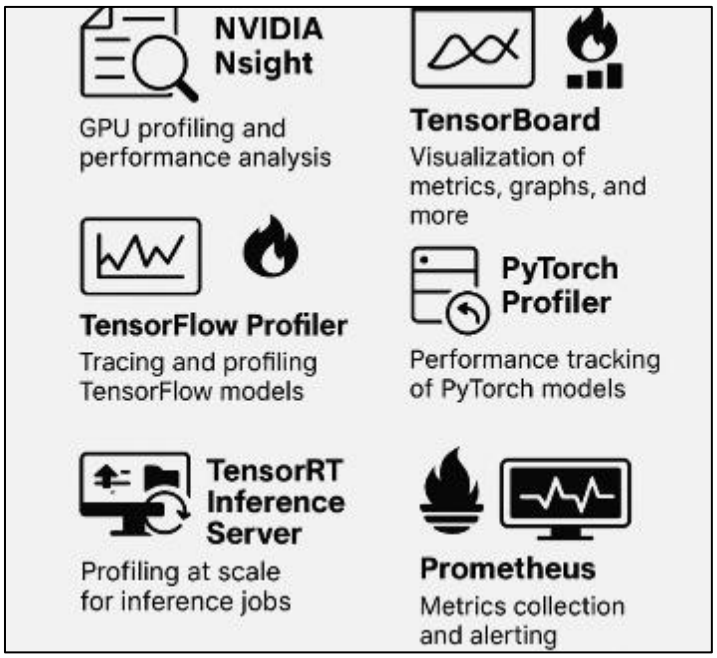

Figure 2 Profiling Tools and Frameworks for Real-Time AI Performance Monitoring

Profiling data display is not easy, and thus, visualisation tools are important. The most popular tool for visualising execution traces and profiling graphs is still TensorBoard, although there is also support for interpreting JSON traces

exported by other tools (such as Nsight and PyTorch) in Chrome Trace Viewer. Another method of visualising stack traces of functions to find hotspots within performance-sensitive programs is flame graphs, popularized by Brendan Gregg [41].

As a brief conclusion, it is quite clear that the choice of a suitable profiling tool is affected by a variety of properties, which include hardware type, deployment environment, and the performance goal of the AI system. Profilers integrated into the frameworks are very good at providing preliminary guidance through the initial development and debugging phases, but vendor-specific tools are necessary to optimize a production deployment. Lightweight profiling tools that scale well and can be used to continuously monitor a cloud or edge environment at little cost in terms of performance are of use. The tools will play imperative roles in ensuring that models run effectively in diverse workload and resource conditions as AI systems continue to become complex, with real-time responsiveness becoming a requisite. It is based on the insights they yield and which are the basis of runtime inference optimisations that are detailed in a subsequent section.

5. Extracting Bottleneck Insights for Runtime Optimisation

Dynamic profiling tools can provide an abundant output in performance data at run time, but to make productive use of this, the data must be structured into interpretable data through a systematic interpretation. Bottlenecks in the context of AI inference are model components or execution stages that slow down the execution of the model overall disproportionately, relative to other components or stages. That can be as a result of unoptimal memory access, underutilised compute resources, high data transfers, or inefficient execution of certain layers. Dynamic profiling allows engineers to segment and trace these performance bottlenecks by examining execution traces and comparing them with the performance of other units of measure, like memory bandwidth, kernel launch times, or GPU warp efficiency. As a practice, two forms of bottlenecks can be established. Compute bottlenecks indicate that intensive processes such as matrix multiplications or convolution operations take most of the time, and are usually linked with poor kernel choice or ineffective use of SIMD/Tensor Cores. The slow speed causes latency, poor locality, or cache misses that slow the access of the model weights and intermediate tensors, creating memory bottlenecks. Transfer overheads such as unneeded host-device copy or off-chip communication, in particular, can be an issue in GPU and TPU applications. On the same note, some of these layers might be computationally intensive or poorly parallelised, thus making it inefficient to execute a layer. Another large problem, hardware underutilisation, is usually caused by ineffective batching, unbalancing of the distribution of CPU and GPU, or unoptimal threading calculations, often leaving computational resources unusable [42,43]. These challenges can be mitigated only by teaching developers to read important profiling statistics. An example of such information is the launch time in the kernel that indicates pipeline design faults or delays in thread dispatching. Poor low floating-point calculation of operations per second (FLOPS) represents that the resources are not being used to the maximum of their computational capacity. Bus congestion or contention on data access may be suggested by the use of memory bandwidth on its own, particularly when combined with low throughput. Layer latency analyses can be used to identify particular sections of the network that slow down the runtime, and warp efficiency of GPU-based systems can give an insight into the coherent execution of threads. Very low warp efficiency can usually suggest non-check thread paths or ineffective thread loads [44]. Following these diagnostic understandings, a variety of optimisation methods could be used at run time to improve the efficiency of the inferences significantly.

A popular trick is layer fusion, whereby neighbouring operations, e.g., convolution, batch normalisation, and activation, are combined into a single kernel. This not only lowers kernel launch overhead but also memory locality and less cache thrashing, thus giving rise to lower latency. Dynamic profiling data is utilised to inform automated graph optimisations, such as fusion using tools such as TensorRT and OpenVINO. Another strong trick is quantisation, where weights and activations of higher precision (FP32) are converted to lower ones (INT8 or FP16). This saves memory and speeds up inference, particularly on those devices that have purpose-built low-precision compute units. All of these tools (PyTorch Quantisation Toolkit, TensorFlow Lite, and ONNX Runtime) are based on quantisation-aware workflows and conversion pipelines after or during training. Equally, pruning removes duplicated network connections or neurons, decreasing the computational load at no cost to the accuracy. The layers with low-importance or high-latency as identified during profiling can be selectively pruned, assisted by such tools as the TensorFlow Model Optimization Toolkit and Sparse Tensor Core improvements provided by NVIDIA [45,46].

Another example of the critically important run-time optimisations would be the ability to tune the batch size in the queries, where, with profiling, developers can test which batch size will use the hardware to maximum without inundating the memory. In other systems, batching size can be dynamically altered throughout inference depending on real-time feedback and thus prevent under-utilisation or thrashing. Memory optimisation, an edge-relevant case, is the allocation pattern enhancement of tensors. This encompasses buffer recycling, a static allocation of memory, and detailing against unneeded recomputation approaches that are profile-driven directly. Frameworks may substitute

under-performing default compute kernels with an optimised version, as well. As another example, Winograd convolution kernels are used in some cases with specific filter sizes; in the case of a preferred filter size, or a custom fused kernel may be used. Such optimisations are facilitated by tools such as TVM, XLA, and Glow, performing dynamic code generation on the basis of profiling data.

Performance is also improved further by the fact that runtime scheduling coordinates the execution of operations depending on the availability of resources and inter-layer dependencies. Dynamic profiling allows exploiting techniques like asynchronous execution, where I/O is overlapped with compute, layer pipelining in transformer networks, and model parallelism across mixed hardware devices, all of which minimize wait time and achieve high throughput. Another advantage of dynamic profiling is adaptive precision and energy-aware inference that enables systems to switch between low-power and high-accuracy modes according to the environment. As an example, models running in thermally-constrained environments might trade precision or complexity to stay at power budgets, whereas in a high-resource state, the same model can employ more compute-intensive pathways at the expense of time to gain greater accuracy. Such adaptation inference has already been implemented in the AI cameras, mobile assistants [47].

To sum up, dynamic profiling is much more than a diagnosis tool: it can be used as a strategic performance enhancer in AI. It is core in minimizing latency, maximising throughput, and SvTPE optimisation of energy efficiency by exposing bottlenecks and influencing the choice of specific optimisation techniques, such as fusion, quantisation, pruning, memory allocation, and adaptive scheduling. These optimisations are essential to systematically deploy AI models to production systems, including installations on cloud servers, all the way to resource-limited edge devices.

6. Adaptive AI Deployment: Future Directions

With the accelerated adoption of AI across numerous technology areas, including autonomous transportation, healthcare, smart cities, and industrial IoT, the pressure to deploy adaptive strategies to implement AI is increasing to respond in an intelligent way to environmental and system dynamics. All-purpose AI deployment strategies are fixed and rigid and may presuppose a specific employment situation. But currently, inference workloads are typified by variable user demand, hard heterogeneity, and input variability, which requires real-time adaptive systems that can alter behavior on the fly. Dynamic profiling is central to facilitating this flexibility because it provides essential information about system performance to inform activities like model selection, precision tuning, load balancing, and offloading. Examples include allowing systems to fall back to lower-precision inference (e.g., INT8 or FP16) when thermal headroom or latency targets dictate this, where engines such as TensorRT are able to run mixed-precision work with minimal accuracy loss.

Similarly, profiling information at the run-time can justify model ensemble or multi-capacity networks, which calibrate themselves given device constraints by choosing suitable model variations on the fly, a strategy already investigated in mobile vision systems and mobile robotics. In cloud-edge systems, workload migration strategies are guided by dynamic profiling, enabling devices to delegate inference to cloud instances in the case of resource saturation or to execute it locally in the case of resource requirements where latency-aware responsiveness is critical, as exhibited by systems such as AWS Greengrass and Azure Percept. Besides, frameworks based on compilers like TVM and Glow use dynamic traces to implement optimised code using meta-compilation, having the ability to compile code dynamically that meets the requirements of present hardware states and workloads. Profiling also plays a role in continuous optimisation pipelines, with deployed models being monitored continuously and re-trained using real-time performance data, a technique essential to troubleshooting model drift and meeting service-level agreements (SLAs) delivered by production systems. Adaptive deployment in edge computing acquires a new significance as resource constraints are critical. Devices such as NVIDIA Jetson and Qualcomm AI Engine dynamically scale the complexity of the model, frequency of execution, and power states through the profiling signal of power consumption, thermal states, and CPU/GPU utilisation.

However, there are several challenges underlying adaptive AI systems, among which are the need to maintain low-latency within systems that make decisions, high-robustness to uncertainty, and the security of profiling data against both misuse and leaks. Newer developments are in reinforcement-based automatic tuning of models during execution time and federated models that exchange anonymised profiling signals between distributed nodes to increase overall efficiency. This trajectory ultimately leads to self-optimising AI systems self-optimising ais automotive that can monitor their performance, compare it with respect to baselines, adjust themselves as per the situation demands, and even recover by themselves once their performance is impaired. The systems will form the core of the next generation of smart infrastructure, supporting scalable, sustainable, and efficient deployment of AI both in-cloud and in-edge.

7. Conclusion

Artificial Intelligence has developed at a fast pace, resulting in new innovations in relation to computer vision, speech recognition, and decision-making systems in ways never seen before. Nevertheless, as AI moves to production, new issues emerge, especially those that optimize the performance of inference workloads in a variety of conditions. The success of AI systems is no longer gauged on the basis of their model accuracy, but also on their execution capacity to supply low-latency, energy-efficient, scalable inference in deployment situations under real-world circumstances. As this paper has discussed, dynamic profiling is a very important technique to overcome these performance issues and drive the best future of AI deployment.

In contrast with static profiling, which functions on predetermined examination cases and does not consider the differences in the circumstantial performance at run time, dynamic profiling provides real-time insight into model performance at run time. Dynamic profiling enables developers to identify and target areas of performance overheads in a highly granular manner, by recording operational details like kernel runtimes, memory accesses, data transfer delays, and compute utilisation. It allows getting a closer insight into where inferences are delayed, inefficient, or the hardware is not maximally utilised. Profiling tools like NVIDIA Nsight, Intel VTune, PyTorch Profiler, and TensorFlow Profiler give insights that are invaluable across both CPUs, GPUs, TPUs, and edge devices, helping in cross-platform optimisation and model refinement.

With these insights, practical optimisation techniques are translated, like operator fusion, mixed-precision inference, structurally-pruned, memory tiling, kernel replacement, and batch size tuning. All of these methods have the potential to gain much more speed and efficiency of inference, particularly with guidance based on real-world data. Of greater significance is that the addition of dynamic profiling into AI pipelines can support adaptive deployment, where AI systems dynamically adapt their execution patterns, in line with real-time profiling feedback. This involves modifying the precision of a model during thermal stress, the selection of resource-constrained models with lightweight models, or offloading, as required.

Adaptive AI is not only another step towards the new approach to design and implementation of AI systems. Instead of being based on prescriptive, one-size-fits-all deployment models, future AI applications will become self-optimising systems, i.e., systems that can themselves monitor, evaluate, and adapt their performance dynamically. Along with being favorable towards latency and throughput enhancements, this paradigm is also important to enable applications in areas of autonomous driving, remote healthcare, and smart infrastructure, where resilience to system failures and responsiveness are non-negotiable.

But in order to maximise the potential of dynamic profiling and adaptive AI, a few challenges will have to be overcome. The profiling tools should incur as little overhead as possible so that they do not affect the performance of systems. Adaptation schemes need to be resilient, not to overfit to temporary circumstances or respond to noisy hints. Security and privacy of data need to be maintained, especially in settings where profiling data can result in sensitive features of operations. Along with this, the tool and framework fragmentation across platforms is also still a hindrance to massive adoption, and thus requires standardisation and interoperability.

To sum up, dynamic profiling is a foundation stone for the development of effective and smart AI implementation. It closes the gap between design assumptions that are fixed and the dynamic environment of operation. Through precise, context-sensitive optimisation and enabling the creation of adaptive inference systems, dynamic profiling will enable delivery of scalable, responsive, and sustainable AI solutions to cloud, edge, and hybrid ecosystems. The need to embrace dynamic profiling holds the key to tapping the potential of AI as it finds its way into daily systems and infrastructure-making intelligent systems not only accurate, but performing, agile, and efficient amidst dynamic circumstances.

References

[1] Goodfellow I, Bengio Y, Courville A. Deep learning. Cambridge: MIT Press; 2016.

[2] LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015;521(7553):436-44.

[3] Sze V, Chen YH, Yang TJ, Emer JS. Efficient processing of deep neural networks: A tutorial and survey. Proc IEEE. 2017;105(12):2295-323.

[4] Zhang C, Li P, Sun G, Guan Y, Xiao B, Cong J. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In: Proc 2015 ACM/SIGDA Int Symp Field-Programmable Gate Arrays. 2015. p. 161-70.

[5] Lane ND, Bhattacharya S, Georgiev P, Forlivesi C, Jiao L, Qendro L, et al. DeepX: A software accelerator for low-power deep learning inference on mobile devices. In: Proc 15th ACM/IEEE Int Conf Information Processing in Sensor Networks (IPSN). 2016. p. 1-12.

[6] Chamoso P, Bartolomé Á, García-Retuerta D, Prieto J, De La Prieta F. Profile generation system using artificial intelligence for information recovery and analysis. J Ambient Intell Humaniz Comput. 2020;11(11):4583-92.

[7] Chen Q, Gao C, Fu Y. Cerebron: A reconfigurable architecture for spatiotemporal sparse spiking neural networks. IEEE Trans VLSI Syst. 2022;30(10):1425-37.

[8] Banner R, Nahshan Y, Soudry D. Post training 4-bit quantization of convolutional networks for rapid-deployment. Adv Neural Inf Process Syst. 2019;32.

[9] Wiedemann S, Kirchhoffer H, Matlage S, Haase P, Marban A, Marinč T, et al. DeepCABAC: A universal compression algorithm for deep neural networks. IEEE J Sel Top Signal Process. 2020;14(4):700-14.

[10] Fukuda T, Suzuki M, Kurata G, Thomas S, Cui J, Ramabhadran B. Efficient knowledge distillation from an ensemble of teachers. In: Interspeech. 2017. p. 3697-701.

[11] Li D, Wang X, Kong D. DeepRebirth: Accelerating deep neural network execution on mobile devices. In: Proc AAAI Conf Artif Intell. 2018;32(1).

[12] Zhang X, Zhou X, Lin M, Sun J. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In: Proc IEEE Conf Comput Vis Pattern Recognit. 2018. p. 6848-56.

[13] Zhou J, Liu S, Guo Q, Zhou X, Zhi T, Liu D, et al. Tunao: A high-performance and energy-efficient reconfigurable accelerator for graph processing. In: Proc IEEE/ACM Int Symp Cluster, Cloud Grid Comput (CCGRID). 2017. p. 731-4.

[14] Pang B, Nijkamp E, Wu YN. Deep learning with TensorFlow: A review. J Educ Behav Stat. 2020;45(2):227-48.

[15] Yousefzadeh-Asl-Miandoab E, Robroek T, Tozun P. Profiling and monitoring deep learning training tasks. In: Proc 3rd Workshop Mach Learn Syst. 2023. p. 18-25.

[16] Davoodi P, Gwon C, Lai G, Morris T. TensorRT inference with TensorFlow. In: GPU Technology Conference. 2019.

[17] Han Y, Huang G, Song S, Yang L, Wang H, Wang Y. Dynamic neural networks: A survey. IEEE Trans Pattern Anal Mach Intell. 2021;44(11):7436-56.

[18] Lane ND, Georgiev P. Can deep learning revolutionize mobile sensing? In: Proc 16th Int Workshop Mobile Comput Syst Appl. 2015. p. 117-22.

[19] Chen T, Moreau T, Jiang Z, Zheng L, Yan E, Shen H, et al. TVM: An automated end-to-end optimizing compiler for deep learning. In: Proc 13th USENIX Symp Operating Syst Design Implementation (OSDI). 2018. p. 578-94.

[20] Gholami A, Kim S, Dong Z, Yao Z, Mahoney MW, Keutzer K. A survey of quantization methods for efficient neural network inference. In: Low-Power Computer Vision. Chapman and Hall/CRC; 2022. p. 291-326.

[21] Carballo-Hernández W, Pelcat M, Bhattacharyya SS, Galán RC, Berry F. Flydeling: Streamlined performance models for hardware acceleration of CNNs through system identification. ACM Trans Model Perform Eval Comput Syst. 2023;8(3):1-33.

[22] Zou B, Sun B, Hu Y, Klod T, Caccamo M, Abdelzaher T. A performance prediction-based DNN partitioner for edge TPU pipelining. In: Proc IEEE Military Commun Conf (MILCOM). 2024. p. 1-6.

[23] Rauschmayr N, Kumar V, Huilgol R, Olgiati A, Bhattacharjee S, Harish N, et al. Amazon SageMaker Debugger: A system for real-time insights into machine learning model training. Proc Mach Learn Syst. 2021;3:770-82.

[24] Choi W, Choi T, Heo S. A comparative study of automated machine learning platforms for exercise anthropometry-based typology analysis: Performance evaluation of AWS SageMaker, GCP VertexAI, and MS Azure. Bioengineering. 2023;10(8):891.

[25] Alameddine HA, Sharafeddine S, Sebbah S, Ayoubi S, Assi C. Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing. IEEE J Sel Areas Commun. 2019;37(3):668-82.

[26] Golpayegani F, Chen N, Afraz N, Gyamfi E, Malekjafarian A, Schäfer D, et al. Adaptation in edge computing: A review on design principles and research challenges. ACM Trans Auton Adapt Syst. 2024;19(3):1-43.

[27] Lin J, Chen WM, Lin Y, Gan C, Han S. MCUNet: Tiny deep learning on IoT devices. Adv Neural Inf Process Syst. 2020;33:11711-22.

[28] Lee YJ, Jung HG, Suhr JK. Semantic segmentation network slimming and edge deployment for real-time forest fire or flood monitoring systems using unmanned aerial vehicles. Electronics. 2023;12(23):4795.

[29] Vu TH, Jain H, Bucher M, Cord M, Pérez P. Advent: Adversarial entropy minimization for domain adaptation in semantic segmentation. In: Proc IEEE/CVF Conf Comput Vis Pattern Recognit. 2019. p. 2517-26.

[30] Cui J, Liu S, Tian Z, Zhong Z, Jia J. ResLT: Residual learning for long-tailed recognition. IEEE Trans Pattern Anal Mach Intell. 2022;45(3):3695-706.

[31] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is all you need. Adv Neural Inf Process Syst. 2017;30.

[32] Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proc Conf North Am Chapter Assoc Comput Linguist (NAACL-HLT). 2019. p. 4171-86.

[33] Yang TJ, Howard A, Chen B, Zhang X, Go A, Sandler M, et al. NetAdapt: Platform-aware neural network adaptation for mobile applications. In: Proc Eur Conf Comput Vis (ECCV). 2018. p. 285-300.

[34] Erol R. Case studies for energy efficient machine learning inference acceleration [dissertation]. Little Rock: University of Arkansas at Little Rock; 2024.

[35] Ren S, He K, Girshick R, Sun J. Faster R-CNN: Towards real-time object detection with region proposal networks. Adv Neural Inf Process Syst. 2015;28.

[36] Kasoju A, Vishwakarma T. Optimizing Transformer Models for Low-Latency Inference: Techniques, Architectures, and Code Implementations. Int J Sci Res (IJSR). 2025;14:857-66.

[37] Deng J, Dong W, Socher R, Li LJ, Li K, Fei-Fei L. ImageNet: A large-scale hierarchical image database. In: Proc IEEE Conf Comput Vis Pattern Recognit. 2009. p. 248-55.

[38] Sandler M, Howard A, Zhu M, Zhmoginov A, Chen LC. MobileNetV2: Inverted residuals and linear bottlenecks. In: Proc IEEE Conf Comput Vis Pattern Recognit. 2018. p. 4510-20.

[39] Tan M, Le Q. EfficientNet: Rethinking model scaling for convolutional neural networks. In: Proc Int Conf Mach Learn. 2019. p. 6105-14.

[40] Sudharsan B, Salerno S, Nguyen DD, Yahya M, Wahid A, Yadav P, et al. TinyML benchmark: Executing fully connected neural networks on commodity microcontrollers. In: Proc IEEE 7th World Forum Internet Things (WF-IoT). 2021. p. 883-4.

[41] Abhishek AVS, Kotni S. Detectron2 object detection and manipulating images using cartoonization. Int J Eng Res Technol (IJERT). 2021;10(1-5):2.

[42] Rastegari M, Ordonez V, Redmon J, Farhadi A. XNOR-Net: ImageNet classification using binary convolutional neural networks. In: Proc Eur Conf Comput Vis. 2016. p. 525-42.

[43] Wan A, Dai X, Zhang P, He Z, Tian Y, Xie S, et al. FBNetV2: Differentiable neural architecture search for spatial and channel dimensions. In: Proc IEEE/CVF Conf Comput Vis Pattern Recognit. 2020. p. 12965-74.

[44] Nimmagadda Y. Model optimization techniques for edge devices. In: Model Optimization Methods for Efficient and Edge AI: Federated Learning Architectures, Frameworks and Applications. 2025. p. 57-85.

[45] Berndl M, Vitale B, Zaleski M, Brown AD. Context threading: A flexible and efficient dispatch technique for virtual machine interpreters. In: Proc Int Symp Code Generation Optimization. 2005. p. 15-26.

[46] Abdelkhalik H, Arafa Y, Santhi N, Badawy AHA. Demystifying the NVIDIA Ampere architecture through microbenchmarking and instruction-level analysis. In: Proc IEEE High Perform Extreme Comput Conf (HPEC). 2022. p. 1-8.

[47] Joshi P, Hasanuzzaman M, Thapa C, Afli H, Scully T. Enabling all in-edge deep learning: A literature review. IEEE Access. 2023;11:3431-60.