Balcony: A Lightweight Approach to Dynamic Inference of Generative Language Models

Benyamin Jamialahmadi[*1], **Parsa Kavehzadeh**[*1], **Mehdi Rezagholizadeh**[2†], **Parsa Farinneya**[1], **Hossein Rajabzadeh**[3], **Aref Jafari**[1], **Boxing Chen**[1], and **Marzieh S. Tahaei**[1]
[1]Huawei Noah's Ark Lab
[2]Advanced Micro Devices, Inc. (AMD)
[3]University of Waterloo
{benyamin.jami76, parsareal}@gmail.com

Abstract

Deploying large language models (LLMs) in real-world applications is often hindered by strict computational and latency constraints. While dynamic inference offers the flexibility to adjust model behavior based on varying resource budgets, existing methods are frequently limited by hardware inefficiencies or performance degradation. In this paper, we introduce **Balcony**, a simple yet highly effective framework for **depth-based dynamic inference**. By freezing the pretrained LLM and inserting additional transformer layers at selected exit points, Balcony maintains the full model's performance while enabling real-time adaptation to different computational budgets. These additional layers are trained using a straightforward self-distillation loss, aligning the submodel outputs with those of the full model. This approach requires significantly fewer training tokens and tunable parameters, drastically reducing computational costs compared to prior methods. When applied to the LLaMA3-8B model, using only 0.2% of the original pretraining data, Balcony achieves minimal performance degradation while enabling significant speedups. Remarkably, we show that Balcony outperforms state-of-the-art methods such as Flextron and Layerskip as well as other leading compression techniques on multiple models and at various scales, across a variety of benchmarks.[1]

1 Introduction

We are entering an era of rapid advancements in generative foundation models, with tens or even hundreds of billions of parameters emerging at an accelerating pace Touvron et al. (2023); Bai et al. (2023); DeepSeek-AI et al. (2024). The demand for their deployment is greater than ever. However, deploying these models in real-world applications, particularly in industrial environments, whether on edge devices or in cloud settings, is constrained by strict computational and latency requirements.

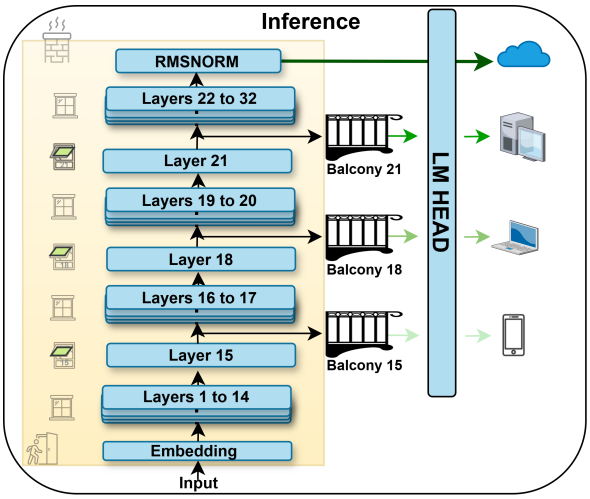

Figure 1: An overview of the *Balcony* inference. Balcony preserves the base model's performance while enabling efficient, on-the-fly adaptation to different computational budgets.

These constraints are not static; they fluctuate based on task complexity, sample difficulty, user load, and inference budget. To address these challenges, *dynamic inference* has become increasingly important. This approach allows models to adjust their computational demands on the fly, eliminating the need for expensive and time-consuming retraining. As a result, it enables more efficient and scalable deployment.

Dynamic inference is typically achieved in two ways: 1) width-based methods, which adjust the number of active neurons and attention heads in a model Kudugunta et al. (2023); Yu and Huang (2019a), and 2) depth-based methods, which selectively reduce the number of transformer layers used during inference Kavehzadeh et al. (2024).

However, GPUs are inherently optimized for par-

[*] Equal contribution.
[†] Work done while at Huawei.
[1]Please find our code and models at https://github.com/benyaminjami/Balcony-LLaMA/tree/finetuning.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 24853–24867
November 4-9, 2025

allelized deep computations, making depth-based compression significantly more favorable in terms of speed and efficiency. Reducing depth results in fewer sequential operations, which directly translates to lower latency and improved throughput. This effect is illustrated in Figure 2, where for a fixed parameter budget, reducing depth consistently yields greater speed improvements compared to reducing width.

As a result, prior research has explored methods that introduce elasticity along the depth dimension Xin et al. (2020); Kavehzadeh et al. (2024); Elhoushi et al. (2024). Despite its advantages, depth-based dynamic inference introduces a critical challenge: it often degrades both full-model accuracy and sub-model performance. This occurs because existing methods require extensive perturbation to the base model.

Specifically, at each exit point, the sub-model must serve two competing roles: producing an intermediate representation for the next layer (to support larger sub-models) while simultaneously generating a refined output representation at the current layer. These competing objectives at the training time lead to *conflicting gradients*, introducing a trade-off that ultimately compromises the accuracy of both the sub-models and the full model. Rotem et al. (2023); Kavehzadeh et al. (2024).

In this paper, we propose Balcony, a simple yet highly efficient framework for depth-based dynamic inference. By freezing the pretrained LLM and adding a decoder layer at each exit point, Balcony preserves the base model's performance while enabling efficient, on-the-fly adaptation to different computational budgets. While there is an inherent trade-off between sub-model accuracy and computational cost, we show that adding a single transformer layer and sharing the LM head, across all sub-models achieves an optimal balance. We train the Balcony layers using a straightforward self-distillation loss, aligning Balcony layers outputs with those of the full model.

In contrast to prior works, which also train the base model, freezing enables lossless performance on the original base model and allows for efficient tuning due to the low number of tunable parameters. Additionally, freezing facilitates seamless adaptation to different computational budgets during inference by simply switching the Balcony layers.

Our experiments demonstrate that for LLaMA3-8B, freezing the base model and tuning only the Balcony layers (2.5% of full model parameters for each balcony layer), using just 0.2% of the data compared to the 15T tokens used for full model pretraining, yields remarkably strong results.

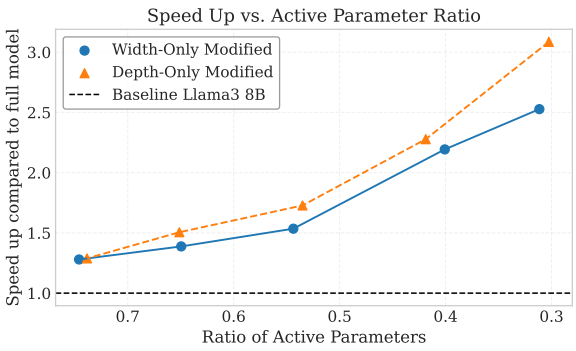

Figure 2: Speed-up as a function of the ratio of active parameters in modified versions of Llama 3 8B. Comparisons are shown between width-only and depth-only modifications, with the unmodified baseline included for reference.

Notably, Balcony surpasses state-of-the-art (SoTA) dynamic inference methods, including Flextron Cai et al. (2024) and LayerSkip Elhoushi et al. (2024), while using a minimal training approach that requires a much simpler training flow, significantly fewer training tokens and a much smaller number of tunable parameters (see Related Work for details on the training strategies used in these methods). This paper makes the following key contributions:

- Introducing Balcony, a depth-based dynamic inference framework that employs single transformer layers at exit points while freezing the base model.
- Efficient tuning of Balcony through a self-distillation loss on a small dataset, significantly reducing training costs compared to prior methods while outperforming Flextron, LayerSkip and SoTA compression methods.
- An extensive evaluation of the proposed framework through ablation studies on various components of the method, including pretraining on a 1B-parameter LLM.

2 Related work

Dynamic inference has gained significant attention over the past decade, with various methods proposed to make CNNs and small-scale encoder based NLP models dynamic Yu et al. (2018); Yu and Huang (2019b); Li et al. (2021); Cai et al.

(2020); Xin et al. (2020); Hou et al. (2020); Kusupati et al. (2022). These approaches often require sophisticated and heavy training, are architecture-dependent and hence have not yet been effectively translated to modern large-scale generative LLMs. This paper focuses specifically on generative LLMs, emphasizing approaches applied within the realm of generative AI.

MatFormer Kudugunta et al. (2023) introduces a nested architecture along the width that enables the extraction of multiple sub-models by incorporating a nested Feed Forward Network (FFN) block structure from a single trained network. The largest reported MatFormer model is an 850M decoder-only language model (MatLM), from which smaller models ranging from 582M to 850M parameters can be derived. MatFormer demonstrates superior performance compared to independently trained models and older methods like OFA Cai et al. (2020) and DynaBERT Hou et al. (2020).

Building on this, Flextron proposed a more sophisticated approach that integrates a nested elastic structure with input-adaptive routing, allowing automatic token processing through sub-networks. However, their training strategy is highly sophisticated: first, they train a large number of submodels using their nested architecture. Then, they train routers to select the appropriate submodels based on a given budget (for static inference) or dynamically for each token (for adaptive inference). However, training the router is challenging due to limited gradient flow. To address this, they introduce an auxiliary model that provides the necessary signals to facilitate the router's training. We demonstrate that, with significantly fewer training tokens and a far simpler training approach, our submodels achieve superior performance.

Similar to Balcony, SortedLLaMA Kavehzadeh et al. (2024) explores elasticity along the depth dimension by extending the SortedNet Valipour et al. (2023) training technique to generative LLMs. They eliminate the need for pretraining by replacing standard fine-tuning with sorted fine-tuning. LayerSkip Elhoushi et al. (2024) is another recent dynamic depth approach that is used along speculative decoding for faster inference. During training, the method employs layer dropout with increasing rates for deeper layers and applies an early exit loss on all transformer layers while sharing the LM head.

Note that the approaches mentioned above perturb the base model to create a nested design of sub-

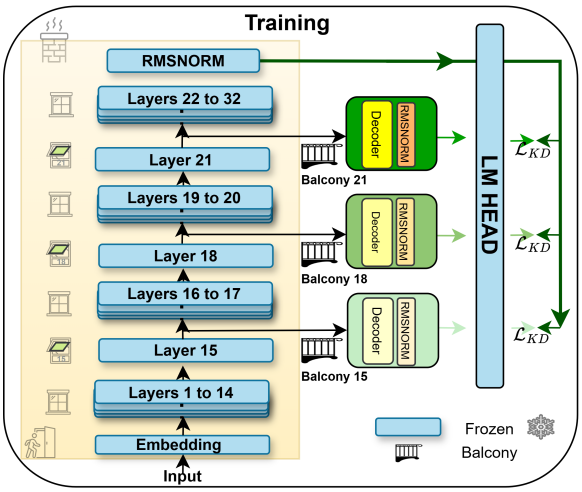

Figure 3: Training in the Balcony framework: By freezing the pretrained base LLM and adding a decoder layer at each exit point, Balcony can outperform SoTA with significantly fewer training tokens.

models, allowing multiple submodels to be hosted within a shared architecture while reducing memory overhead. This, in turn, leads to performance degradation in the full model. In contrast, our method achieves the same objective without enforcing a nested structure. Instead, we freeze the base model and train only the Balcony Exit and auxiliary tokens, ensuring that the base model's performance remains intact and uncompromised by nested biases. Additionally, since all submodels share the same base model, the memory overhead of loading multiple submodels simultaneously or switching between them remains manageable.

3 Balcony framework

In this section, we present the architecture, training, and inference methods for the proposed framework. Consider a model M with N layers, where each layer is represented by the function $X_i = f(X_{i-1}, W_i)$, with $i \in [1, N]$ indexing the layers, X_{i-1} being the input to layer i (with dimensions $B \times S \times D$ for batch size, sequence length, and embedding dimension), and W_i denoting the parameters for that layer. In this context, the term "layers" includes not only traditional transformer blocks but also other depthwise modules, such as Mamba Gu and Dao (2023) and Mixture of Experts (MoE) Fedus et al. (2022) blocks.

Our objective is to make the model M dynamic in depth, allowing it to adapt to user-defined goals, such as latency, memory, and accuracy. To achieve this, we introduce exit points based on the desired inference time and resource budget. For instance,

in cloud computing, we can select exit layers depending on the query-per-second rate. Similarly, on edge devices, one can extract submodels depending on the available computation budget.

An **exit point** in our framework refers to a designated layer within the model where inference can be halted, allowing a prediction to be generated using a lightweight *Balcony module* instead of processing all layers of the model. We introduce a set of **exit points** $\mathcal{E} \subseteq \{1, 2, \ldots, N\}$, each associated with a *Balcony module*. At each exit point $j \in \mathcal{E}$, the intermediate representation X_j is processed by a Balcony module, defined as:

$$\mathcal{X'}_j = f_b(X_j, \mathcal{W'}_j), \tag{1}$$

where f_b represents a Balcony module composed of a decoder layer(e.g a transformer block) followed by a normalization layer, $\mathcal{W'}_j$ is the set of parameters for the Balcony module, and $\mathcal{X}'_j$ is the output of the Balcony module at exit point j.

At each exit point, the intermediate output X_j is forwarded to the corresponding Balcony module $f_b(X_j, \mathcal{W'}_j)$ then passes through an RMSNorm layer, followed by a shared LM head and a softmax function to produce the final probabilities. The LM head is the same as that of the original model and is shared across all submodels. The transformer layer in Balcony uses the same architecture as the base model.

During training, self-distillation is used to align the probability distribution $p(\cdot; W_{1:j}, \mathcal{W}'_j,)$ of each submodel with the full model's distribution $p(\cdot; W_{1:N})$. This is achieved by minimizing the Kullback-Leibler (KL) divergence across all possible next tokens at position t. The objective function is given by:

$$\mathcal{L} = \sum_{j \in \mathcal{E}} \mathrm{KL}\left(p(\cdot; W_{1:N}) \parallel p(\cdot; W_{1:j}, \mathcal{W}'_j)\right), \tag{2}$$

where $\mathcal{W}'_j$ represents the balcony layer inserted after layer j, and $\mathrm{KL}(\cdot \parallel \cdot)$ denotes the Kullback-Leibler divergence.

Note that both the base model parameters $W_{1:N}$ and the LM head remain frozen during training. In our framework, Balcony layers are initialized from the last transformer layer of the trained model (see Section 4.3 for the impact of this initialization). The rationale behind this choice is that the last layer is already aligned with the shared LM head, which helps in seamlessly integrating the Balcony layers for effective submodel extraction.

Since the base model remains frozen, the training of Balcony layers is independent; each Balcony layer receives gradients based only on its corresponding submodel's loss. Consequently, while all Balcony layers can be trained in a single training round to minimize computation, they can also be added and trained individually without impacting any other submodel.

4 Experiments

4.1 Setup

Model Configuration We compare the performance of Balcony to SoTA methods using two models, LLaMA3-8B and LLaMA2-7B. The rationale behind selecting these models is that prior SoTA works have reported their results on them. To explore the methods at a smaller scale, we use an LLM with the same architecture as LLaMA3-1B and train it on FineWebEDU and Cosmopedia V2, which are part of the SmoLLM corpus Ben Allal et al. (2024) from Hugging Face. We refer to this model as LLM-1B. This model provides a baseline on which Balcony and sortedNet method are applied to. It also provides a baseline for our training from scratch experiment. For the LLM-1B model, we insert Balcony modules at layers 4 (S), 8 (M), and 12 (L), while for the LLaMA models, Balcony modules are placed at layers 15 (S), 18 (M), and 21 (L). In both cases, we refer to the full model as XL.

Training Details For fine-tuning, both LLaMA3-8B and LLaMA2-7B were trained with a batch size of 256 for 30K steps using a cosine learning rate scheduler with a maximum learning rate of $5e^{-4}$. The sequence length was set to 4,096 tokens, and the training corpus consisted of 31.5B tokens from Cosmopedia V2. LLM-1B followed the same fine-tuning setup but with a sequence length of 2,048 tokens and 15.7B tokens from Cosmopedia V2. For pretraining, LLM-1B was trained from scratch using FineWebEDU and Cosmopedia V2, with a batch size of 384 for 500K steps, a sequence length of 2,048 tokens, and a learning rate of $5e^{-4}$ following a trapezoidal scheduler. For pretraining the LLM-1B model from scratch, we used 384B tokens sourced from FineWebEDU and Cosmopedia V2.

Baselines For the baseline in this paper, Flextron is used as the SoTA method in width-based dynamic inference, and LayerSkip and Sorted are considered for depth-based dynamic inference. In the comparison, the number of non-embedding parameters is reported. Regarding speedup, for depth-based

Method	Model	# Params	ARC-E	LMBD	PIQA	WG	MMLU (5)	Avg. (Drop)
Base	**Llama2-7B-Full model**	6.5B	76.3	71.1	78.1	69.1	45.9	68.1
Balcony	**Balcony-XL**	6.5B	76.3	71.1	78.1	69.1	45.9	68.1(**0**)
	Balcony-L	4.4B	72.0	67.0	75.9	67.5	45.0	65.5 (**-2.6**)
	Balcony-M	3.8B	68.9	61.3	75.2	66.0	43.0	62.9 (**-5.2**)
	Balcony-S	3.2B	64.9	54.9	73.5	63.8	39.8	59.4 (-8.7)
LayerSkip	**Layerskip-XL**	6.5B	76.5	70.5	77.6	70.3	43.2	67.6 (-0.5)
	Layerskip-L	4.4B	68.5	65.9	73.7	66.4	42.4	63.4 (-4.7)
	Layerskip-M	3.8B	61.4	55.1	71.1	65.8	42.4	59.2 (-8.9)
	Layerskip-S	3.2B	50.3	43.5	68.6	63.9	37.8	52.8 (-15.3)
Flextron	**Full**	6.5B	75.1	71.5	77.5	69.1	45.1	67.7 (-0.4)
	Dynamic-0.7	4.1B	68.6	65.1	76.1	63.7	42.2	63.1 (-5)
	Dynamic-0.6	3.9B	67.1	63.8	74.9	62.2	39.4	61.5 (-6.6)
	Dynamic-0.5	3.4B	66.5	62.9	74.1	62.0	36.8	60.5 (**-7.6**)
Base	**LLaMA3-8B**	6.9B	81.8	71.2	80.1	73.6	65.0	74.3
Balcony	**Balcony-XL**	6.9B	81.8	71.2	80.1	73.6	65.0	74.3 (**0**)
	Balcony-L	4.7B	77.0	67.0	77.4	72.3	64.0	71.6 (**-2.7**)
	Balcony-M	4.4B	75.7	62.7	76.7	69.7	64.4	69.9 (**-4.4**)
	Balcony-S	3.4B	70.0	54.5	75.0	68.1	48.3	63.2 (**-11.1**)
LayerSkip	**LayerSkip-XL**	6.9B	79.7	72.5	80.1	73.8	59.6	73.2 (-1.1)
	LayerSkip-L	4.7B	73.2	61.8	77.2	71.0	59.1	68.5 (-5.8)
	LayerSkip-M	4.4B	68.6	60.8	74.2	70.1	59.3	66.7 (-7.6)
	LayerSkip-S	3.4B	59.0	47.7	70.4	66.4	37.8	56.3 (-18)
Flextron	**Full-Flextron-8B**	6.4B	71.7	69.7	79.4	68.8	35.4	65
	Dynamic-0.7	4.3B	67.0	64.8	75.9	64.1	30.0	60.4
	Dynamic-0.6	3.9B	66.2	63.7	76.1	62.7	29.1	59.6
	Dynamic-0.5	3.3B	65.0	62.5	75.8	61.8	27.1	58.4
Open-Source	**OpenLLaMA-7Bv2**	6.5B	69.5	63.8	79.9	66.0	40.4	63.92
	OpenLLaMA-3Bv2	3.2B	63.7	59.1	78.1	63.3	25.7	58.0
	NutePrune	3.2B	51.7	-	71.0	57.5	-	-
	Compresso-compressed LLaMA-7B	4.5B	66.0	-	72.9	63.4	25.9	-
	LaCo-compressed LLaMA2-7B	4.7B	-	-	69.8	-	26.5	-

Table 1: We evaluate the downstream task performance of Balcony, comparing it against Flextron, LayerSkip, open-source models, and other compression methods. For LayerSkip, we evaluated their publicly available models. For all other baselines, the results are taken from their papers. We report zero-shot accuracy on ARC-easy, LAMBADA, PIQA, and WinoGrande, along with 5-shot performance on MMLU. For LayerSkip, we evaluated their publicly available models. #Params denotes the number of non-embedding parameters.

	Method	Number of tunable parameters	Training cost in tokens	Percentage of pretraining tokens
LLaMA2-7B	**Pretraining**	7B	2T	100%
	Balcony	$3 \times 200\text{M} = \mathbf{600M}$	**31B**	**1.5%**
	Flextron	7B	89.9B (Main model excluding the router)	4.50%
	LayerSkip	7B	52B	2.6%
LLaMA3-8B	**Pretraining**	8B	15T	100%
	Balcony	$3 \times 200\text{M} = \mathbf{600M}$	**31B**	**0.2%**
	LayerSkip	8B	419B	2.8%

Table 2: Training cost comparison for Flextron, LayerSkip, and Balcony methods applied to LLaMA2-7B and LLaMA3-8B models, with costs presented in terms of tokens. In the Balcony method, the base model is frozen, and only the Balcony layers are updated. Each Balcony layer is a single transformer layer, comprising 202M parameters. The training cost for Flextron is taken from the paper Yu and Huang (2019a). Additionally, the table includes the percentage of pretraining cost relative to the total pretraining cost for each method.

methods, similar speedup can be achieved across different methods with the same number of parameters. However, for width-based methods, the same number of parameters results in lower speedup, as shown in Figure 2. Since Flextron is not open-sourced, speedup comparisons cannot be reported.

Nonetheless, for the same number of parameters, Balcony is expected to deliver better speedup. Furthermore, we contrast our method with several prominent open-source and compression model families, specifically, OpenLLaMA Geng and Liu (2023), Compresso Guo et al. (2023), NutePruner Li et al. (2024), SliceGPT Ashkboos et al. (2024), and LaCo Yang et al. (2024). Evaluation is performed on ARC (Clark et al., 2018), BoolQ (BQ) (Clark et al., 2019), OpenbookQA (OBQA) (Mihaylov et al., 2018), PIQA (Bisk et al., 2020a), WinoGrande (WG) (Sakaguchi et al., 2021), LAMBADA (LMBD) (Paperno et al., 2016b), 5-shot MMLU (Hendrycks et al., 2020), and 10-shot HellaSwag (HS) (Zellers et al., 2019a). These evaluations were conducted using the LM-Evaluation-Harness repository (Gao et al., 2024).

4.2 Results

Analysis of speedup in dynamic depth vs dynamic width To evaluate the effectiveness of depth-based model pruning in our Balcony framework, we conducted an empirical analysis comparing the impact of width pruning and depth pruning on model latency.The latency measurements in Figure 2 were obtained using vLLM Kwon et al. (2023) for efficient deployment and were performed on an NVIDIA V100 32GB GPU, prompt size of 32, output size of 2048. These measurements compare both depth and width pruning on LlaMA38B. In depth pruning, the number of hidden layers was reduced from 32 layers to fewer layers, leading to an almost linear reduction in the number of active parameters proportional to the number of layers. For width pruning, we initially reduced the intermediate size of the MLP block until it reached the hidden size, followed by reducing the number of attention heads.

Figure 2 clearly illustrates that for all tested parameter ratios, representing pruned models relative to the non-pruned base model, depth pruning consistently yields higher speed-ups compared to width pruning. This reinforces the effectiveness of depth-based pruning strategies in achieving significant latency reductions.

Balcony performance In this section, we assess Balcony's performance on several downstream tasks, as shown in Table 1. We use LLaMA2-7B as the baseline and compare it to Flextron and LayerSkip. The results for Flextron are taken from their original paper, while for LayerSkip, the dynamic models are open-sourced, so we conducted our own evaluation using the provided dynamic LLaMA-7B. For Flextron, we report the dynamic version, as it demonstrates superior performance compared to the static version. Among the three methods, Balcony is the only one that maintains the performance of the full model by freezing it during tuning. In contrast, both Flextron and LayerSkip experience a performance drop for their full models. For smaller submodels, with approximately 0.7x, 0.6x, and 0.5x of the non-embedding parameters, Balcony shows a significantly smaller performance drop compared to the baselines. The only exception is Balcony-S, which, with 3.2B parameters, experiences a 1.1% larger drop than the Dynamic Flextron model at 3.4B parameters.

The same evaluation is also performed on LLaMA-3-8B. Here, we compare the base model to those of Balcony and LayerSkip. Similarly, the LayerSkip results are obtained using their open-sourced model. For Flextron, their 8B model (Flextron-8B) does not come from the same base model and is therefore placed in a separate section of the paper. It can be observed that for all submodel sizes, from Small to XL, Balcony provides significantly less performance drop than LayerSkip across all submodels.

Figure 4 plots the trade-off between accuracy and the number of parameters for the Balcony-LLaMA7B family models and compares them against those of Flextron-Dynamic, Flextron-static, and LayerSkip, as well as post-hoc compression methods like Compresso, LLM-Pruner, SliceGPT, and LaCo. The Balcony model family achieves superior performance on both MMLU and ARC-E compared to all the baselines.

Balcony speedup Table 3 shows the latency of Balcony family models. The latency measurements in seconds are based on vLLM Kwon et al. (2023) for efficient deployment and conducted on an NVIDIA V100 32GB GPU. The latency is measured in seconds for 30 dummy input samples with a prompting length of 32, a generation length of 2048, a batch size of 2 and float16 precision.

Model	Full	0.7×	0.6×	0.5×
Balcony-Llama3-8B	58.16	41.82 (1.4×)	35.26 (1.7×)	30.37 (1.9×)
Balcony-Llama2-7B	52.54	37.23 (1.4×)	32.46 (1.6×)	27.99 (1.9×)

Table 3: Average latency (in seconds) for Balcony family models at different scaling factors. Speed-up factors (relative to the full model) are shown in parentheses.

	Model	# Params	ARC-C	ARC-E	BQ	LMBD-Op	LMBD-ST	OQA	PIQA	WG	HS(10)	MMLU(5)	Avg
	LLM-1B (Baseline)	973M	37.6	74.3	58.4	50.4	43.3	30	74.9	59.8	46.4	25.9	50.1
Pretraining	**Balcony-XL**	973M	38.1	72.7	60.0	47.9	40.2	27.8	74.4	56.2	45.5	25.1	**48.8**
	Sorted-XL	973M	36.6	71.8	59.8	43.6	35.6	28.8	72.8	53.67	43.5	25.6	47.2
	Balcony-L	790M	37.8	72.0	61.8	47.4	40.0	27.2	74.4	55.17	44.4	25.3	**48.6**
	Sorted-L	729M	35.3	71.8	60.9	43.1	35.6	28.6	71.8	54.38	42.5	24.8	46.9
	Balcony-M	547M	33.4	69.1	62.9	46.2	37.8	26.4	71.5	56.75	41.7	26.1	**47.2**
	Sorted-M	486M	32.1	68.1	60.3	42.2	32.7	26.8	70.4	54.06	40.4	24.8	45.2
	Balcony-S	304M	27	63.6	60.5	36.3	24.2	23.2	70.5	50.7	35.7	25.9	**41.8**
	Sorted-S	243M	26.5	61.6	60	31.8	20.0	21.0	67.4	50.59	34.0	27.1	40.0
Standard	**Balcony-XL**	973M	37.6	74.3	58.4	50.4	43.3	30.0	74.9	59.8	46.4	25.9	**50.1**
	Sorted-XL	973M	34.4	68.1	49.9	46.3	34.8	27.0	72.2	56.7	43.2	25.7	45.8
	Balcony-L	790M	32.8	66.3	61.4	45.1	36.4	25.4	71.3	57.6	41.3	26.3	**46.4**
	Sorted-L	729M	30.5	57.5	53.4	39.2	26.7	22.4	67.4	53.7	36.8	25.5	41.3
	Balcony-M	547M	24.9	58.7	61.4	29.5	20.5	21.8	67.8	53.4	34.3	27.2	**40**
	Sorted-M	486M	22.9	44.2	43.4	20.2	13.0	17.8	63.7	52.6	30.4	25.5	33.4
	Balcony-S	304M	22.3	51.7	62.0	19.3	8.2	17.4	64.1	52.2	30.3	24.6	**35.2**
	Sorted-S	243M	21.2	40.8	55.6	11.3	4.4	13.6	60.3	49.1	27.9	25.6	31.0

Table 4: Performance Comparison of pretraining from scratch and tuning a Pretrained model for Balcony and Sorted Approaches. For pretraining, both the Balcony and Sorted methods use 384B tokens. In the standard approach, 15B tokens are used to tune the baseline. In the Balcony method, the base model is frozen, while in the Sorted approach, the entire model is updated. The reported numbers represent 5-shot performance on MMLU, 10-shot performance on HellaSwag and zero-shot on other tasks. The notation #Params refers to the number of non-embedding parameters.

4.3 Ablation studies

Pretraining from scratch In this experiment, we train both the base model and the Balcony layers from scratch to assess the representational capacity of Balcony compared to a nested design. This approach is particularly useful when the entire pretraining budget is allocated to developing a dynamic model.

We begin by training the LLM-1B model from scratch, which serves as the baseline for normal training (see the Training Details in 4.1). Next, we train the Balcony layers alongside the base model using the same training budget. The Balcony layers are initialized randomly, and since we are training from scratch, we omit self-distillation. Instead, we use the average loss over all submodels as the training objective. The Balcony exit layers are placed after layers 4, 8, 12, and 16 of the model.

We also compare Balcony's representational capacity to SortedLLaMA, which applies sorting training only at the fine-tuning stage. To ensure a fair comparison, we perform the same pretraining with SortedLLaMA, but with a sorted objective function. The results, shown in Table 4, indicate that for all submodels, pretraining using the Balcony design outperforms the nested approach used in SortedLLaMA. However, it is important to note that when training from scratch, since the base model is not frozen, the accuracy of the resulting model, despite outperforming SortedLLaMA, is lower than the baseline model.

Furthermore, we compare pretraining with the standard efficient training method proposed by Balcony and evaluate it against tuning using the Sorted approach. Note that in Balcony, the base model is frozen, and only the Balcony module is updated, whereas in the Sorted approach, the entire model is updated. The results show that Balcony provides significantly higher accuracy across submodels compared to the Sorted approach.

Random Initialization, MLP-Only and Attention-Only Balcony A key question that arises is why we use a single transformer decoder layer and why we initialize it from the final layer of the full architecture. To investigate this, we conduct an ablation study by repeating the Balcony-LLM-1B experiment under different configurations to assess the impact of each module. First, we perform balcony training with randomly initialized balcony modules (transformer decoders) to evaluate the effect of initializing from the final layer on submodel performance. Additionally, we train balcony models with MLP-only and Attention-only modules, using the same set of intermediate layers in the LLM-1B model, to isolate the contributions of each component. Figure 5 presents the average results of these submodels across ARC-E, ARC-C, BoolQ, HellaSwag, Lambada, MMLU, OpenBookQA, PIQA, and Winogrande benchmarks. As shown,

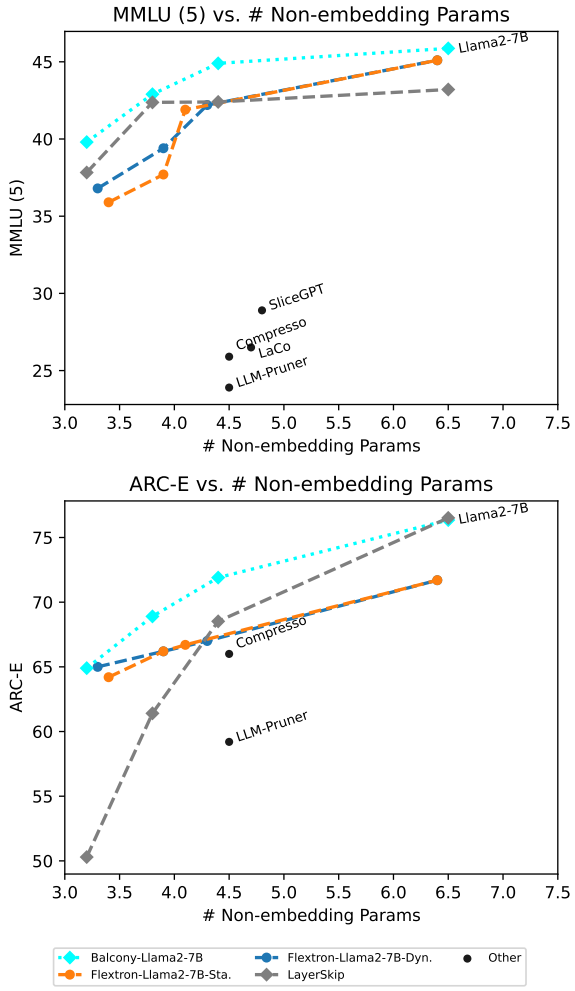

Figure 4: The Balcony-LLaMA2-7B model family demonstrates superior performance on MMLU and ARC-E compared to Flextron-Dynamic/Static, LayerSkip and post-hoc compression methods like Compresso, LLM-Pruner, SliceGPT, and LaCo.

standard balcony training with a transformer decoder initialized from the final layer outperforms both random initialization and the MLP-only and Attention-only variants.

Effect of Cross-Entropy Loss During post-training of the balcony modules, we used only KL-divergence loss between the frozen, pretrained full model's output and the outputs of the balcony modules. To examine the potential effect of incorporating Cross-Entropy (CE) loss in training balcony modules, we conducted an ablation study by training them with both KL-divergence and CE loss on the submodels of our LLM-1B model. In this experiment, we set the KL loss weight to 0.001. Figure 5 presents the results of standard balcony training (KL-only) and the KL + CE variant across different submodels and benchmarks. As shown, incorporating CE loss does not yield significant improvements in the performance of balcony submodels.

Effect of Freezing during Balcony Training During balcony training, we kept the pretrained backbone model weights frozen. To assess the impact of updating the main model's weights alongside the balcony modules, we conducted an ablation study where the backbone model's weights were also made trainable. Additionally, we included another dynamic inference baseline that does not freeze the backbone model's weights: Sorted Fine-Tuning (Kavehzadeh et al., 2024). Figure 5 presents the results of three setups: standard balcony training (frozen backbone), balcony training with an unfrozen backbone, and Sorted Fine-Tuning (unfrozen backbone) on different submodels of the LLM-1B model. As observed, training balcony modules while keeping the backbone model frozen not only preserves the performance of the full pretrained architecture but also outperforms both the unfrozen balcony and Sorted Fine-Tuning approaches across most submodels. More detailed results can be found in the appendix.

5 Conclusion

In this paper, we presented Balcony, a novel framework for depth-based dynamic inference that addresses key challenges in the deployment of large-scale language models under strict computational constraints. By introducing additional transformer layers at selected exit points and training them with a self-distillation loss, Balcony enables flexible, real-time adaptation to various computational budgets without compromising the performance of the full model. Our experiments demonstrate that Balcony achieves minimal performance degradation while using a small training budget. This efficiency represents a significant improvement over state-of-the-art methods. Balcony's simplicity, effectiveness, and minimal resource requirements position it as a promising approach to dynamic inference, paving the way for more efficient deployment of large-scale models across diverse hardware environments. Future work will explore extending Balcony and combining it with more efficient architectures, such as MoE and state space models like Mamba, to further enhance performance and scalability. Additionally, Balcony can be leveraged for self speculative decoding (Leviathan et al., 2023; Chen et al., 2023) to achieve speedup without degradation of model performance.

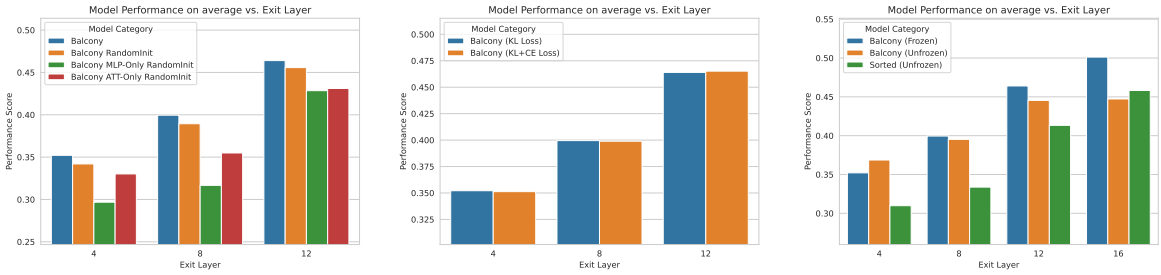

Figure 5: Ablation studies on the Balcony framework. From left to right: (1) Impact of random initialization and the contributions of MLP-only and Attention-only modules. (2) Effect of incorporating Cross-Entropy (CE) loss in self-distillation. (3) Effect of freezing the main architecture during post-training. The results are the average scores across ARC-E, ARC-C, BoolQ, HellaSwag, Lambada, MMLU, OpenBookQA, PIQA, and Winogrande.

6 Limitations

While Balcony minimizes performance degradation at reduced latencies, there remains an inherent trade-off between latency and model accuracy. For extremely low-latency requirements, further performance degradation might be inevitable and combining Balcony with width compression methods may become necessary.

While Balcony can be easily combined with certain compression strategies, like quantization, we have not explored how it interacts with techniques (e.g., low-rank factorization or pruning). Understanding these interactions could further optimize memory and speed.

Also, in our experiments, we only evaluate the model at predefined budgets. However, implementing token-level confidence-based exits could potentially yield better trade-offs between accuracy and latency, and help boost our performance even further.

Moreover, recent methods that utilize token-level routers such as Mixture-of-Depths (MoD) (Raposo et al., 2024) offer per-token, per-layer adaptivity that is theoretically more flexible than static subnetwork selection. However, these methods introduce major practical challenges at inference: token-wise routers lead to non-uniform compute paths that complicate batching, incur scatter/gather overhead. Moreover, the dynamic computation graphs inherent to token-level routing hinder compiler optimizations, so reported gains are often FLOP-based rather than true wall-clock latency. Translating these methods into robust, production-scale serving remains an open engineering problem.

References

Saleh Ashkboos, Maximilian L Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. 2024. Slicegpt: Compress large language models by deleting rows and columns. *arXiv preprint arXiv:2401.15024*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Loubna Ben Allal, Anton Lozhkov, Guilherme Penedo, Thomas Wolf, and Leandro von Werra. 2024. Smollm-corpus.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020a. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020b. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2020. Once-for-all: Train one network and specialize it for efficient deployment. *Preprint*, arXiv:1908.09791.

Ruisi Cai, Saurav Muralidharan, Greg Heinrich, Hongxu Yin, Zhangyang Wang, Jan Kautz, and Pavlo Molchanov. 2024. Flextron: Many-in-one flexible large language model. *Preprint*, arXiv:2406.10260.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *NAACL*.

DeepSeek-AI, :, Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, Huazuo Gao, Kaige Gao, Wenjun Gao, Ruiqi Ge, Kang Guan, Daya Guo, Jianzhong Guo, Guangbo Hao, Zhewen Hao, Ying He, Wenjie Hu, Panpan Huang, Erhang Li, Guowei Li, Jiashi Li, Yao Li, Y. K. Li, Wenfeng Liang, Fangyun Lin, A. X. Liu, Bo Liu, Wen Liu, Xiaodong Liu, Xin Liu, Yiyuan Liu, Haoyu Lu, Shanghao Lu, Fuli Luo, Shirong Ma, Xiaotao Nie, Tian Pei, Yishi Piao, Junjie Qiu, Hui Qu, Tongzheng Ren, Zehui Ren, Chong Ruan, Zhangli Sha, Zhihong Shao, Junxiao Song, Xuecheng Su, Jingxiang Sun, Yaofeng Sun, Minghui Tang, Bingxuan Wang, Peiyi Wang, Shiyu Wang, Yaohui Wang, Yongji Wang, Tong Wu, Y. Wu, Xin Xie, Zhenda Xie, Ziwei Xie, Yiliang Xiong, Hanwei Xu, R. X. Xu, Yanhong Xu, Dejian Yang, Yuxiang You, Shuiping Yu, Xingkai Yu, B. Zhang, Haowei Zhang, Lecong Zhang, Liyue Zhang, Mingchuan Zhang, Minghua Zhang, Wentao Zhang, Yichao Zhang, Chenggang Zhao, Yao Zhao, Shangyan Zhou, Shunfeng Zhou, Qihao Zhu, and Yuheng Zou. 2024. Deepseek llm: Scaling open-source language models with longtermism. *Preprint*, arXiv:2401.02954.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. 2024. Layer skip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research*, 23(1):5232–5270.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2024. A framework for few-shot language model evaluation.

Xinyang Geng and Hao Liu. 2023. Openllama: An open reproduction of llama.

Albert Gu and Tri Dao. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*.

Song Guo, Jiahang Xu, Li Lyna Zhang, and Mao Yang. 2023. Compresso: Structured pruning with collaborative prompting learns compact large language models. *arXiv preprint arXiv:2310.05015*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. 2020. Dynabert: Dynamic bert with adaptive width and depth. *Advances in Neural Information Processing Systems*, 33:9782–9793.

Parsa Kavehzadeh, Mojtaba Valipour, Marzieh Tahaei, Ali Ghodsi, Boxing Chen, and Mehdi Rezagholizadeh. 2024. Sorted llama: Unlocking the potential of intermediate layers of large language models for dynamic inference. In *Findings of the Association for Computational Linguistics: EACL 2024*, pages 2129–2145.

Sneha Kudugunta, Aditya Kusupati, Tim Dettmers, Kaifeng Chen, Inderjit Dhillon, Yulia Tsvetkov, Hannaneh Hajishirzi, Sham Kakade, Ali Farhadi, Prateek Jain, et al. 2023. Matformer: Nested transformer for elastic inference. *arXiv preprint arXiv:2310.07707*.

Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, et al. 2022. Matryoshka representation learning. *Advances in Neural Information Processing Systems*, 35:30233–30249.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding.

Changlin Li, Guangrun Wang, Bing Wang, Xiaodan Liang, Zhihui Li, and Xiaojun Chang. 2021. Dynamic slimmable network. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pages 8607–8617.

Shengrui Li, Junzhe Chen, Xueting Han, and Jing Bai. 2024. Nuteprune: Efficient progressive pruning with numerous teachers for large language models. *arXiv preprint arXiv:2402.09773*.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. *Preprint*, arXiv:1809.02789.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Nghia The Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernandez. 2016a. The lambada dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1525–1534.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016b. The lambada dataset: Word prediction requiring a broad discourse context. *Preprint*, arXiv:1606.06031.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. 2024. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*.

Daniel Rotem, Michael Hassid, Jonathan Mamou, and Roy Schwartz. 2023. Finding the sweet spot: Analysis and improvement of adaptive inference in low resource settings. *arXiv preprint arXiv:2306.02307*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. Winogrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8732–8740.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Mojtaba Valipour, Mehdi Rezagholizadeh, Hossein Rajabzadeh, Marzieh Tahaei, Boxing Chen, and Ali Ghodsi. 2023. Sortednet, a place for every network and every network in its place: Towards a generalized solution for training many-in-one neural networks. *arXiv preprint arXiv:2309.00255*.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. Deebert: Dynamic early exiting for accelerating bert inference. *arXiv preprint arXiv:2004.12993*.

Yifei Yang, Zouying Cao, and Hai Zhao. 2024. Laco: Large language model pruning via layer collapse. *arXiv preprint arXiv:2402.11187*.

Jiahui Yu and Thomas Huang. 2019a. Universally slimmable networks and improved training techniques. *Preprint*, arXiv:1903.05134.

Jiahui Yu and Thomas S Huang. 2019b. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1803–1811.

Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. 2018. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019a. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019b. Hellaswag: Can a machine really finish your sentence? *Preprint*, arXiv:1905.07830.

Appendix

A Benchmark Descriptions

This section provides a brief overview of the benchmarks used for evaluation.

A.1 ARC-E (AI2 Reasoning Challenge - Easy)

ARC-E consists of multiple-choice science questions designed to evaluate a model's reasoning ability (Clark et al., 2018). The dataset contains 7,787 questions, with 2,251 categorized as 'easy.' These questions require commonsense reasoning beyond simple retrieval.

A.2 LAMBADA

LAMBADA evaluates a model's ability to predict the last word of a passage, requiring deep contextual understanding (Paperno et al., 2016a). It consists of 10,022 book passages, where the final word is only guessable with full comprehension.

A.3 PIQA (Physical Commonsense Reasoning)

PIQA assesses a model's ability to reason about physical interactions (Bisk et al., 2020b). The dataset contains 16,000 training and 2,000 validation questions, evaluating how well models understand everyday physical tasks.

A.4 Winogrande

Winogrande is a large-scale dataset designed for commonsense reasoning using Winograd-style problems (Sakaguchi et al., 2020). It contains over 44,000 sentence-pair problems that require resolving ambiguous pronouns using contextual knowledge.

A.5 MMLU 5 (Massive Multitask Language Understanding)

MMLU tests knowledge and reasoning across 57 domains (Hendrycks et al., 2021). The MMLU-5 subset focuses on five key categories to assess broad cognitive abilities through multiple-choice questions ranging from elementary to expert level.

A.6 HellaSwag

HellaSwag Zellers et al. (2019b) is a large-scale reasoning benchmark that tests the ability of models to predict the most likely continuation of a sentence in everyday situations. The dataset contains multiple-choice questions designed to challenge models on contextual reasoning and world knowledge.

B Ablations

In order to select the best architecture for the **Balcony** model, ablation studies were conducted on different loss functions, activation functions, and parameter initialization methods.

B.1 Effect of CE and KL Losses

In the Balcony architecture, two possible loss functions can help align the Balcony layer's output with that of the full model. One end-to-end approach is to match the generated output using the **Cross-Entropy (CE)** loss. Another approach is to apply a distillation objective by aligning the logits of the full model and the Balcony layer using the **Kullback-Leibler (KL)** divergence loss.

From Figure 6, we observe that combining both KL divergence and CE loss leads to a slight but consistent decrease in performance across different exit layers and benchmarks.

B.2 Effect of Freezing

Depending on the training budget, different layers of the architecture can be either frozen or unfrozen. To analyze the trade-off between performance and training cost, we report benchmark evaluation scores for both the frozen and unfrozen versions of the main model alongside the Balcony layer. Additionally, we compare these results with Sorted (Kavehzadeh et al., 2024), which shares the same base structure as Balcony but without exit layers.

Interestingly, in Figure 7 we observe that the frozen Balcony model outperforms its unfrozen counterparts. This result suggests that freezing layers may help mitigate issues such as catastrophic forgetting and overfitting, which often occur when fine-tuning a larger network.

B.3 Effect of Random Inititialization

The initialization of a generative model plays a crucial role in its performance, especially under a limited training budget. The Balcony architecture consists of multiple submodules, including MLP layers, attention layers, and the Balcony exit layer.

From our evaluations on eleven different benchmarks, we observe that our chosen initialization strategy, where the Balcony layer is initialized using the last layer of the full model, outperforms all other initialization variations. The results are shown in Figure 8

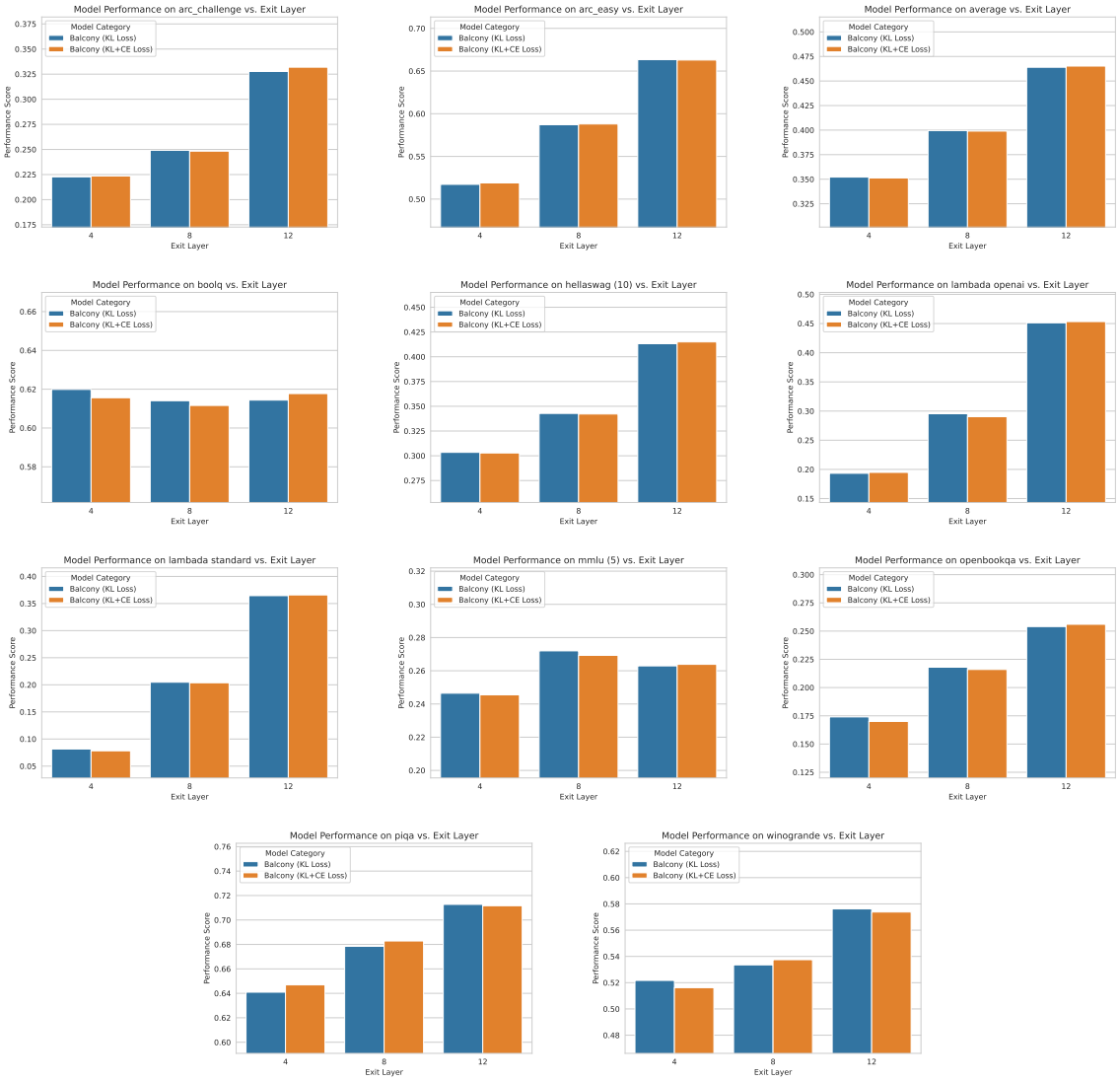

Figure 6: Ablation study on the impact of including CE loss in self-distillation. The results are the score across ARC-E, ARC-C, BoolQ, HellaSwag, Lambada, MMLU, OpenBookQA, PIQA, and Winogrande benchmarks.

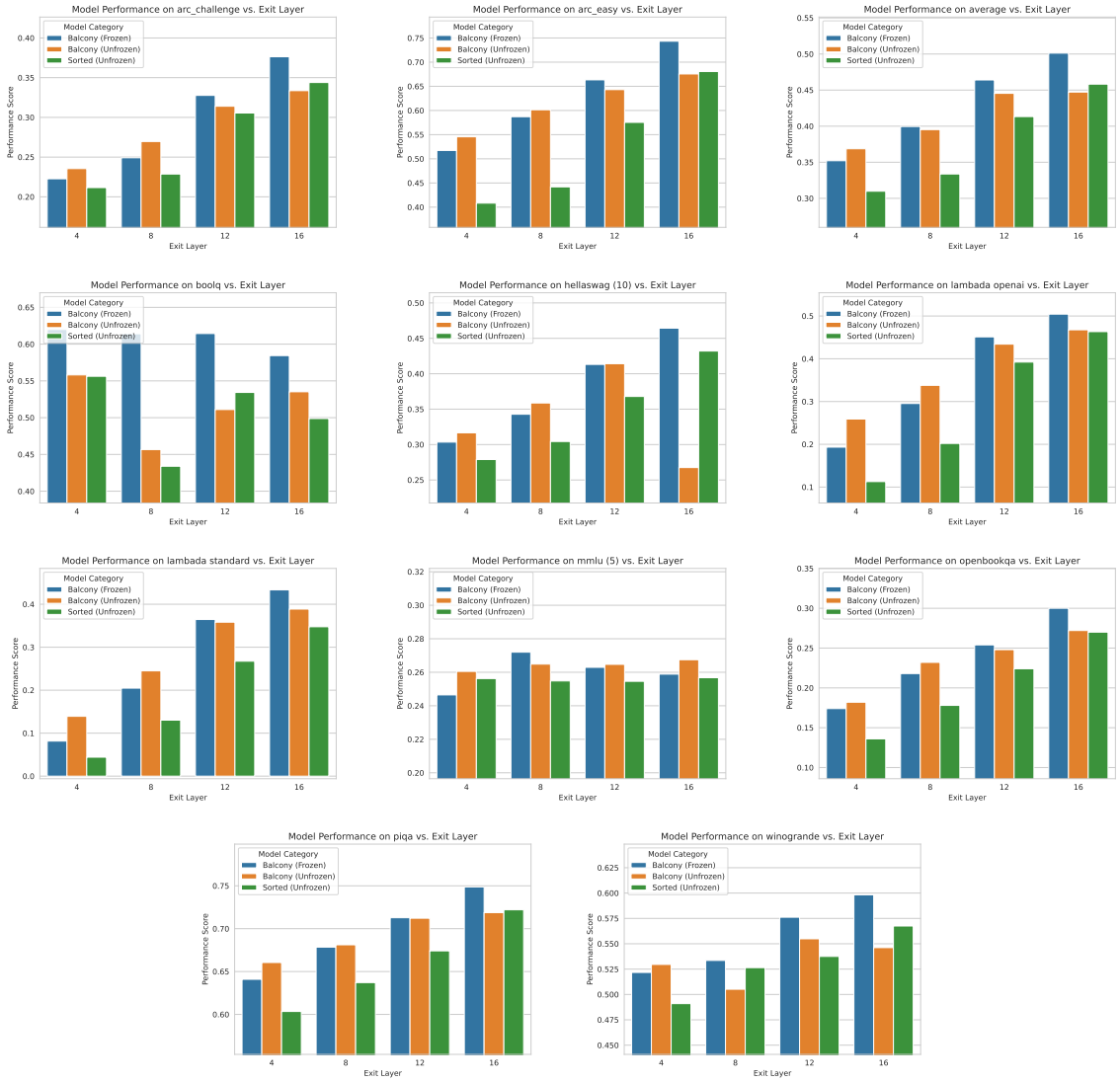

Figure 7: Ablation study on the impact of freezing the main architecture during post training. The results are the score across ARC-E, ARC-C, BoolQ, HellaSwag, Lambada, MMLU, OpenBookQA, PIQA, and Winogrande benchmarks.

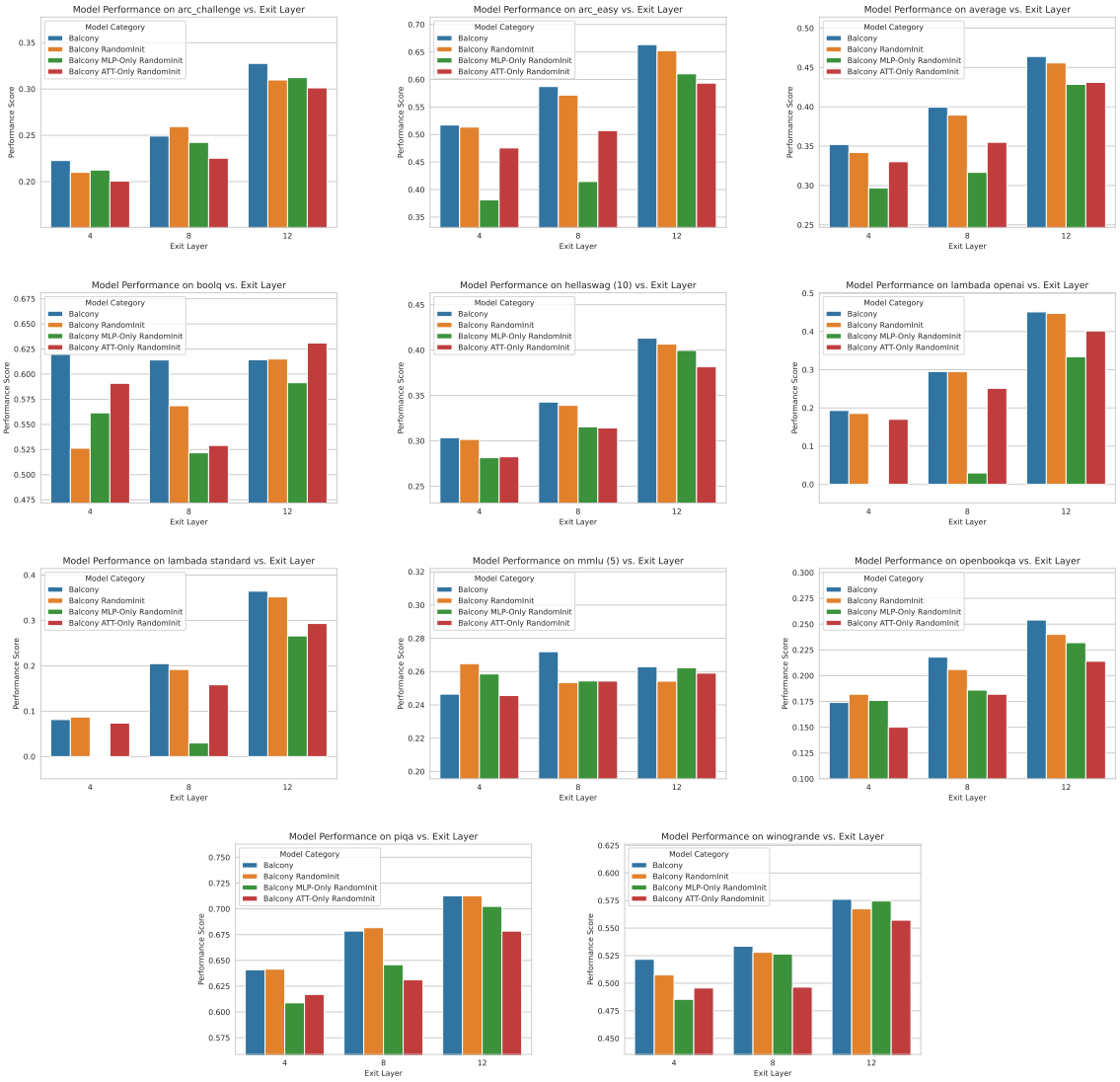

Figure 8: Ablation study on the impact of random initialization of balcony modules compared to regular balcony training starting from the full model final transformer layer weights. Also study on the effect of each MLP and Self-Attention modules in the balcony submodels' performance. The results are the score across ARC-E, ARC-C, BoolQ, HellaSwag, Lambada, MMLU, OpenBookQA, PIQA, and Winogrande benchmarks.