

# Focus Session: Hardware and Software Techniques for Accelerating Multimodal Foundation Models

Muhammad Shafique, Abdul Basit, Muhammad Abdullah Hanif, Alberto Marchisio,

Rachmad Vidya Wicaksana Putra, Minghao Shao

*eBRAIN Lab, New York University (NYU) Abu Dhabi, Abu Dhabi, United Arab Emirates*

{muhammad.shafique, ab7441, mh6117, alberto.marchisio, rachmad.putra, shao.minghao}@nyu.edu

**Abstract**—This work presents a multi-layered methodology for efficiently accelerating multimodal foundation models (MFMs). It combines hardware and software co-design of transformer blocks with an optimization pipeline that reduces computational and memory requirements. During model development, it employs performance enhancements through fine-tuning for domain-specific adaptation. Our methodology further incorporates hardware and software techniques for optimizing MFMs. Specifically, it employs MFM compression using hierarchy-aware mixed-precision quantization and structural pruning for transformer blocks and MLP channels. It also optimizes operations through speculative decoding, model cascading that routes queries through a small-to-large cascade and uses lightweight self-tests to determine when to escalate to larger models, as well as co-optimization of sequence length, visual resolution & stride, and graph-level operator fusion. To efficiently execute the model, the processing dataflow is optimized based on the underlying hardware architecture together with memory-efficient attention to meet on-chip bandwidth and latency budgets. To support this, a specialized hardware accelerator for the transformer workloads is employed, which can be developed through expert design or an LLM-aided design approach. We demonstrate the effectiveness of the proposed methodology on medical-MFMs and on code generation tasks, and conclude with extensions toward energy-efficient spiking-MFMs.

**Index Terms**—Multimodal Foundation Models (MFMs), Generative AI, Large Language Models (LLMs), Vision-Language Models (VLMs), Vision-Language-Actions (VLAs), Hardware and Software Techniques, Optimizations.

## I. INTRODUCTION

The rapid advancements of artificial intelligence (AI) have led to the emergence of *foundation models*, i.e., models that are trained with massive datasets to learn diverse input representations and can be adapted for domain-specific (downstream) tasks with minimal additional training [1]. The growing interest in foundation models is reflected through the active developments of *Large Language Models (LLMs)*, *Vision Transformer Models (ViTs)*, and *Latent Diffusion Models (LDMs)* [1] [2]. Their generality and flexibility have inspired a new approach on how intelligent systems are built and deployed [2]. Recent works explore how these models can be enhanced for processing multimodal data and solving diverse cross-modality tasks (e.g., text-based image retrieval and generation). These models are known as *Multimodal Foundation Models (MFMs)* [1].

As multimodal models fuse multiple modalities (e.g., text, image, and audio), they enable strong performance in diverse application domains (e.g., image and video generation [3] [4], segmentation [5], visual-question answering [6], healthcare [7], and embodied robotics [8]). However, their scale incurs huge computational, memory/storage, bandwidth, and power/energy costs that hinder efficient training and deployment [1], especially for resource-constrained platforms (e.g., edge devices). Therefore, *optimization techniques in hardware and software domain are essential for accelerating MFMs*.

Key Challenges in Accelerating Multimodal Foundation Models (MFMs)

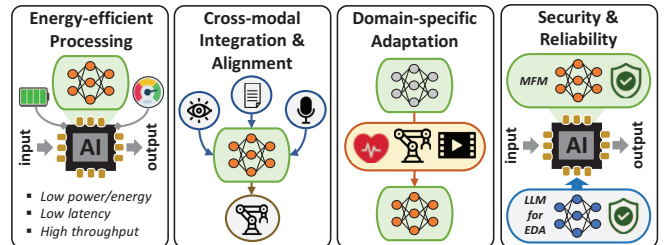


Fig. 1. Overview of challenges in accelerating multimodal foundation models.

### A. Key Challenges in Accelerating MFMs

We highlight the key challenges in accelerating MFMs in Fig. 1 and discuss them in the following.

- **Energy-efficient processing of MFMs:** The size of models leads to substantial computational and memory costs, thus posing energy efficiency challenges in training and inference, especially when they are deployed on resource-constrained platforms. Therefore, model acceleration process should significantly improve its energy efficiency.
- **Cross-modal integration and alignment:** Accelerating the foundation model with heterogeneous modalities (e.g., text, image, and audio) requires an effective multimodal alignment which preserves semantic consistency, while controlling computational and memory overheads.
- **Domain-specific adaptation:** Deployment of the foundation model for a specific domain requires domain-aware fine-tuning and optimization techniques to accelerate the processing and satisfy the constraints of the targeted application (e.g., accuracy, latency, and interpretability).
- **Security and reliability:** Safety-critical application use cases demand strong robustness to adversarial inputs and distribution shifts, along with safeguards for intellectual property (IP), privacy, and potential data contamination throughout the model lifecycle. Therefore, the security and reliability aspects should also be ensured when performing model acceleration.

To address each challenge individually, different techniques have been proposed in the literature. However, *a methodology for systematically integrating effective hardware and software techniques is still required to enable high-performance and energy-efficient MFM-based systems*.

### B. Our Contributions

In the light of the above discussion, **the contributions of this paper** are highlighted in the following.

- *We present an overview of different key challenges in accelerating MFMs*, including energy efficiency, cross-modal integration & alignment, domain-specific adaptation, as well as security & reliability (**Section I-A**).

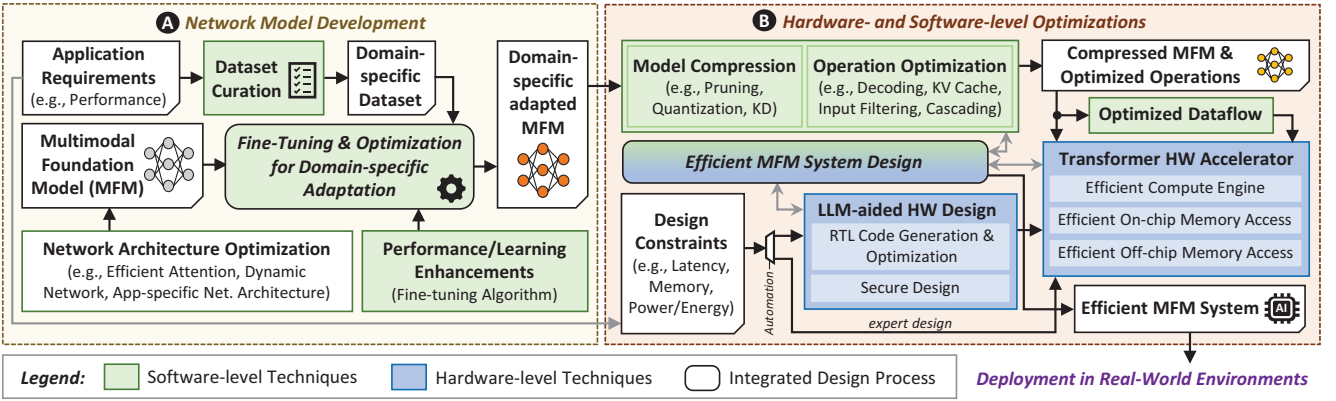


Fig. 2. Overview of our multi-layered methodology for accelerating multimodal foundation models (MFMs).

- We present a multi-layered design methodology for accelerating MFMs, describing the systematic design pipeline that includes model development for domain-specific adaptation as well as hardware- and software-level optimization techniques (Section II-A).
- We present hardware and software techniques for accelerating MFMs, including model compression, application-driven optimization, transformer hardware accelerator design, and LLM-aided electronic design automation (EDA) that can shorten the hardware design cycle (Section II-B and II-C).
- We extend the energy efficiency of MFMs through spiking neural networks (SNNs), developing the suitable optimization techniques for their event-based operations (Section IV).

## II. OUR METHODOLOGY FOR ACCELERATING MFMS

### A. Overview

To enable the acceleration of MFMs while satisfying the application requirements and constraints, we propose a *design methodology that systematically integrates effective hardware- and software-level techniques in a seamless design pipeline*; as shown in Fig. 2 and discussed below.

**A Network Model Development:** An MFM typically requires domain-specific adaptation for targeting the downstream tasks. Hence, it is important to perform *fine-tuning and optimization for domain-specific adaptation*. To support this, the following development aspects are essential.

- *Dataset curation:* A domain-specific dataset has to be carefully curated and then employed for fine-tuning the model considering the application requirements (e.g., performance). Here, dataset distillation [9] can be employed to reduce the size of training data while preserving the information.
- *Performance/learning enhancements:* To improve the performance of the model, pre-training and fine-tuning algorithms can be employed; see ② in Fig. 3. Pre-training algorithm is used for model development in pre-training phase; while fine-tuning algorithm aims at enhancing performance of the pre-trained model, such as additive tuning [10] [11], selective tuning [12] [13], and re-parameterization tuning [14] [15].

In the case when model development from scratch is needed, we can benefit from optimization techniques for network architecture design, such as efficient attention (e.g., sparse [16], approximate [17], and attention-free [18] [19]), dynamic network (e.g., Mixture-of-Expert (MoE) [20] and early-exiting [21]), and application-specific architecture [22]–[24]; see ① in Fig. 3.

**B Hardware- and Software-level Optimizations:** Once the domain-adapted MFM is obtained, further optimizations in hardware and software domains can be performed, as follows.

- *Model compression:* The domain-adapted MFM can be compressed through software techniques, such as pruning (i.e., unstructured [25]–[27] and structured [28]–[30]), quantization [31]–[34], knowledge distillation (KD) [35], and low-rank decomposition (LoRD) [36] [37]; see ③ in Fig. 3. Pruning eliminates insignificant weights; quantization lowers the bit-precision of weights and/or activations. KD transfers knowledge from a bigger model (teacher) to a smaller model (student). Meanwhile, LoRD decomposes the weight matrix in the models into multiple smaller matrices. Furthermore, there is significant potential in combining multiple compression techniques to achieve higher compression ratios with minimal accuracy loss. For instance, Optimal Brain Restoration (OBR) framework [38] employs a training-free approach that jointly leverages pruning and quantization.
- *Operation optimization:* Operations of the domain-adapted MFM can be optimized using software techniques, e.g., Key-Value (KV) cache [39], efficient decoding (e.g., speculative decoding) [40], input filtering (i.e., prompt compression [41] and token pruning [42]), model cascading [43], and long-context optimization [44]; see ④ in Fig. 3. KV cache avoids re-computation of KV by reusing the existing results. Speculative decoding avoids costly sequential decoding by using a small model to guess next tokens and a larger model to verify them in a single forward pass. Input filtering reduces the input size using prompt compression and token pruning. Model cascading routes queries through a small-to-large cascade and uses lightweight self-tests to determine when to escalate to larger models. Meanwhile, long-context optimization handles long input via recurrent structure and attention enhancements.
- *Optimized dataflow:* To efficiently execute the compressed MFM and its operations, dataflow should be optimized based on the underlying hardware architecture [45]. This execution is scheduled for maximizing data reuse on-chip, since on-chip operations incur significantly lower energy consumption than off-chip memory access [46]; see ⑤ in Fig. 3.
- *Hardware accelerator:* To maximize the efficiency gains in accelerating MFMs, specialized hardware accelerators are employed [47]; see ⑥ in Fig. 3. These accelerators aim to satisfy computational and memory requirements of targeted models. To achieve this, the compute engine can be designed

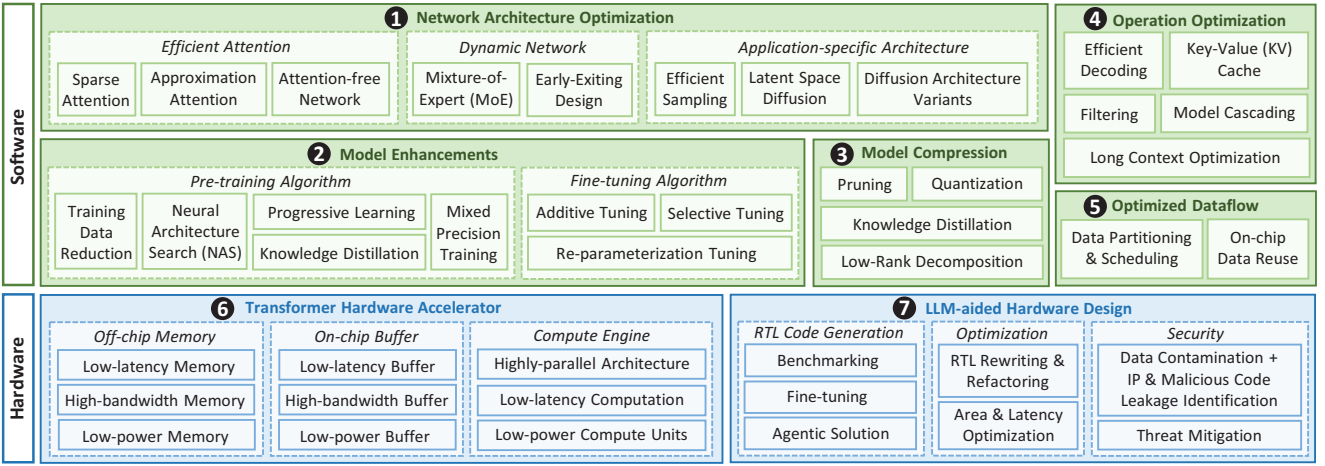


Fig. 3. Overview of hardware and software techniques for accelerating MFMs.

with highly-parallel architecture, low-latency computation, and low-power compute units. Meanwhile, the off-chip and on-chip memories can be designed with low-latency, high-bandwidth, and low-power memory technologies.

- **LLM-aided hardware design:** To enable a fast and scalable hardware accelerator design process, LLMs can be employed to generate register-transfer level (RTL) codes of the accelerator from natural language specifications (including design constraints) in a close-loop design flow (i.e., specifications  $\rightarrow$  RTL generation  $\rightarrow$  verification  $\rightarrow$  synthesis  $\rightarrow$  feedback); see 7 in Fig. 3. Furthermore, this approach also allows a seamless integration with the existing EDA tools.

Once the MFM and hardware accelerator are developed and optimized, they are then integrated as a system for deployment in the targeted real-world application.

### B. Software-level Optimization Methods

1) **Model Compression:** MFMs can achieve impressive performance at the cost of huge computational and memory requirements, hence huge power/energy consumption. To address this, model compression is essential. In the pruning category, SparseGPT [25] and Wanda [26] target unstructured layer-wise sparsity, Plug-and-Play [27] leverages relative importance of weights and activations, LLM-Pruner [28] employs structured group-wise pruning based on gradient information, while SLEB [29] considers transformer block-level pruning. In the quantization category, SmoothQuant [31] migrates quantization difficulties from activations to weights, AWQ [32] quantizes most of the weights while keeping the salient ones in higher precision, OmniQuant [33] uses block-wise quantization error minimization, while SpinQuant [34] uses a rotation matrix to reduce outliers and enhance quantizability. These techniques usually rely on uniform quantization across transformer blocks, thus leading to suboptimal memory savings. Furthermore, our experimental results demonstrate that different blocks may have different sensitivity under quantization, as shown in Fig. 4(a). Based on this observation, we perform exploration to obtain pareto-optimal quantized models that achieve competitive performance with significant memory savings from the baseline through a mixed-precision approach; see Fig. 4(b). For instance, a selected model at label-① has perplexity score of 4.73, which is slightly above the baseline model with perplexity score of 4.46, while significantly saving 40.74% of memory footprint.

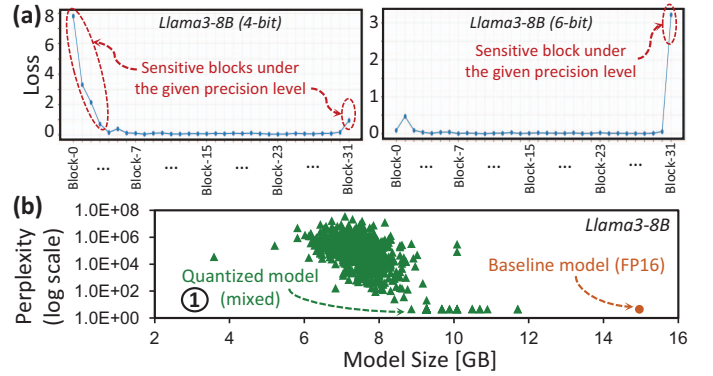


Fig. 4. Our experimental results on the WikiText-2 dataset for (a) the impact of different precision levels in each transformer block of Llama3-8B; and (b) the performance of Llama3-8B under weight quantization considering different combinations of precision levels across transformer blocks.

### C. Hardware-level Optimization Methods

1) **Transformer Accelerator:** When executing Transformers on general-purpose processors or GPUs, inference often suffers from excessive latency and power consumption, limiting practical use at the edge. Existing ML accelerators primarily target convolutional workloads and dense linear algebra, offering limited native support for Transformer-specific primitives. Operations such as multi-head attention, normalization layers, and nonlinear activations are frequently emulated through sequences of basic kernels, leading to inefficiencies and unnecessary data movement. Several prior designs focus on accelerating isolated components, such as matrix multiplications or attention blocks, without addressing the full inference pipeline [48]. Moreover, the reliance on floating-point arithmetic significantly increases silicon area, energy usage, and critical path delay. To overcome these limitations, an integer-only execution model that uniformly supports all Transformer operations is essential for achieving both high efficiency and low power consumption.

**Quantization Scheme for Accelerators:** As shown in Fig. 5, the accelerator adopts a unified quantization strategy that enables integer-only computation across the entire Transformer pipeline. Model parameters and intermediate activations are represented using 8-bit signed integers, providing a compact and energy-efficient storage format. Intermediate accumulations and nonlinear transformations are carried out in 32-bit integer

precision to preserve numerical stability. After each operation, a requantization step rescales the results back to 8-bit precision, ensuring that values remain within a valid dynamic range. This quantization-requantization flow is consistently applied to all major components, including attention, feed-forward layers, normalization, and activation functions, enabling a homogeneous and hardware-friendly design.

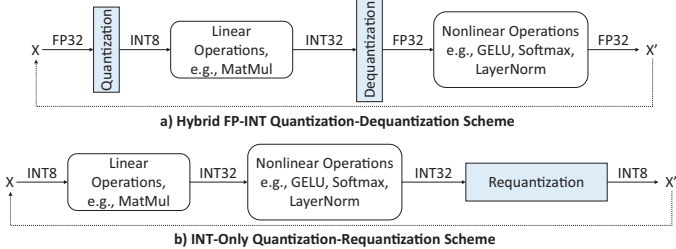


Fig. 5. Computation flows for (a) the Quantization-Dequantization scheme of [49], and (b) the quantization-based scheme with *Requantization* blocks [50].

**Hardware Accelerator Architecture:** As shown in Fig. 6, the SwiftTron architecture [50] is organized around a set of tightly coupled functional units optimized for Transformer workloads. The matrix multiplication engine serves as the primary compute backbone, handling both projection layers and feed-forward networks with high throughput. A dedicated requantization unit follows each compute stage to normalize intermediate results and manage precision transitions efficiently. The multi-head self-attention module orchestrates query, key, and value projections, while the attention engine performs scaled dot-product computations. Specialized units implement Softmax and GELU using integer arithmetic, and a LayerNorm block ensures stable activation distributions. On-chip memory buffers minimize external memory access, and a centralized control unit schedules dataflow and synchronizes all components.

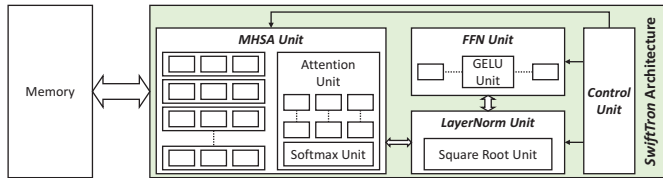


Fig. 6. Top-level overview of the *SwiftTron* architecture.

An end-to-end integer-only datapath substantially reduces hardware complexity and energy consumption when compared to floating-point alternatives. As shown in Table I, experimental results demonstrate significant performance gains, achieving over  $3.5\times$  speedups over GPU baselines across language and vision Transformer benchmarks. The hardware analysis shows that matrix multiplication dominates both area and power consumption (i.e., 55% and 79%, respectively), highlighting it as the primary optimization target. Overall, the results indicate that a fully integrated, quantized Transformer accelerator can deliver high performance while maintaining a compact and energy-efficient footprint.

TABLE I  
ACCURACY AND INFERENCE LATENCY COMPARISON BETWEEN SWIFTRON AND RTX 2080 Ti GPU.

Model	Accuracy	Latency	Speedup w.r.t. GPU
RoBERTa-base on STT-2	95.2%	1.83 ms	3.81 $\times$
RoBERTa-large on STT-2	96.4%	45.70 ms	3.90 $\times$
DeiT-S on ImageNet	79.11%	1.13 ms	3.58 $\times$

2) **LLM-aided Hardware Design:** We discuss the challenges of current accelerator design, RTL code generation and optimization, as well as security concern in LLM-aided EDA.

**Challenges of Current Accelerator Design:** The design of hardware accelerators for transformer-based models poses major challenges that slow development and deployment. Conventional workflows require substantial expertise in digital circuit design and hardware description languages (HDLs), creating high barriers to entry and long design cycles. Teams often spend months iterating through specification, RTL implementation, verification, and synthesis, as constraints discovered at later stages force revisions to earlier choices. Manual design also limits effective exploration of the design space for energy-efficient architectures. Engineers rely on prior experience and heuristics, yet the complex interactions among compute, data movement, and on-chip storage make the evaluation of many alternatives impractical. These issues are compounded by the rapid evolution of foundation models, where workloads vary widely from dense attention to sparse mixture-of-experts [51]. Therefore, automated hardware generation is increasingly important for translating high-level specifications and constraints into optimized RTL designs with reduced dependence on scarce hardware expertise [52].

**RTL Code Generation and Optimization:** LLMs have demonstrated strong ability in generating codes [53] from natural language specifications, providing potential to transform hardware design workflows. This capability spans in multiple levels, including datasets and benchmarks, model-level methods, and agentic architectures, which together enable a closed-loop flow integrating specification, generation, verification, synthesis, and feedback. High-quality datasets and benchmarks form the foundation of LLM-aided RTL generation. Efforts include VerilogDB [54] for large-scale training corpora with synthesis validation, while benchmarks such as VerilogEval [55] and RTLLM [56] assess functional correctness. At the model level, domain-adapted LLMs achieve significant gains over general-purpose models through techniques such as domain-adaptive pretraining (e.g., ChipNeMo [57]) and efficient fine-tuning (e.g., RTLCoder [58]). Closed-loop refinement frameworks such as VeriAssist [59] leverage compiler and simulator feedback for iterative code repair. At the agentic level, multi-agent systems such as MAGE [60] coordinate specialized agents for RTL generation, testbench creation, and debugging, reflecting human design workflows. For EDA tool integration, frameworks such as ChatEDA [61] demonstrate strong task planning, while VeriDispatcher [62] explores multi-LLM dispatching for RTL generation. For both security and optimization, NetDeTox [63] leverages LLMs to minimize chip area overhead while maintaining robustness against GNN-based attacks.

**Security Concerns in LLM-aided EDA:** As LLMs become an integrated part of hardware design workflows, security has emerged as a dual-use research area. One direction applies LLMs to hardware security tasks, while the other examines the risks introduced by LLM-assisted design and verification [64]. On the offensive side, GHOST shows that an LLM can automate hardware Trojan construction and insertion at the RTL level. TrojanLoC [65] further indicates that general-purpose LLMs can help attackers navigate large SoC codebases and identify regions for Trojan insertion or secret leakage. On

the defensive side, BugWhisperer improves RTL-level vulnerability detection [66]. LLM-aided EDA also introduces assurance challenges. VeriContaminated highlights that benchmark contamination can inflate performance claims and complicate evaluation, and proposes mitigation through token generation probability statistics such as Min-K% probability combined with similarity checks [67]. Fig. 7 shows different models’ contamination rates with two detection approaches against VerilogEval and RTLLM. Verileaky shows that models fine-tuned on proprietary RTL may leak IP, and proposes a logic-locking-based approach for mitigation [68], as shown in Fig. 8. SALAD further addresses related data risks through machine unlearning to forget contaminated or sensitive RTL patterns [69].

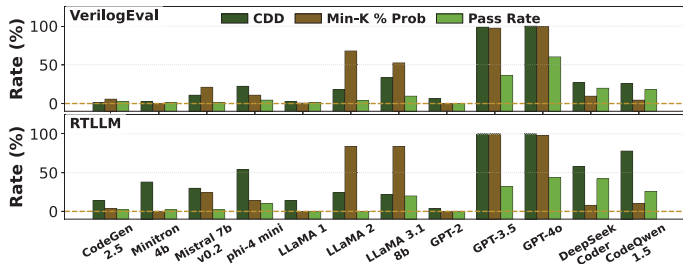


Fig. 7. Data contamination rate from pre-training data and pass rate for different LLMs on VerilogEval and RTLLM.

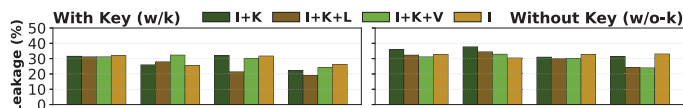


Fig. 8. Logic locking (I + K + L and I + K + V) successfully mitigates the potential IP leakage from LLMs.

### III. CASE STUDY IN HEALTHCARE AND ROBOTICS

To demonstrate the effectiveness of compression techniques for multimodal large language model (MLLM)-based real-world systems, we present two representative case-studies for different safety-critical domains: (i) medical MLLMs for radiology assistance in low-resource healthcare settings, and (ii) vision–language–action (VLA) pipelines for embodied robotic navigation, which typically demand high energy efficiency due to limited compute, memory, and power budgets. In both cases, domain-specific adaptation of an MFM is followed by model compression (e.g., quantization and pruning), and hardware-aware deployment on embedded GPUs.

1) **Healthcare VLM:** We evaluate an optimization and deployment framework for medical MLLMs in which a general-purpose TinyLLaVA-1.5B backbone [70] is adapted to the biomedical domain and then aggressively quantized. The optimization stage comprises three fine-tuning phases: (1) biomedical alignment on 600k PMC-15M image–text pairs [71], updating only the projection layer; (2) instruction tuning on 60k PMC-15M instruction-style samples to obtain a conversational assistant; and (3) downstream fine-tuning on VQA-RAD, SLAKE, and PathVQA for radiology and pathology VQA. The resulting TinyLLaVA-Med-F model attains competitive accuracy to larger 7B medical MLLMs with only 1.5B parameters.

In the compression stage, post-training quantization (PTQ) is applied to both TinyLLaVA-Med-F [7] and the 7B LLaVA-Med model [72], producing 8-bit and 4-bit variants. Quantization is performed on weights and activations using a calibration set

from the medical VQA datasets, without additional gradient updates. This yields a family of low-bit medical MLLMs across model sizes and bit-widths, allowing us to jointly analyze task accuracy, conversational quality, and memory footprint; the key results are summarized in Fig. 9.

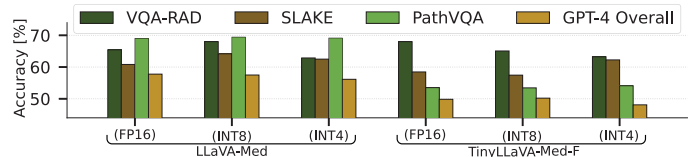


Fig. 9. Compression results on medical VQA (closed-question accuracy).

VQA accuracy is largely robust to low-bit quantization since both 7B and 1.5B models exhibit only minor variation across datasets, with 8-bit variants occasionally matching or slightly exceeding full-precision performance, consistent with calibration-induced regularization. GPT-4 evaluation shows a similar trend, indicating that conversational quality remains stable under quantization. From a deployment perspective, moving to 4-bit precision reduces dynamic memory by  $\sim 3\times$  for the 7B model and  $\sim 9\times$  for the 1.5B model, hence enabling execution on 6GB consumer GPUs and Jetson-class embedded platforms. Together, these results demonstrate that the quantization-centric pipeline achieves substantial memory savings while preserving clinically meaningful performance.

2) **Robotics VLM:** In robotic perception, we consider open-vocabulary object detection (OVD) as a key *vision–language* capability that underpins embodied navigation.

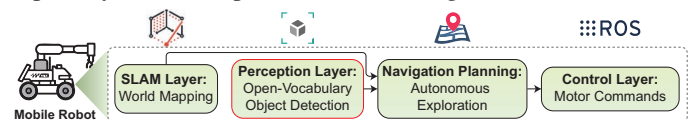


Fig. 10. Overview of robotics system (SLAM, OVD, embodied navigation).

As illustrated in Fig. 10, OVD operates at the perception layer, providing semantically grounded object representations that complement geometric state estimation from SLAM and are primarily consumed by navigation and task-planning modules. OVD outputs (e.g., bounding boxes and language-conditioned labels) inform semantic reasoning, obstacle awareness, and goal specification during planning. Although OVD models do not directly output actions or motor commands, their latency and throughput critically influence the end-to-end responsiveness of embodied robotic systems by enabling faster perception-driven replanning and decision-making [73].

OVD models used in robotics span across diverse architectural families, ranging from convolutional detectors augmented with language embeddings for real-time inference (e.g., YOLO-World) to transformer-based vision–language models that rely on global self-attention for stronger semantic generalization (e.g., OWLv2 and GroundingDINO). YOLO-World offer low latency due to convolutional inductive biases and dense prediction heads, but typically trade-off fine-grained semantic alignment and long-tail generalization. In contrast, transformer-based OVD models achieve superior open-vocabulary accuracy and robustness across diverse categories at substantially higher computational and memory cost. This trade-off motivates the need for systematic compression strategies that can bring these VLMs closer to real-time operation on robotic platforms.

Starting from this broader OVD landscape, we focus on OWLv2 as a representative transformer-based VLM for robotics

perception and investigate a training-free compression pipeline following the multi-layered methodology in Section II. The pipeline combines software-level mixed-precision inference with network-level simplification through encoder layer removal and token sparsification. Post-training INT8 quantization is also evaluated but excluded due to its adverse impact on vision–language alignment. Fig. 11 highlights several key insights. Mixed-precision inference alone substantially improves throughput while preserving detection accuracy, confirming that OWLv2 representations are robust to reduced numerical precision. Modest structural simplification (i.e., either via layer removal or token drop) yields additional latency reductions with limited accuracy impact, while more aggressive token sparsification exposes a clear accuracy–efficiency trade-off. Notably, combining layer removal with moderate token drop recovers most of the baseline accuracy while achieving improved latency profile, significantly narrowing the efficiency gap.

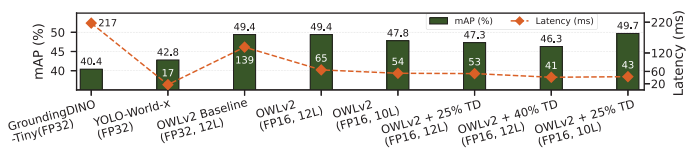


Fig. 11. OVD performance on COCO val2017 for robotics-relevant perception (TD denotes token drop).

Beyond COCO, the same compressed configuration generalizes well to LVIS and 13 robotics-focused ODinW datasets. While long-tail categories in LVIS are more sensitive to depth reduction, overall performance remains high. Across diverse aerial, driving, manipulation, and environmental monitoring scenarios in ODinW, compressed OWLv2 variants consistently achieve multi-fold speedups with comparable or improved F1 scores, indicating that structural pruning and FP16 inference act as effective regularizers and preserve domain generalization for robotics-oriented vision–language perception.

#### IV. EXTENSIONS TOWARD SPIKING-MFMS

SNNs have emerged as promising low-power/energy AI algorithms due to their sparse event-based operations [74]. Therefore, researchers recently leveraged SNNs for MFMs (so-called *Spiking-MFMs*) by applying event-based operations in transformer blocks. Here, we discuss the state-of-the-art works, their limitations, and representative optimization techniques.

1) *Spiking Language Models (SLMs)*: Several SLMs have been proposed in the literature (e.g., SpikeBERT, SpikingBERT, SNN-BERT, SpikeLM, SpikeLLM, and SpikeGPT), and they aim at achieving high performance [75]. Hence, the optimization methods for SLMs have not been extensively explored. To this end, a quantization method based on block-wise sensitivity analysis, called QSLM, is proposed in [75]. Specifically, the analysis in QSLM exposes that the attention blocks are typically less sensitive than the input and output blocks when their weights are quantized; see Fig. 12(a). Hence, a hierarchy-aware mixed-precision approach is employed to appropriately compress each block based on its sensitivity level. Experimental results show that, this technique leads to competitive performance with significant memory savings from the baseline model (i.e., SpikeGPT-216M [76]); see Fig. 12(b). For the SST-2 dataset, the quantized model achieves 84.4% accuracy, comparable to 85.7% accuracy from the baseline model, while

saving 85% of memory footprint; see ②. Meanwhile, for the WikiText-2 dataset, the quantized model achieves perplexity score of 24.6, comparable to 26.5 from the baseline model, while saving 68.7% of memory footprint; see ③.

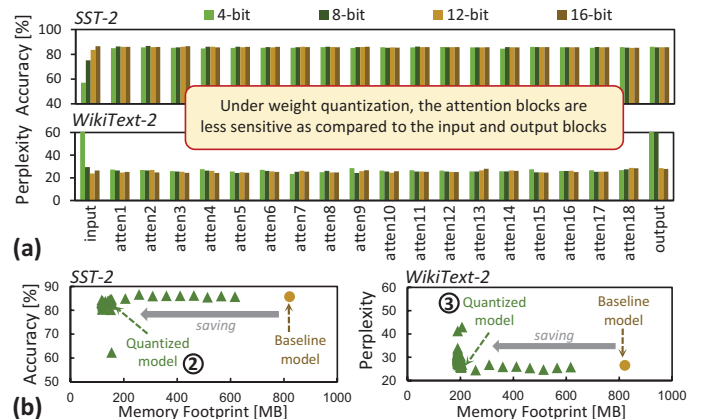


Fig. 12. Experimental results on (a) the impact of different precision levels in each attention block of SpikeGPT-216M, and (b) the performance of SpikeGPT-216M under weight quantization considering different combinations of precision levels across attention blocks [75].

2) *Spiking Vision Transformers (SViTs)*: Several SViTs have been proposed in the literature, such as Spikformer, Spike-Driven Transformer (SDT), and Spike-Driven Transformer v2 (SDTv2) [77]. These models still focus on achieving high performance (e.g., accuracy), thus their optimization methods have not been comprehensively studied. Toward this, a quantization method for SViTs is proposed in [77], called QSViT, leveraging layer-wise sensitivity analysis. Specifically, QSViT investigates the impact of block-wise quantization in the state-of-the-art SViTs on accuracy, identifies the highest and lowest precision levels across blocks that lead to acceptable accuracy, leverages this setting to explore the quantized model candidates, then selects the quantized model that achieves the highest acceptable accuracy. QSViT effectively saves 22.75% of memory footprint and reduces 21% of power consumption, while maintaining high accuracy as compared to the baseline (i.e., SDTv2 [78]).

#### V. CONCLUSION

The use of MFMs is growing rapidly, thereby ensuring their high energy efficiency, effective functionality, and security, is important. To address such design challenges, we propose a multi-layered methodology for accelerating MFMs through a design and optimization pipeline, that systematically integrates effective hardware and software techniques. The case studies highlight the effectiveness of our methodology in addressing design challenges in MFM optimization and deployment for high-performance and energy-efficient MFM-based systems. In summary, our work contributes to the advancements of novel hardware and software techniques for accelerating MFMs in an integrated methodology.

#### ACKNOWLEDGMENT

This work was partially supported by the NYUAD Center for CyberSecurity (CCS), funded by Tamkeen under the NYUAD Research Institute Award G1104, as well as partially supported by the NYUAD Center for Artificial Intelligence and Robotics (CAIR), funded by Tamkeen under the NYUAD Research Institute Award CG010.

## REFERENCES

- [1] M. Xu *et al.*, “Resource-efficient algorithms and systems of foundation models: A survey,” *ACM CSUR*, vol. 57, no. 5, Jan. 2025.
- [2] —, “A survey of resource-efficient llm and multimodal foundation models,” *arXiv preprint arXiv:2401.08092*, 2024.
- [3] A. Ramesh *et al.*, “Zero-shot text-to-image generation,” in *ICML*, 2021.
- [4] L. Khachatryan *et al.*, “Text2video-zero: Text-to-image diffusion models are zero-shot video generators,” in *ICCV*, 2023, pp. 15 954–15 964.
- [5] N. Carion *et al.*, “Sam 3: Segment anything with concepts,” *arXiv preprint arXiv:2511.16719*, 2025.
- [6] H. Liu *et al.*, “Visual instruction tuning,” *NeurIPS*, vol. 36, 2023.
- [7] A. Mir *et al.*, “Advancing healthcare in low-resource environments through an optimization and deployment framework for medical multimodal large language models,” 11 2024, pp. 1–8.
- [8] K. Kawaharazuka *et al.*, “Vision-language-action models for robotics: A review towards real-world applications,” *IEEE Access*, vol. 13, 2025.
- [9] T. Wang *et al.*, “Dataset distillation,” *arXiv preprint:1811.10959*, 2018.
- [10] G. Chen *et al.*, “MPrompt: Exploring multi-level prompt tuning for machine reading comprehension,” in *EMNLP*, 2023.
- [11] J. Li *et al.*, “Prefix propagation: Parameter-efficient tuning for long sequences,” in *ACL (Volume 2: Short Papers)*, 2023, pp. 1408–1419.
- [12] Z. Fu *et al.*, “On the effectiveness of parameter-efficient fine-tuning,” in *AAAI*, vol. 37, no. 11, 2023, pp. 12 799–12 807.
- [13] K. Huang *et al.*, “Towards green AI in fine-tuning large language models via adaptive backpropagation,” in *ICLR*, 2024.
- [14] Y. He *et al.*, “EfficientDM: Efficient quantization-aware fine-tuning of low-bit diffusion models,” in *ICLR*, 2024.
- [15] E. J. Hu *et al.*, “LoRA: Low-rank adaptation of large language models,” in *ICLR*, 2022.
- [16] K. Li, R. Yang, and X. Hu, “An efficient encoder-decoder architecture with top-down attention for speech separation,” in *ICLR*, 2023.
- [17] X. Ma *et al.*, “Mega: Moving average equipped gated attention,” in *ICLR*, 2023.
- [18] B. Peng *et al.*, “RWKV: Reinventing RNNs for the transformer era,” in *EMNLP*, 2023.
- [19] A. Gu and T. Dao, “Mamba: Linear-time sequence modeling with selective state spaces,” in *COLM*, 2024.
- [20] C. Riquelme *et al.*, “Scaling vision with sparse mixture of experts,” *NeurIPS*, vol. 34, pp. 8583–8595, 2021.
- [21] S. Bae *et al.*, “Fast and robust early-exiting framework for autoregressive language models with synchronized parallel decoding,” in *EMNLP*, 2023.
- [22] J. Song *et al.*, “Denosing diffusion implicit models,” in *ICLR*, 2021.
- [23] R. Rombach *et al.*, “High-resolution image synthesis with latent diffusion models,” in *CVPR*, 2022, pp. 10 684–10 695.
- [24] Y. He *et al.*, “Scalecrafter: Tuning-free higher-resolution visual generation with diffusion models,” in *ICLR*, 2023.
- [25] E. Frantar and D. Alistarh, “Sparsegpt: massive language models can be accurately pruned in one-shot,” in *ICML*, 2023.
- [26] M. Sun *et al.*, “A simple and effective pruning approach for large language models,” in *ICLR*, 2024.
- [27] Y. Zhang *et al.*, “Plug-and-play: An efficient post-training pruning method for large language models,” in *ICLR*, 2024.
- [28] X. Ma, G. Fang, and X. Wang, “Llm-pruner: On the structural pruning of large language models,” in *NeurIPS*, vol. 36, 2023, pp. 21 702–21 720.
- [29] J. Song *et al.*, “Sleb: streamlining llms through redundancy verification and elimination of transformer blocks,” in *ICML*, 2024.
- [30] M. Xia *et al.*, “Sheared LLaMA: Accelerating language model pre-training via structured pruning,” in *ICLR*, 2024.
- [31] G. Xiao *et al.*, “Smoothquant: Accurate and efficient post-training quantization for large language models,” in *ICML*, 2023.
- [32] J. Lin *et al.*, “Awq: Activation-aware weight quantization for on-device llm compression and acceleration,” in *MLSys*, vol. 6, 2024, pp. 87–100.
- [33] W. Shao *et al.*, “Omniquant: Omnidirectionally calibrated quantization for large language models,” in *ICLR*, 2024.
- [34] Z. Liu *et al.*, “Spinquant: Llm quantization with learned rotations,” in *ICLR*, 2025.
- [35] Y. Gu *et al.*, “MiniLLM: Knowledge distillation of large language models,” in *ICLR*, 2024.
- [36] Y. Li *et al.*, “Lospars: Structured compression of large language models based on low-rank and sparse approximation,” in *ICML*, 2023.
- [37] A. Kaushal, T. Vaidhya, and I. Rish, “LoRD: Low-rank decomposition of monolingual code LLMs for one-shot compression,” in *ICML 2024 Workshop on Foundation Models in the Wild*, 2024.
- [38] H. Guo, Y. Li, and L. Benini, “Optimal brain restoration for joint quantization and sparsification of llms,” *arXiv preprint:2509.11177*, 2025.
- [39] Z. Liu *et al.*, “Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time,” in *NeurIPS*, 2023.
- [40] Z. Sun *et al.*, “Spectr: Fast speculative decoding via optimal transport,” in *NeurIPS*, 2023.
- [41] H. Jiang *et al.*, “LLMLingua: Compressing prompts for accelerated inference of large language models,” in *EMNLP*, 2023.
- [42] S. Anagnostidis *et al.*, “Dynamic context pruning for efficient and interpretable autoregressive transformers,” in *NeurIPS*, 2023.
- [43] B. Chen *et al.*, “Model cascading for code: A cascaded black-box multi-model framework for cost-efficient code completion with self-testing,” in *IJCNN*, 2025, pp. 1–9.
- [44] G. Xiao *et al.*, “Efficient streaming language models with attention sinks,” in *ICLR*, 2024.
- [45] R. V. W. Putra *et al.*, “Romanet: Fine-grained reuse-driven off-chip memory access management and data organization for deep neural network accelerators,” *IEEE TVLSI*, vol. 29, no. 4, pp. 702–715, 2021.
- [46] R. V. W. Putra, M. A. Hanif, and M. Shafique, “Drmap: A generic dram data mapping policy for energy-efficient processing of convolutional neural networks,” in *DAC*, 2020, pp. 1–6.
- [47] J. Li *et al.*, “Large language model inference acceleration: A comprehensive hardware perspective,” *arXiv preprint arXiv:2410.04466*, 2024.
- [48] T. J. Ham *et al.*, “A<sup>3</sup>: Accelerating attention mechanisms in neural networks with approximation,” in *HPCA*, 2020, pp. 328–341.
- [49] Y. Lin *et al.*, “Towards fully 8-bit integer inference for the transformer model,” in *IJCAI*, 2020, pp. 3759–3765.
- [50] A. Marchisio *et al.*, “Swifttron: An efficient hardware accelerator for quantized transformers,” in *IJCNN*, 2023, pp. 1–9.
- [51] C. Kachris, “A survey on hardware accelerators for large language models,” *Applied Sciences*, vol. 15, no. 2, p. 586, 2025.
- [52] J. Pan *et al.*, “A survey of research in large language models for electronic design automation,” *ACM TODAES*, vol. 30, no. 3, 2025.
- [53] M. Shao *et al.*, “Survey of different large language model architectures: Trends, benchmarks, and challenges,” *IEEE Access*, 2024.
- [54] P. E. Calzada *et al.*, “Verilogdb: The largest, highest-quality dataset with a preprocessing framework for llm-based rtl generation,” *arXiv preprint arXiv:2507.13369*, 2025.
- [55] M. Liu, N. Pinckney, B. Khailany, and H. Ren, “Verilogval: Evaluating large language models for verilog code generation,” in *ICCAD*, 2023.
- [56] Y. Lu *et al.*, “Rtlm: An open-source benchmark for design rtl generation with large language model,” in *ASP-DAC*, 2024, pp. 722–727.
- [57] M. Liu *et al.*, “Chipnemo: Domain-adapted llms for chip design,” *arXiv preprint arXiv:2311.00176*, 2023.
- [58] S. Liu *et al.*, “Rtlcoder: Fully open-source and efficient llm-assisted rtl code generation technique,” *IEEE TCAD*, 2024.
- [59] H. Huang *et al.*, “Towards llm-powered verilog rtl assistant: Self-verification and self-correction,” *arXiv preprint arXiv:2406.00115*, 2024.
- [60] Y. Zhao, H. Zhang, H. Huang, Z. Yu, and J. Zhao, “Mage: A multi-agent engine for automated rtl code generation,” in *DAC*, 2025, pp. 1–7.
- [61] H. Wu *et al.*, “Chateda: A large language model powered autonomous agent for eda,” *IEEE TCAD*, vol. 43, no. 10, pp. 3184–3197, 2024.
- [62] Z. Wang *et al.*, “Veridispatcher: Multi-model dispatching through pre-inference difficulty prediction for rtl generation optimization,” *arXiv preprint arXiv:2511.22749*, 2025.
- [63] —, “Netdetox: Adversarial and efficient evasion of hardware-security gns via rl-llm orchestration,” *arXiv preprint arXiv:2512.00119*, 2025.
- [64] —, “Llms and the future of chip design: Unveiling security risks and building trust,” in *ISVLSI*, 2024, pp. 385–390.
- [65] W. Xiao *et al.*, “Trojanloc: Llm-based framework for rtl trojan localization,” *arXiv preprint arXiv:2512.00591*, 2025.
- [66] S. Tarek *et al.*, “Bugwhisperer: Fine-tuning llms for soc hardware vulnerability detection,” in *VTS*, 2025, pp. 1–5.
- [67] Z. Wang *et al.*, “Vericontaminated: Assessing llm-driven verilog coding for data contamination,” *arXiv preprint arXiv:2503.13572*, 2025.
- [68] —, “Verileaky: Navigating ip protection vs utility in fine-tuning for llm-driven verilog coding,” *arXiv preprint arXiv:2503.13116*, 2025.
- [69] —, “Salad: Systematic assessment of machine unlearning on llm-aided hardware design,” *arXiv preprint arXiv:2506.02089*, 2025.
- [70] B. Zhou *et al.*, “Tinyllava: A framework of small-scale large multimodal models,” *arXiv preprint arXiv:2402.14289*, 2024.
- [71] S. Zhang *et al.*, “Biomedclip: a multimodal biomedical foundation model pretrained from fifteen million scientific image-text pairs,” 2025.
- [72] C. Li *et al.*, “Llava-med: Training a large language-and-vision assistant for biomedicine in one day,” 2023.
- [73] A. Basit *et al.*, “Rover: Autonomous open-vocabulary object searching in unexplored environments using vlm-driven scene understanding,” in *IJCNN*, 2025, pp. 1–8.
- [74] R. V. W. Putra and M. Shafique, “Spikenas: A fast memory-aware neural architecture search framework for spiking neural network-based embedded ai systems,” *IEEE TAI*, pp. 1–12, 2025.
- [75] R. V. W. Putra, P. Wickramasinghe, and M. Shafique, “Qslm: A performance- and memory-aware quantization framework with tiered search strategy for spike-driven language models,” *arXiv preprint arXiv:2601.00679*, 2026.
- [76] R.-J. Zhu *et al.*, “SpikeGPT: Generative pre-trained language model with spiking neural networks,” *TMLR*, 2024.
- [77] R. V. W. Putra, S. Iftikhar, and M. Shafique, “Qsvit: A methodology for quantizing spiking vision transformers,” in *IJCNN*, 2025, pp. 1–8.
- [78] M. Yao *et al.*, “Spike-driven transformer v2: Meta spiking neural network architecture inspiring the design of next-generation neuromorphic chips,” in *ICLR*, 2024.