TinyAgent: Quantization-aware Model Compression and Adaptation for On-device LLM Agent Deployment

Blind Author Review [1]

Abstract

Deploying LLMs on edge devices is challenging due to stringent memory resources and compute constraints. In edge applications, existing deployment solutions for LLM agents disaggregate the fine-tuning process for domain-specific adaptation and the post-training model compression process. As a result, it requires extensive experimentation to find a readily available model compression technique that minimizes a fine-tuned model's performance loss while satisfying a target hardware's memory constraints. To address this problem, we propose TinyAgent, which optimizes the deployment workflow by using a quantization-aware model compression technique for specialized decision-making LLM agents under resource-constrained environments. Our approach takes into account both deployment-time hardware constraints and challenges in post-training quantization during fine-tuning. Experimental results demonstrate that our approach not only achieves $8\times$ less memory usage to make LLM inference possible across a variety of edge devices, but also consistently speeds up LLM inference by up to $4.5\times$ without compromising accuracy.

1. Introduction

The recent global explosion of connected Internet of Things (IoT) devices presents opportunities to integrate LLM agents interactively with users on a massive scale. LLM agents are specialized instances of LLMs designed to perform specific tasks autonomously. Deploying LLM agents to edge devices will require both domain-specific adaptation in the form of fine-tuning and LLM model compression to fit the models in memory. Various techniques like instruction fine-tuning (Wei et al., 2021) and RLHF (Ziegler et al., 2019) are developed to align LLMs with human preferences and application-specific behaviors to serve specialized roles (Python coder, disk storage manager, etc.) in a system. Deploying LLM-based autonomous agents holds significant potential for sectors requiring real-time processing and decision-making capabilities, such as healthcare, smart cities, and industrial automation (Wang et al., 2024). However, conventional workflows for deploying instruction-aligned LLMs are not flexible enough to adapt to resource constraints, while ensuring sufficient internal domain-specific knowledge. Typical solutions require accelerator clusters in cloud services, potentially exposing user data to service providers, and are growingly energy inefficient. On-device deployment solutions, on the other hand, require no data transfer to the cloud and better protect user privacy with specialized models running on energy-efficient hardware at the edge.

Running LLM inference, even with smaller-scale models such as LLaMA2-7B (Touvron et al., 2023) or LLaMA3-8B (Meta, 2024) requires up to 18GB of memory. This presents a formidable challenge for on-device deployment since they have limited computational resources and memory capacities. For example, Nvidia Jetson TX2, with only 8GB of LPDDR4 host memory and 32GB of flash memory, cannot accommodate all model weights not to mention the commonly used KV cache (Pope et al., 2023). Therefore, the challenge of deploying LLM agents can be characterized as two-fold. The first challenge lies in compressing full-sized LLMs to fit edge devices. To be viable for deployment, the storage requirements of LLM models must be significantly reduced to satisfy given resource and memory limitations (Dhar et al., 2024). The second challenge is ensuring that each LLM agent retains internal domain-specific knowledge to conduct sophisticated decision making. Without adequate LLM fine-tuning, LLM agents might suffer from insufficient internal knowledge. As a result, this will sacrifice the collaborative performance of the LLM agents as a whole. Therefore, these two challenges necessitate innovative frameworks that compress and fine-tune LLM agents at the same time.

We address these open challenges by proposing TINYAGENT, a novel integration of TrimLLM, a LLM compression method using progressive layer-dropping techniques (Hu et al., 2024) with activation-aware weight quantization (AWQ) (Lin et al., 2023) in a hardware-aware manner to address memory bottleneck challenges during LLM agent deployment. TrimLLM's layer-dropping technique, combined with AWQ Quantization, allows for a significant reduction

in model size with minimal compromise in performance.

To demonstrate the efficacy of our solution, we chose a LLM agent application for video-monitoring memory management for several reasons. First, it exemplifies a typical memory-hungry edge application with stringent resource constraints, making it an ideal candidate for testing our quantization-aware model compression techniques. Second, video-monitoring applications require continuous and adaptive decision-making, where LLM agents can significantly enhance performance and efficiency. By combining TrimLLM's layer-dropping technique and AWQ in TinyAgent, each LLM agent is able to retain their respective internal knowledge domain such as python coding or memory management, while making LLM agent inference efficient enough to be viable across a variety of edge device. In comparison to conventional rule-based memory management systems, which might trigger low video frame rates whenever a memory usage limit is exceeded, our LLM agent system is more adaptive and responsive to real-time changes in memory usage.

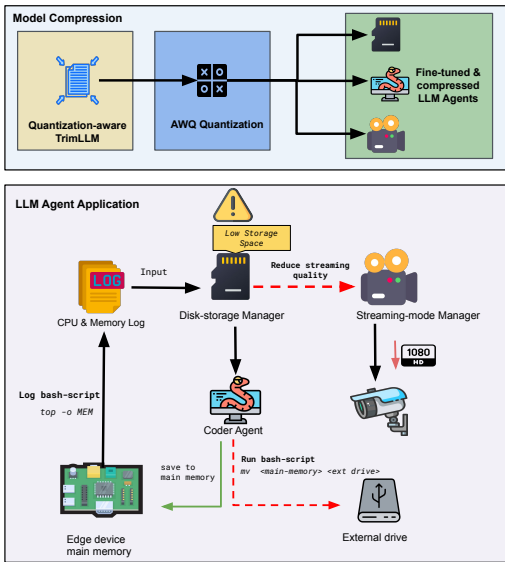

Figure 1. Proposed workflow for TinyAgent. Our system is built on top of AIOS (Mei et al., 2024), a framework of LLM operating system, for video-monitoring application.

2. Related Work

Layer Dropping. Previous studies have explored the potential of layer-dropping by compressing a foundational model during the pre-training phase (Zhang & He, 2020) to expedite training and enhance efficiency. Recent works also find there exits layer-wise sparsity in LLM where not all layers are needed for task-specific use cases (Men et al., 2024; Gromov et al., 2024). TrimLLM (Hu et al., 2024) is the state-of-the-art layer-dropping compression technique, which selectively drops less important layers during a fine-tuning process, and allows LLMs to adapt to specialized knowledge domains simultaneously. TinyAgent builds on top of TrimLLM and improves it by taking hardware constraints and PTQ challenges into consideration when specializing LLM agents to perform domain-specific tasks.

Post-training Quantization. PTQ is particularly effective for LLMs deployed in edge devices, optimizing inference speed under constrained computational and memory resources. AWQ (Lin et al., 2023), or Activation-aware Weight Quantization, optimizes weights bit-widths by selectively using INT4 representations (instead of FP16 floating point formats) in specific layers of LLM models. Specifically, AWQ is based on the observation that not all weights contribute equally to the performance of LLMs, a finding similar to what previously discussed in Xiao et al. (2023a). Therefore, by optimizing only non-critical layers, AWQ enables model compression while minimizing accuracy loss.

Pruning. Latest pruning algorithms for LLMs exploit either structured and unstructured sparsity at different levels of granularity (Liu et al., 2023; Sun et al., 2023; Frantar & Alistarh, 2023; Ma et al., 2023; Syed et al., 2023; Ashkboos et al., 2024; Xia et al., 2023). TinyAgent does not use pruning techniques, because it is hard to achieve measured speedup on edge devices without sparsity-aware kernels or computational units, not to mention the relatively large pruning configuration search space for accuracy vs. memory saving trade off given a hardware memory constraint.

LLM Agent Powered Applications. Many LLM agents applications have already demonstrated robust reasoning capabilities in specific domains, such as robotics and software engineering. For example, chatDev (Qian et al., 2023) could facilitate nuanced task delegation, while manually assigning distinct profiles to each agent. TidyBot (Wu et al., 2023) could design personalized household cleaning tasks, while adhering to user's preference on object placement and orientation. On a more general level, the widely received autoGPT (Yang et al., 2023) breaks down a wide range of tasks into subroutines for effective execution. However, it is important to note that these existing LLM agent deployments are limited to cloud deployment and, therefore, not exposed to the resource constraints of on-device deployment.

LLM Deployment Optimizations. Several approaches were proposed to address memory issues during LLM edge deployment. Dhar et al. (2024) and Edge-MoE (Yi et al., 2023) involve partially storing "secondary" weights in external storage, while maintaining "primary" weights in main memory. Somewhat similarly, Yin et al. (2024) manages KV cache in chunks to further optimize LLM contextual memory management. Therefore, previous works have not explored compressing the model itself prior to deployment in edge devices. Other hardware-aware (Dao et al., 2022; Sun et al., 2024; Zhang et al., 2024; Ye et al., 2024b) and

system (Kwon et al., 2023; Xiao et al., 2023b; Ye et al., 2024a) optimizations are also orthogonal to our proposed method to further optmize deployment-time throughput.

3. TinyAgent

Our deployment workflow begins by compressing the LLM model by executing quantization-aware TrimLLM techniques and AWQ sequentially on the cloud using GPUs. Once the model is compressed, we are ready to deploy our finalized and fine-tuned LLM as our LLM agents in applications like the security camera monitoring at the edge.

The proposed model compression technique first determines the minimum number of layers to drop from a pre-trained model like LLaMA3 (Meta, 2024). After identifying the target device's memory limitations, TinyAgent uses TrimLLM's techniques to iteratively compress the model until two criteria are met: (1) model size is reduced to smaller than the memory constraint, (2) a noticeable performance degradation ($> 1\%$) is detected. Therefore, TinyAgent's design is well-suited for edge devices due to its ability to customize models for specific knowledge domain uses.

To meet the two criteria, the next step is to use a calibration dataset and an activation-based metric to efficiently compute the importance score for each layer and eliminate non-essential layers as TrimLLM does (appendix A.1). TinyAgent makes the best use of TrimLLM's adaptability by taking a target device's hardware memory constraint into account during model adaptation (appendix A.2). The target device's memory constraint can therefore be translated into a model size that the LLM will be compressed into.

For the second part of model compression, we specifically chose AWQ quantization as our PTQ technique, because it enables effective model compression without significant accuracy loss. We sequentially apply AWQ quantization after TrimLLM, not only to further minimize model size but, more importantly, because TrimLLM performs fine-tuning during its compression process, which involves updates to the model's weights. This sequential order of execution will avoid inherent limitations of low-precision general matrix multiplication (GEMM) in INT4 after PTQ, which will impede the efficacy of TrimLLM and lead to significant performance degradation.

However, when applying AWQ with distribution-shifted calibration dataset, a TrimLLM trained with the above techniques could result in overflow problem when grid searching the scaling factor. Upon further examination, we find that fine-tuning on the initial and last few layers can exacerbate the well-observed outlier phenomenon in LLM quantization (Dettmers et al., 2022; Xiao et al., 2023a). The initial and last few layers are often times retained during TrimLLM's training process and incur even larger outliers when middle layers are dropped. This effectively makes the already significant outlier problem even worse (Figure 2) and TrimLLM more dataset-sensitive during the quantization phase.

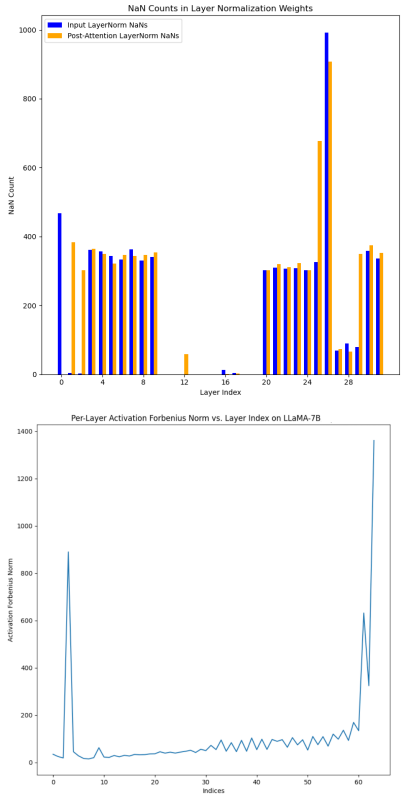

Figure 2. Number of NaNs per layer (up) and activation's Forbenius norm per layer (bottom) when applying AWQ to TrimLLM. We see a strong correlation between the "size" of each activation tensor measured by Forbenius norm and the number of NaN occurs. In TrimLLM, outliers emerging from activation tensors with lager "sizes"are will be pushed to larger values.

To address this, we propose a simple yet effective solution by freezing the initial and later layers during TrimLLM training to prevent activation outliers from becoming excessively large. By adapting this technique, NaN values disappear when applying AWQ to a model fine-tuned on MedMCQA (Pal et al., 2022), with PubMedQA (Jin et al., 2019) as the calibration dataset.

4. Experiments and Results

QA Benchmarks. In this section, we test TinyAgent's efficacy when applying AWQ to TrimLLM. The accuracy, memory consumption, and inference latency of the compressed models are reported in Table 1, considering the LLaMA-7B model across a variety of benchmarks, including: ScienceQA (Johannes Welbl, 2017), MedMCQA (Pal et al., 2022), and FinanceQA (Bharti, 2023) respectively.

Table 1. Performance comparison and inference-time memory usage of LLaMA-7B variants when applying AWQ to TrimLLM on domain-specific tasks. The numerical values are percentage in accuracy. Throughputs are measured on A100 GPU, NVIDIA Jetson TX2, Raspberry Pi4 4GB and Pi5 8GB with sequence length 512 and batch size 1. Symbol '-' means the same numerical numbers across different hardware. 'OOM' refers to out-of-memory error due to limited RAM on the target device. For LLaMA-7B-TINYAGENT, we use quantization-aware layer dropping with 50% model compression ratio.

models	SciQ	MedMCQA	FinanceQA	Hardware	Throughput (tokens/s)	Final Mem (↓)
LLaMA-7B						
w/o training	89.7	22.4	33.6	A100	42.3	16GB (100%)
+ Full-FT	95.6	54.6	45.1	A100	42.3	100%
+ Full-FT	-	-	-	Jetson TX2	OOM	100%
+ Full-FT	-	-	-	Raspberry Pi5	OOM	100%
+ Full-FT, LLM.int8	93.6	52.0	44.9	A100	29.6	$> 50\%$
+ Full-FT, LLM.int8	-	-	-	Jetson TX2	0.8	$> 50\%$
+ Full-FT, LLM.int8	-	-	-	Raspberry Pi5 8GB	< 0.1	$> 50\%$
+ Full-FT, AWQ-int4	93.0	50.7	42.1	A100	115.3	$> 25\%$
+ Full-FT, AWQ-int4	-	-	-	Jetson TX2	1.6	$> 25\%$
+ Full-FT, AWQ-int4	-	-	-	Raspberry Pi5 8GB	0.9	$> 25\%$
LLaMA-7B-TINYAGENT (50%)						
w/o PT compression	**94.2**	**53.1**	**43.6**	A100	103.1	$\geq 50\%$
w/o PT compression	-	-	-	Jetson TX2	5.7	$\geq 50\%$
w/o PT compression	-	-	-	Raspberry Pi5 8GB	2.5	$\geq 50\%$
w/o PT compression	-	-	-	Raspberry Pi4 4GB	OOM	$\geq 50\%$
+ AWQ-int4	91.5	49.2	40.5	A100	**188.7**	$>$ **13%**
+ AWQ-int4	-	-	-	Jetson TX2	6.7	$>$ **13%**
+ AWQ-int4	-	-	-	Raspberry Pi5 8GB	3.1	$>$ **13%**
+ AWQ-int4	-	-	-	Raspberry Pi4 4GB	1.4	$>$ **13%**

The results show that TinyAgent not only enables LLMs to be deployed on edge devices including NVIDIA Jetson TX2, Raspberry Pi4 4GB, Pi5 8GB, with as much as $8\times$ model compression ratio in terms of memory consumption, but also consistently achieve up to $7.1\times$ inference speedup on Jetson TX2 and $4.5\times$ speedup on A100 in comparison with the other SOTA LLM compression baselines with $< 1\%$ loss in accuracy across many datasets. This performance improvement is becuase TinyAgent accelerates by reducing model depth, saving significant number of MACs, whereas many SOTA model compression techniques including LLM.int8 (Dettmers et al., 2022) and AWQ (Lin et al., 2023) rely heavily on specialized low-precision computational units for acceleration.

LLM Agent Application. In the video-monitoring application, the streaming rate and memory usage were analyzed with and without LLM agents. In conventional rule-based memory management, the streaming rate decreases stepwise as memory thresholds are crossed (Fig. 4a). The available flash and host memory (Fig. 4b) and external disk memory (Fig. 4c) deplete rapidly. With LLM agents, the streaming rate adjusts dynamically based on available memory, checking and adjusting every 200 seconds (Fig. 4d), resulting in smoother memory management (Figs. 4e, f). This simulation shows that without LLM agents, the system runs out of memory in 2000 seconds, while with LLM agents, the system extends the operation time to 4600 seconds.

Conclusion

In this paper, we present a novel framework for deploying large language model (LLM) agents on edge devices by integrating TrimLLM's layer-dropping and AWQ. Our key contribution involves compressing the model in a hardware-aware way by taking a target device's hardware memory constraints into account during model adaptation. During post-training quantization, we identified and addressed the layer-dropping exacerbation problem in a quantization-aware way by freezing the initial and later layers, preventing excessive activation outliers and preserving model performance. Our experimental results show our deployment workflow achieves significant memory saving and inference speedup in LLM inference across various edge devices. By achieving such results, we demonstrated the effectiveness of our framework in a LLM-agent powered video-monitoring memory management application across a variety of resource-constrained hardware platforms showcasing the practicality and effectiveness of LLM agents in real-time, resource-constrained environments.

Impact Statement

The implementation of LLM agents in edge devices facilitates more sophisticated and immediate data processing. This can lead to improved responsiveness and accuracy in critical applications such as medical diagnostics and emergency response systems, where rapid decision-making is essential.

By reducing the memory and computing requirements, this research extends the benefits of advanced AI technologies to regions and sectors where access to cutting-edge hardware is limited. This democratization of technology can help bridge the digital divide, providing more equitable access to AI benefits. Ultimately, our contribution will narrow the divide for deploying edge intelligence in large-scale and real-world IoT applications.

References

Aghajanyan, A., Zettlemoyer, L., and Gupta, S. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*, 2020.

Ashkboos, S., Croci, M. L., do Nascimento, M. G., Hoefler, T., and Hensman, J. Slicegpt: Compress large language models by deleting rows and columns, 2024.

Bharti, G. gbharti/finance-alpaca, 2023. URL https://huggingface.co/datasets/gbharti/finance-alpaca. Accessed: 2023-09-20.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35:30318–30332, 2022.

Dhar, N., Deng, B., Lo, D., Wu, X., Zhao, L., and Suo, K. An empirical analysis and resource footprint study of deploying large language models on edge devices. In *Proceedings of the 2024 ACM Southeast Conference on ZZZ*, pp. 69–76, 2024.

Frantar, E. and Alistarh, D. Sparsegpt: Massive language models can be accurately pruned in one-shot. 2023.

Gromov, A., Tirumala, K., Shapourian, H., Glorioso, P., and Roberts, D. A. The unreasonable ineffectiveness of the deeper layers. *arXiv preprint arXiv:2403.17887*, 2024.

Hu, L., Kailkhura, B., Rosing, T., and Zhang, H. Trim-llm: Progressive layer dropping for domain-specific llms. 2024.

Jin, Q., Dhingra, B., Liu, Z., Cohen, W. W., and Lu, X. Pubmedqa: A dataset for biomedical research question answering. *arXiv preprint arXiv:1909.06146*, 2019.

Johannes Welbl, Nelson F. Liu, M. G. Crowdsourcing multiple choice science questions. 2017.

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

Lin, J., Tang, J., Tang, H., Yang, S., Dang, X., and Han, S. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2023.

Liu, Z., Wang, J., Dao, T., Zhou, T., Yuan, B., Song, Z., Shrivastava, A., Zhang, C., Tian, Y., Re, C., et al. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pp. 22137–22176. PMLR, 2023.

Ma, X., Fang, G., and Wang, X. Llm-pruner: On the structural pruning of large language models. *arXiv preprint arXiv:2305.11627*, 2023.

Mei, K., Li, Z., Xu, S., Ye, R., Ge, Y., and Zhang, Y. Aios: Llm agent operating system. *arXiv e-prints, pp. arXiv–2403*, 2024.

Men, X., Xu, M., Zhang, Q., Wang, B., Lin, H., Lu, Y., Han, X., and Chen, W. Shortgpt: Layers in large language models are more redundant than you expect. *arXiv preprint arXiv:2403.03853*, 2024.

Meta. Introducing meta llama 3: The most capable openly available llm to date, May 2024. URL https://ai.meta.com/blog/meta-llama-3/.

Pal, A., Umapathi, L. K., and Sankarasubbu, M. Medmcqa: A large-scale multi-subject multi-choice dataset for medical domain question answering. In *Conference on Health, Inference, and Learning*, pp. 248–260. PMLR, 2022.

Pope, R., Douglas, S., Chowdhery, A., Devlin, J., Bradbury, J., Heek, J., Xiao, K., Agrawal, S., and Dean, J. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5, 2023.

Qian, C., Cong, X., Yang, C., Chen, W., Su, Y., Xu, J., Liu, Z., and Sun, M. Communicative agents for software development. *arXiv preprint arXiv:2307.07924*, 2023.

Sun, H., Chen, Z., Yang, X., Tian, Y., and Chen, B. Triforce: Lossless acceleration of long sequence generation with hierarchical speculative decoding. *arXiv preprint arXiv:2404.11912*, 2024.

Sun, M., Liu, Z., Bair, A., and Kolter, J. Z. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

Syed, A., Guo, P. H., and Sundarapandiyan, V. Prune and tune: Improving efficient pruning techniques for massive language models. 2023.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):1–26, 2024.

Wei, J., Bosma, M., Zhao, V. Y., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., and Le, Q. V. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

Wu, J., Antonova, R., Kan, A., Lepert, M., Zeng, A., Song, S., Bohg, J., Rusinkiewicz, S., and Funkhouser, T. Tidybot: Personalized robot assistance with large language models. *Autonomous Robots*, 47(8):1087–1102, 2023.

Xia, M., Gao, T., Zeng, Z., and Chen, D. Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694*, 2023.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pp. 38087–38099. PMLR, 2023a.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023b.

Yang, H., Yue, S., and He, Y. Auto-gpt for online decision making: Benchmarks and additional opinions. *arXiv preprint arXiv:2306.02224*, 2023.

Ye, Z., Chen, L., Lai, R., Zhao, Y., Zheng, S., Shao, J., Hou, B., Jin, H., Zuo, Y., Yin, L., Chen, T., and Ceze, L. Accelerating self-attentions for llm serving with flashinfer, February 2024a. URL https://flashinfer.ai/2024/02/02/introduce-flashinfer.html.

Ye, Z., Lai, R., Lu, B.-R., Lin, C.-Y., Zheng, S., Chen, L., Chen, T., and Ceze, L. Cascade inference: Memory bandwidth efficient shared prefix batch decoding, February 2024b. URL https://flashinfer.ai/2024/02/02/cascade-inference.html.

Yi, R., Guo, L., Wei, S., Zhou, A., Wang, S., and Xu, M. Edgemoe: Fast on-device inference of moe-based large language models. *arXiv preprint arXiv:2308.14352*, 2023.

Yin, W., Xu, M., Li, Y., and Liu, X. Llm as a system service on mobile devices. *arXiv preprint arXiv:2403.11805*, 2024.

Zhang, M. and He, Y. Accelerating training of transformer-based language models with progressive layer dropping. *Advances in Neural Information Processing Systems*, 33: 14011–14023, 2020.

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

Ziegler, D. M., Stiennon, N., Wu, J., Brown, T. B., Radford, A., Amodei, D., Christiano, P., and Irving, G. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

A. Appendix

A.1. TrimLLM Layer-wise Importance Score

Sensitivity-based Scoring. The first method is a performance scanning based on a small calibration dataset. Before each time a layer is to be dropped, a small subset of the fine-tuning dataset's validation set is sampled as the calibration dataset. For each layer, its importance score is the reciprocal of the model's performance after dropping the layer. Calibration scanning gives the importance score of any layer i and the expression is presented in Equation 1, where $a_i \in [0, 100]$ is the accuracy of the model after dropping the i-th layer and δ is a small positive number such that $\frac{100}{1+\delta^2}$ is the maximum importance score when $a_i = 0$.

$$s_{i,\text{scan}} = \frac{100 - a_i}{(1+\delta^2) + (1+\delta)\, a_i} \tag{1}$$

Activation-based Scoring. The second method is to make activation-norm comparison on different layers' activations. Recent studies (Dettmers et al., 2022; Xiao et al., 2023a) have shown preserving information carried by activations is critical to model's performance when it comes to model compression techniques.

In our work, our goal is to only preserve activations that are meaningful to the knowledge domain of interest. We can drop the rest to trade the model's generality for efficiency and specialization. A new metric is therefore needed to quantify the importance of an activation.

Our assumptions consist of two parts: (1) there exists a feature space $\mathcal{X}$ and a corresponding low intrinsic dimension (Aghajanyan et al., 2020). (2) activation tensors are dense with mostly small-magnitude elements and a few large-magnitude outliers based on widely recognized observations (Dettmers et al., 2022; Xiao et al., 2023a).

Among common matrix norms including the $\ell_{2,1}$ norm, the Forbenius norm and the nuclear norm, at the same numerical value, nuclear norm should be the best metrics for directly measuring the rank of a matrix which is defined as the sum of the singular values of the matrix: $||W||_* = \sum_i \sigma_i$. The nuclear norm is a convex surrogate for the rank function and is often used in rank minimization problems. However, The nuclear norm introduces extra computational overhead because it requires the computation of the SVD of the matrix. Computing the SVD is computationally intensive, especially for large matrices, as it has a complexity of $O(\min(nm^2, mn^2))$ for $m \times n$ matrix. As a result, we use the Forbenius norm to approximate the nuclear norm. By expanding the Forbenius norm with SVD, it follows: $||W||_F = \sqrt{\sum_i \sigma_i^2}$.

Therefore, we choose the Forbenius norm to identify activations with high-rank representations and sparse domain-specific knowledge. Dropping the one with highest norm is analogous to Forbenius norm minimization. Let $\{\|\boldsymbol{x}_j\|_F\}$ be the set of Forbenius norm for all remaining layers in the model $f(\cdot)$. This activation-norm importance score can be expressed in the form of Equation 2 such that $s_{i,\text{norm}} \in (0, 100]$.

$$s_{i,\text{norm}} = \frac{100 \min\{\|\boldsymbol{x}_j\|_F\}}{\|\boldsymbol{x}_i\|_F} \tag{2}$$

A.2. TinyAgent Adaptability

On the flexibility side, as we can see from Figure 3, quantization and pruning offers a very limited set of operating points corresponding to each of the bit precision scheme for each model. Since sparsity ratio in pruning can not be easily translated into memory saving during inference, pruning oftentimes gives even fewer operating points in the trade-off space. In contrast, the Pareto frontiers of TrimLLM span a wide range of operating points. As a result, TinyAgent is more flexible and is capable of fitting a model to a wide spectrum of hardware.

A.3. Benchmarking Baseline Inference Speed

From table 2, it can be seen that LLM inference speed with LLaMA-2-7B is very slow across different edge devices after INT4 quantization. TinyAgent offers a better solution by further compressing LLM agents when deploying them at the edge with much higher throughtput (Table 2) and saves more space for memory-hungry application like the security camera monitoring one elaborated in Section 4.

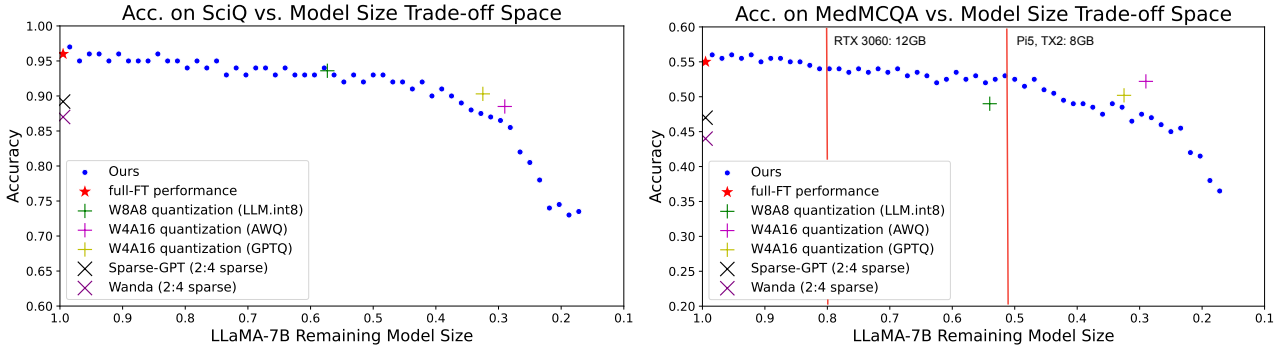

Figure 3. The Parento Frontier of LLaMA-7B-TINYAGENT on SciQ and MedMCQA. TINYAGENT has a much wider spectrum of operating points to fit the model into different hardware with competitive performance. The layer dropping strategy employed is the best strategy reported in TrimLLM.

Table 2. Inference performance of executing LLaMA-2 7B with INT4 quantization on different edge devices. Raspberry Pi 4B results are referenced from Yi et al. (2023).

Devices	Performance (Tokens/Second)
Raspberry Pi 4B 1GB	0.01
Raspberry Pi 4B 2GB	0.01
Raspberry Pi 4B 8GB	0.11
Raspberry Pi 5 8GB	0.28

A.4. TinyAgent Powered Security Camera Memory Management System

Comparisons between TinyAgent powered security camera memory management system and a rule-based system is shown in Figure 4, where we see TinyAgent can adjust video streaming rate much more responsively to system memory usage, allowing the application to run much longer than the rule-based system.

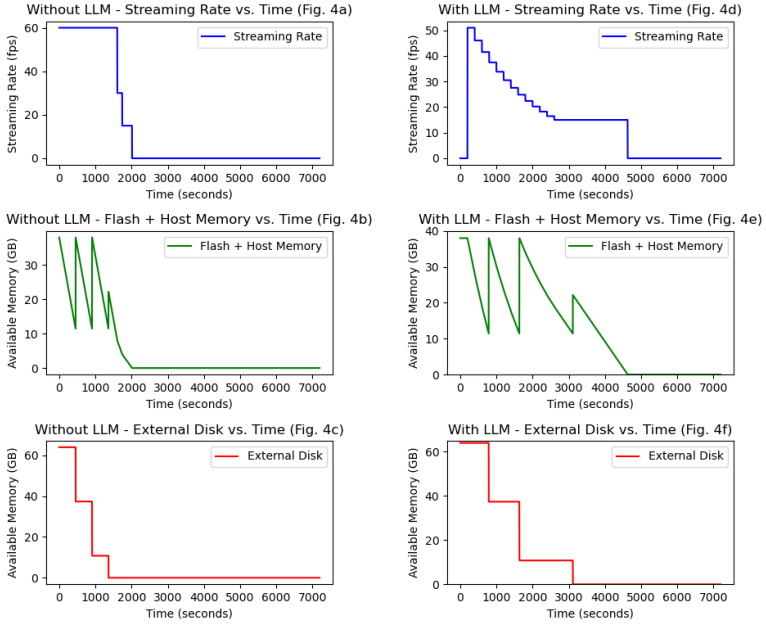

Figure 4. Video-monitoring Memory Management Performance

FPS Adaptation with Multiple Concurrent Applications. In practice, systems often run multiple applications simultane-

ously, making host memory availability unpredictable at times. Using LLM agents allows the video-monitoring application to adapt the streaming rate in real-time based on fluctuating memory conditions. The equation used for dynamic streaming rate adjustment is:

$$\text{streaming\_rate}(t) = \max\left(\frac{a \cdot \text{main\_memory}(t) + b \cdot \text{external\_memory}(t)}{c}, \text{min\_rate}\right)$$

Here, $\text{main\_memory}(t)$ is the available main memory at time t, which can be analyzed by LLM agents to make real-time decisions. However, it requires intelligence to extract and interpret this information, especially when other applications are running concurrently.