
Dynamic layer selection in decoder-only transformers

Theodore Glavas^{123*} Joud Chataoui^{123*} Florence Regol¹²³ Wassim Jabbour¹

Antonios Valkanas¹²³ Boris N. Oreshkin^{4†} Mark Coates¹²³

¹McGill University ²International Laboratory on Learning Systems (ILLS)

³Mila Québec AI Institute ⁴Amazon Science

{theodore.glavas, joud.chataoui

florence.robert-regol, wassim.jabbour, antonios.valkanas} @mail.mcgill.ca

boris.oreshkin@gmail.com, mark.coates@mcgill.ca

Abstract

The vast size of Large Language Models (LLMs) has prompted a search to optimize inference. One effective approach is dynamic inference, which adapts the architecture to the sample-at-hand to reduce the overall computational cost. We empirically examine two common dynamic inference methods for natural language generation (NLG): layer skipping and early exiting. We find that a pre-trained decoder-only model is significantly more robust to layer removal via layer skipping, as opposed to early exit. We demonstrate the difficulty of using hidden state information to adapt computation on a per-token basis for layer skipping. Finally, we show that dynamic computation allocation on a per-sequence basis holds promise for significant efficiency gains by constructing an oracle controller. Remarkably, we find that there exists an allocation which achieves equal performance to the full model using only 23.3% of its layers on average.

1 Introduction

The distribution and reuse of pre-trained LLMs means that inference now dominates the computational cost over the model’s lifecycle [1, 2]. As such, a variety of techniques have been proposed to improve LLM inference efficiency [3, 4]. While some approaches such as quantization and structural pruning can be readily applied with minor modifications, the auto-regressive nature of natural language generation (NLG) imposes new challenges for input-adaptive techniques such as dynamic networks. **First**, the generation output is highly sensitive to error introduced by reducing computation for any one token. Interdependencies cause error to compound by (a) relying on incorrect previously predicted tokens, and (b) filling missing KV cache entries with approximations [5, 6]. Despite some empirical studies [7, 8], the existing literature fails to unequivocally answer whether early exit is better than layer skipping at minimizing the error in the model’s hidden states.

Second, the large output vocabulary space in NLG renders the evaluation of the classification head at intermediate layers prohibitively costly. Instead, lightweight controllers route a token using the hidden state [5, 6, 9]. It is not clear that existing token-level layer skipping controllers derive a genuine benefit from processing the hidden state.

Third, per-token adaptation adds overhead to each autoregressive pass, which accumulates over long generations. Per-token controllers must therefore be lightweight networks to mitigate this effect [5, 6, 9, 10]. We investigate in Section 3.3 whether it is possible to reduce computation, with a limited performance drop, by making sequence-level decisions that only need to be evaluated once.

In this work, we conduct a series of three experiments and making the following contributions[‡]:

*Equal contribution

†This work is not related to the author’s position at Amazon.

‡The experiment code is available on this Github repository.

1. We compare the early exit and layer skip mechanisms on a fixed model. Our results show that layer skipping is the preferable strategy for minimizing the accumulated hidden state error.
2. We examine the efficacy of per-token layer skipping based on the hidden state, using a low-cost controller, and show that learning a token-agnostic skipping probability is equally effective.
3. By introducing an oracle controller that optimally assigns computation to sequences, we show that per-sequence adaptation has the potential to achieve a dramatic computation reduction.

2 Problem Setting

We consider the problem of auto-regressive language generation using a neural network, denoted by $T(\cdot) = d^L \circ \dots \circ d^1 \circ e(\cdot)$, and consisting of stacked decoder blocks d^l and an embedding layer e . For NLG, we augment T with a classification head σ . Consider an input sequence $\mathbf{s}_t = [\mathbf{x}_{1:p}, \mathbf{y}_{1:t}]$ consisting of the concatenation of a prompt $\mathbf{x}_{1:p}$ and a sequence of generated tokens $\mathbf{y}_{1:t}$ where $x_k, y_k \in \mathcal{Y}$, with \mathcal{Y} denoting the vocabulary space. Sequences are generated auto-regressively by sampling the next token \hat{y}_{t+1} from the predicted probability distribution $\hat{\mathbf{p}}_{t+1} = \sigma(T(\mathbf{s}_t))$. In the field of efficient dynamic inference, the early exit method dynamically selects an exit layer L_E , and executes layers $\{d^l\}_{l \leq L_E}$. Thus, the network can be written as $T_{EE}(\cdot) = d^{L_E} \circ \dots \circ d^1 \circ e(\cdot)$. For layer skipping, the execution of each layer d^l is controlled by a binary variable G^l , where $G^l = 1$ denotes executing layer l . The layer skipping model output is then defined as $T_{LS}(\cdot) = (G^L d^L + (1 - G^L)I) \circ \dots \circ (G^1 d^1 + (1 - G^1)I) \circ e(\cdot)$, where I is the identity function. For each case, we define the computational cost c as the number of decoder layers executed, omitting the fixed costs of the embedding layer $e(\cdot)$ and classification head $\sigma(\cdot)$. For early exiting, $c = L_E$, while $c = \sum_{l=1}^L G_l$ for layer skipping. For all experiments, we only consider the computation performed on the autoregressively generated tokens $\mathbf{y}_{1:T}$, as is commonly done [5, 6, 11, 12]. The prompt is propagated through the full network since the prompt tokens can be efficiently processed once and in parallel and provide important context for NLG [12]. In general, we aim to achieve the best ROUGE metric [13] for the lowest computational cost c .

3 Empirical Studies, Results, and Analysis

3.1 Is layer skipping more effective than early exiting?

We hypothesize that for NLG tasks and a pre-trained model without fine-tuning, layer skipping is more effective than early exit. Previous empirical results show that residual connections in decoder blocks contribute a greater portion of the total block’s output than the MLP and attention blocks [7]. Consequently, skipping a layer’s execution should result in limited hidden state drift. We assess “effectiveness” by two methods: (i) for next-token prediction error, we assess how the final-layer hidden state is preserved for a specific computational cost, where preservation is measured via cosine similarity denoted $\mathcal{S}(\mathbf{h}_t^L, \mathbf{h}_{t,c}^L)$; (ii) for KV cache propagation error, we measure the layerwise cosine similarity of all layers’ hidden states used for KV caching, and report the mean.

Experiment setup: Using OPT-1.3B [14] fine-tuned on Alpaca [15], and given a computational cost $c \in \{1, \dots, L\}$, we compare the early exiting $T_{EE,c}$ (where $L_E = c$) with two layer skipping approaches. The *uniform* layer skipping network $T_{ULS,c}$ evenly spaces skipped and executed layers [11]. The *random* layer skipping network $T_{RLS,c}$ executes c random layers, shuffling the executed layers every batch. Layer $l = 1$ always executes, since it plays a important role in transforming embeddings [7]. Further details and analysis are provided in Appendices A.2 and A.3.

Results: Figure 1a shows that the final layer hidden state obtained from uniform layer skipping $\mathbf{h}_{t,ULS,c}^L$ is closest to the original hidden state \mathbf{h}_t^L . Early exit has the lowest final hidden state similarity, but the highest overall cosine similarity of intermediate hidden states. These results reveal an interesting trade-off: Early exit concentrates error into the upper layers, leading to poorer next-token prediction, but improving average KV cache accuracy. Conversely, skipping layers more evenly distributes hidden state error, improving token prediction despite less accurate caches. At costs $c = \{12, 16\}$, uniform layer skipping is a clear winner, matching early exit for similarity of intermediate hidden states while achieving a much more similar final layer state.

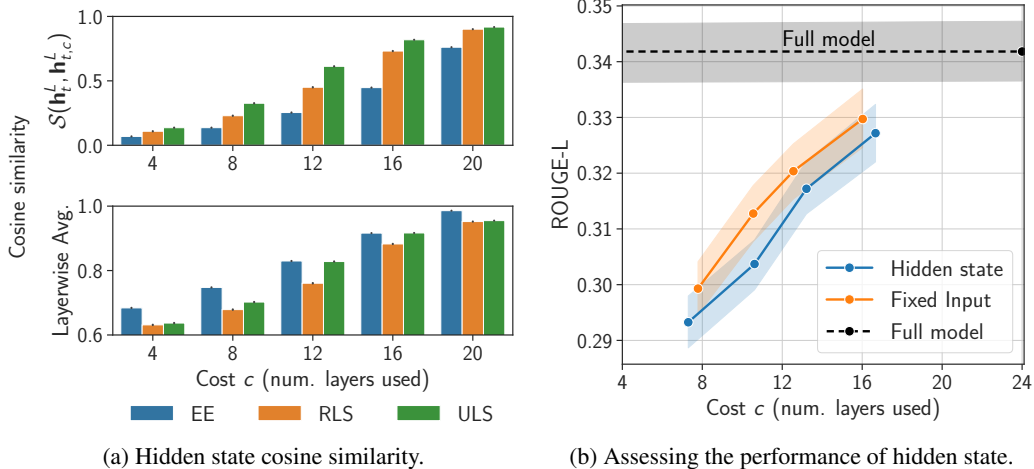


Figure 1: Using OPT-1.3B (24 layers) on the Alpaca dataset. We report the mean and 95% confidence interval on the test set. **(a)**: Average cosine similarity of the final hidden states \mathbf{h}_t^L (top) and layerwise intermediate hidden states \mathbf{h}_t^l , $1 \leq l \leq 23$ (bottom) obtained from early exit (EE), random and uniform layer skipping (RLS/ULS) compared to the full model execution. **(b)** Comparison of cost-performance curves when training skip controllers to use the hidden states \mathbf{h}^{l-1} or a fixed input. We see that, with statistical significance, using the hidden state is not helpful to the model’s performance.

3.2 Are layer skip controllers using hidden states more effective than a constant input?

We hypothesize that layer skipping controllers, limited in capacity because of their associated overhead costs, are ineffective at extracting meaningful context from token hidden states, since they are high dimensional embeddings designed to be processed by much larger decoder blocks. We assess “effectiveness” by comparing the performance to a sample-agnostic baseline, wherein the same controllers are provided with a constant input instead of the hidden state.

Experiment formulation: Following the approach in [9], we start by augmenting a fine-tuned model T with L trainable skip controllers to model the skip variable $G^l := g_{\phi^l}^l(\mathbf{h}_t^{l-1})$ at each layer with $\mathbf{h}_t^0 = e(\mathbf{s}_t)$. The skip-controllers are trained using the following loss formulation:

$$\begin{aligned} \mathcal{L} &= \mathbb{E}_{\hat{Y}_t, Y_t} [\mathbb{E}_{\mathbf{G}|Y} [\mathbb{1}(\hat{Y}_t \neq Y_t) + \alpha C(\hat{Y}_t)]] = \mathbb{E}_{\hat{Y}_t, Y} [\mathbb{E}_{\mathbf{G}|Y} [\mathbb{1}(\hat{Y}_t \neq Y_t) + \alpha \sum_{l=1}^L P(G^l | \mathbf{s}_{t-1}; \phi)]] \\ &\approx \frac{1}{N} \sum_{i=1}^N \left(KL(\hat{\mathbf{p}}_{i,t} | \mathbf{p}_{i,t}) + \alpha \sum_{l=1}^L g_{\phi^l}^l(\mathbf{h}_{i,t}^{l-1}) \right) \end{aligned} \quad (1)$$

The auxiliary term $C(\hat{Y}_t)$, weighted by α , introduces an inference cost penalty proportional to the number of executed layers, providing inference cost regularization. To make the loss differentiable, we use the reparametrization trick introduced in Jang et al. [16], which uses the discrete realization $g^l \in \{0, 1\}$ of G^l in the forward path, but uses the continuous $P(G^l | \mathbf{s}_t; \phi)$ during backpropagation.

Experiment setup: We model the skip controllers as a single linear layer following [5, 6, 9], with input size $|\mathbf{h}|$ and output size 2. A Gumbel-Softmax module [16] from the Torch library [17] is attached at the output to allow for differentiable layer skipping [9]. We train the skip controllers jointly with the backbone (we repeat the same experiment with a frozen backbone in Appendix A.4.3). As a comparison, we train the same network by providing a fixed input tensor of size $|\mathbf{h}|$ and value 1 as the controller input, $g_{\phi^l}^l(\mathbf{1})$. The controller’s decision is then no longer sample-aware, but can still learn to optimize the skipping frequency of its associated layer. The output remains stochastic because the Gumbel-Softmax module samples the skip decision from a combination of the linear layer output and a Gumbel(0,1) distribution [16]. We repeat the procedure varying the hyperparameter $\alpha \in \{2, 4, 6, 10\}$ to obtain different operating points in terms of computational cost. All models are trained using OPT-1.3B on Alpaca. The detailed setup can be found in Appendices A.2 and A.4.1.

Results: Figure 1b shows the cost-performance trade-off obtained with and without the hidden states. Since the hidden state does not bring a statistically significant advantage, we conclude that, if we are constrained to use simple skip controllers with limited capacity due to computational restrictions, it is equally effective to learn a token-agnostic average skip rate for each layer. Appendix A.4.2 compares the average skip ratios of the hidden state and fixed input approaches on a per-layer basis for different α values, showing they follow a similar distribution.

3.3 Is it possible for sequence-aware layer skipping to outperform a static method?

Considering the outperformance of layer skipping (Sec. 3.1), and the difficulty of using the hidden state for per-token decisions (Sec. 3.2), we assess whether adaptive layer skipping can be performed at the sequence-level. This is further motivated by the following observations: 1) The primary source of latency when auto-regressively generating tokens using an LLM is the repeated transfer of layer weights in and out of GPU memory [18, 19]. Using a constant subset of layers for an entire sequence generation can thus significantly speed-up inference by reducing the number of IO operations. 2) The controllers need only be evaluated once for an entire generation as opposed to repeatedly for every token. We hypothesize that some prompts are easier to respond to than others and thus need less computation. Therefore, determining the computational path for the whole sequence based on the prompt could provide an efficiency improvement over a static scheme.

Experiment setup: We conduct an oracle experiment to demonstrate the potential of sequence-level methods. The oracle selects the optimal interleaving computational paths for generating multiple tokens. The oracle is given the full OPT-1.3B 24 layer model T , as well as 3 models $T_{\text{ULS},12}$, $T_{\text{ULS},8}$, and $T_{\text{ULS},4}$ that implement uniform layer skipping with varying computational budgets. From each model, given a validation set of n prompts \mathcal{X} , we obtain 4 prediction sets $\mathcal{Y} = \sigma(T(\mathcal{X}))$, $\mathcal{Y}_{12} = \sigma(T_{\text{ULS},12}(\mathcal{X}))$, $\mathcal{Y}_8 = \sigma(T_{\text{ULS},8}(\mathcal{X}))$, $\mathcal{Y}_4 = \sigma(T_{\text{ULS},4}(\mathcal{X}))$ from the Alpaca dataset. Each prediction set item consists of the full sequence \mathbf{y}_i generated by the compressed model for a prompt $\mathbf{x}_i \in \mathcal{X}$. The oracle assigns a cost $c \in \{4, 8, 12, 24\}$ for each sequence to optimize the ROUGE-L score given a budget β on average cost. See Appendix A.5 for details and discussion of the oracle’s sampling advantage.

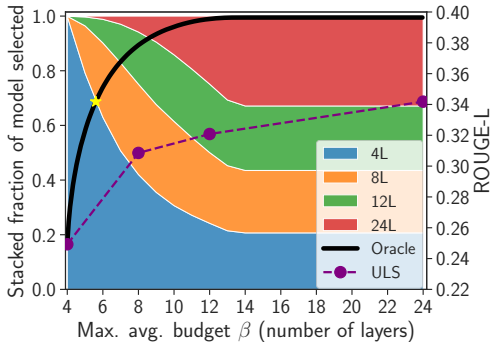


Figure 2: ROUGE-L (right axis) of T and $T_{\text{ULS},x}$ compared to an oracle skip controller that selects the optimal model per sequence given a global budget. For each budget, the selection percentage of each model by the oracle is shown as a stacked plot (left axis). The yellow star denotes where the oracle can match the performance of the full model, using an average of 5.6/24 layers (23.3%).

Results: Figure 2 illustrates the potential advantage of sequence-level computation allocation. Despite $T_{\text{ULS},4}$ achieving an average ROUGE-L score of 0.249 compared to 0.342 for the full model, the oracle can match the full model’s performance using as few as 5.6 layers, selecting $T_{\text{ULS},4}$ 69.6% of the time. The steep performance increase of the oracle illustrates how highly compressed models can be suitable for a majority of sequences, with the larger models rarely required. Even when given a budget that permits usage of all 24 layers for every sequence, the oracle still finds that the compressed models are preferred for more than 68% of sequences.

4 Conclusion and Limitations

Our experimental results offer new guidance for the development of dynamic LLMs. We provide new evidence that layer skipping is a better alternative than early exiting for autoregressive transformer tasks. We highlighted the need for further research in skip controller design, as we found that the hidden states information is too complex for a simple controller. Finally, we provided a motivating experiment for a new approach: controlling layer skipping at the sequence level. The key limitation of our work is the limited scope of architectures and datasets, imposed by computational constraints. We made an effort to select models and tasks that are generally representative, but our findings may not generalize to other architectures and tasks. Additional limitations are discussed in Appendix A.6.

Acknowledgments

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

Cette recherche a été financée par le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG).

References

- [1] C.-J. Wu, R. Raghavendra, U. Gupta, B. Acun, N. Ardalani, K. Maeng, G. Chang, F. Aga, J. Huang, C. Bai, M. Gschwind, A. Gupta, M. Ott, A. Melnikov, S. Candido, D. Brooks, G. Chauhan, B. Lee, H.-H. Lee, B. Akyildiz, M. Balandat, J. Spisak, R. Jain, M. Rabbat, and K. Hazelwood, “Sustainable AI: Environmental implications, challenges and opportunities,” in *Proc. of Mach. Learn. and Syst.*, vol. 4, 2022, pp. 795–813.
- [2] S. Luccioni, Y. Jernite, and E. Strubell, “Power hungry processing: Watts driving the cost of AI deployment?” in *Proc. Conf. on Fairness, Accountability, and Transparency (ACM)*. ACM, Jun. 2024.
- [3] B. Rokh, A. Azarpeyvand, and A. Khanteymooori, “A comprehensive survey on model quantization for deep neural networks in image classification,” *ACM Trans. Intell. Syst. Technol.*, vol. 14, no. 6, nov 2023.
- [4] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, “Dynamic neural networks: A survey,” *IEEE Trans. on Pattern Analysis; Mach. Intell.*, vol. 44, no. 11, pp. 7436–7456, 2022.
- [5] M. Elbayad, J. Gu, E. Grave, and M. Auli, “Depth-adaptive transformer,” in *Proc. Int. Conf. Learn. Representations (ICLR)*, 2020.
- [6] T. Schuster, A. Fisch, J. Gupta, M. Dehghani, D. Bahri, V. Tran, Y. Tay, and D. Metzler, “Confident adaptive language modeling,” in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 35, 2022, pp. 17 456–17 472.
- [7] Z. Liu, J. Wang, T. Dao, T. Zhou, B. Yuan, Z. Song, A. Shrivastava, C. Zhang, Y. Tian, C. Ré, and B. Chen, “Deja vu: Contextual sparsity for efficient llms at inference time,” in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2023.
- [8] A. Fan, E. Grave, and A. Joulin, “Reducing transformer depth on demand with structured dropout,” in *Proc. Int. Conf. Learn. Representations (ICLR)*, 2020.
- [9] D. Zeng, N. Du, T. Wang, Y. Xu, T. Lei, Z. Chen, and C. Cui, “Learning to skip for language modeling,” 2023, arXiv preprint: arXiv 2311.15436.
- [10] Z. Zeng, Y. Hong, H. Dai, H. Zhuang, and C. Chen, “ConsistentEE: A consistent and hardness-guided early exiting method for accelerating language models inference,” in *Proc. AAAI Conf. Artif. Intell.*, Mar. 2024, pp. 19 506–19 514.
- [11] Y. Liu, F. Meng, and J. Zhou, “Accelerating inference in large language models with a unified layer skipping strategy,” 2024, arXiv preprint: arXiv 2404.06954.
- [12] L. D. Corro, A. D. Giorno, S. Agarwal, B. Yu, A. Awadallah, and S. Mukherjee, “Skipdecode: Autoregressive skip decoding with batching and caching for efficient llm inference,” 2023, arXiv preprint: arXiv 2307.02628.
- [13] C.-Y. Lin, “ROUGE: A package for automatic evaluation of summaries,” in *Proc. Assoc. for Comput. Linguistics (ACL)*, 2004, pp. 74–81.
- [14] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, and L. Zettlemoyer, “OPT: Open pre-trained transformer language models,” 2022, arXiv preprint: arXiv 2205.01068.

- [15] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto. (2023) Stanford Alpaca: An instruction-following llama model. [Online]. Available: https://github.com/tatsu-lab/stanford_alpaca
- [16] E. Jang, S. Gu, and B. Poole, “Categorical reparameterization with gumbel-softmax,” in *Proc. Int. Conf. on Learn. Representations (ICLR)*, 2017.
- [17] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, “Automatic differentiation in pytorch,” in *Proc. Adv. Neural Info. Process. Syst. (NeurIPS)*, 2017.
- [18] H. Xia, Z. Yang, Q. Dong, P. Wang, Y. Li, T. Ge, T. Liu, W. Li, and Z. Sui, “Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding,” 2024, arXiv preprint: arXiv 2401.07851.
- [19] Y. Leviathan, M. Kalman, and Y. Matias, “Fast inference from transformers via speculative decoding,” in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2023.
- [20] J. Xin, R. Tang, J. Lee, Y. Yu, and J. Lin, “DeeBERT: Dynamic early exiting for accelerating BERT inference,” in *Proc. Assoc. for Comput. Linguistics (ACL)*, 2020, pp. 2246–2251.
- [21] W. Liu, P. Zhou, Z. Wang, Z. Zhao, H. Deng, and Q. Ju, “FastBERT: A self-distilling BERT with adaptive inference time,” in *Proc. Assoc. for Comput. Linguistics (ACL)*, 2020, pp. 6035–6044.
- [22] S. Bae, J. Ko, H. Song, and S.-Y. Yun, “Fast and robust early-exiting framework for autoregressive language models with synchronized parallel decoding,” in *Proc. Conf. on Empirical Methods in Natural Lang. Process. (EMNLP)*, 2023, pp. 5910–5924.
- [23] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. V. Le, G. E. Hinton, and J. Dean, “Outrageously large neural networks: The sparsely-gated Mixture-of-Experts layer,” in *Proc. Int. Conf. on Learn. Representations (ICLR)*, 2017.
- [24] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. de las Casas, E. B. Hanna, F. Bressand, G. Lengyel, G. Bour, G. Lample, L. R. Lavaud, L. Saulnier, M.-A. Lachaux, P. Stock, S. Subramanian, S. Yang, S. Antoniak, T. L. Scao, T. Gervet, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, “Mixtral of experts,” 2024, arXiv preprint: arXiv 2401.04088.
- [25] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang *et al.*, “A survey on evaluation of large language models,” *ACM Trans. Intell. Syst. Technol.*, vol. 15, no. 3, pp. 1–45, 2024.
- [26] R. Nallapati, B. Zhou, C. dos Santos, Çağlar Gulçehre, and B. Xiang, “Abstractive text summarization using sequence-to-sequence RNNs and beyond,” in *Proc. Conf. on Comput. Natural Lang. Learn.*, 2016, pp. 280–290.
- [27] S. Iyer, X. V. Lin, R. Pasunuru, T. Mihaylov, D. Simig, P. Yu, K. Shuster, T. Wang, Q. Liu, P. S. Koura, X. Li, B. O’Horo, G. Pereyra, J. Wang, C. Dewan, A. Celikyilmaz, L. Zettlemoyer, and V. Stoyanov, “OPT-IML: Scaling language model instruction meta learning through the lens of generalization,” 2023, arXiv preprint: arXiv 2212.12017.
- [28] S. Martello and P. Toth, *Knapsack problems: algorithms and computer implementations*. USA: John Wiley & Sons, Inc., 1990.

A Appendix

A.1 Related Work

A.1.1 Sample-agnostic approaches

We deem approaches *sample-agnostic* if they use a fixed schedule to bypass certain model layers, independent of the samples being processed. In this category, Fan et al. [8] and Liu et al. [11] train encoder-decoder and decoder-only models, respectively, to use a constant skipping schedule for all sequences. Both observe that uniformly spacing the skipped layers achieves superior downstream performance over skipping consecutive layers.

A.1.2 Sample-aware approaches

Sample-aware approaches use information about the samples provided to determine the computational budget to allocate. Elbayad et al. [5] propose an early exit strategy for encoder-decoder models, which consists of executing only the first n layers and bypassing the rest of the model using an early exit classifier, based on the sample being computed. They show that learning a binary controller at each layer using the current hidden state achieves the best performance. Schuster et al. [6] report similar findings using the hidden states on a summarization task, performing on par with a softmax thresholding approach, described in Appendix A.1.3. For decoder-only models, Zeng et al. [10] learn a policy network for early exiting based on the intermediate hidden states. In contrast, Zeng et al. [9] learn a layer skipping controller which uses the hidden states to choose to skip a given layer. While the hidden state-based methods do outperform the fully random or static baselines presented, they do not make comparison to a learnable, token-agnostic method similar to the baseline in Section 3.2. This leaves some ambiguity regarding how much of the performance improvement is due to leveraging the context of a specific token compared to just learning the average importance of each layer.

A.1.3 Routing with intermediate classifiers

In early exit, a common method to determine the exit layer is to perform inference on intermediate classifiers after every layer, and to define a threshold on the classifier output distribution to trigger the early exit [20, 21]. Although successful in other tasks, such an approach is especially costly in an NLG setting where the large output vocabulary size requires a large linear classifier. Evaluating this classifier every layer adds unwanted overhead costs, which motivated research towards lightweight binary classifiers based on the hidden state [5, 6].

A.1.4 KV cache management

During autoregressive inference, the self-attention mechanism in each layer requires the key-value pairs of previous tokens, which are usually stored in a KV cache to avoid redundant computation. However, the key-value pairs are never computed when a layer is bypassed through early exiting or layer skipping. Elbayad et al. [5] identify this concern and propose to copy the hidden state of the closest previously executed layer as an approximation for the missing hidden state, and fill the missing KV cache entries using the key and value projections matrices of each layer. The same copying principle is used by Zeng et al. [9] for sample-aware layer skipping. Bae et al. [22] propose to instead compute the exact KV cache entries when they become required, taking advantage of parallel computation with the current token. With this method, early exiting does not reduce the total computational cost of the model, but does reduce the inference latency by deferring and parallelizing the execution of skipped layers.

A.1.5 Successes of sample-aware routing

Intermediate hidden states in LLMs are high dimensional, complex embeddings which encode contextual information for Transformer blocks to process, so re-purposing them for tasks such as early exit prediction is a challenging undertaking in efficient dynamic inference. Some success has been observed in encoder-decoder models by Elbayad et al. and Schuster et al., but this architecture possesses a distinct advantage in the context of early exiting: Since the initial input is first encoded by the encoder, the early decoder layers have access to a complete encoding of the input, which is not

available in a decoder-only architecture [12]. In the related field of Mixture of Experts (MoE), a linear gating network is used to route samples to a subset of experts using the hidden state with considerable success [23, 24]. In this setting, the goal of the gating network is to balance the use of each experts while allowing them to specialize on a subset of the hidden state space. We conjecture that a linear gating network is sufficient to process the hidden states because it needs only to consistently and fairly divide inputs among experts, which are jointly trained to specialize on the subset of inputs assigned to them. In contrast, a binary controller in efficient dynamic inference must learn to identify inputs which do not require full computation for the next token to be successfully predicted. This involves predicting the output of decoder layers with several orders of magnitude more parameters than itself, using a hidden state which has not been trained to explicitly encode such information.

A.2 Datasets and model details

A.2.1 Datasets

Alpaca [15] All results in the main body of the paper are reported for the Alpaca dataset for the following reasons. First, it is a common NLG benchmark for instruction-following and long-form question answering [25]. Furthermore, the dataset has short prompts and long generation requirements, implying that the autoregressive generation phase dominates over the prompt decoding, which is the main target use case for layer skipping and early exiting. The dataset consists of 52,000 instructions and demonstrations used for instruction-tuning. The median prompt length is 43 tokens, while the median label length is 42 tokens. The maximum context length used throughout training is 512, and the split used throughout all experiments is 70% train, 15% validation, and 15% test.

CNN-DM [26] Plots resulting from the use of the CNN-Daily Mail dataset are included in the appendix to further validate our findings on another popular NLG benchmark. The dataset consists of just over 300,000 unique news articles and associated summaries. The median prompt length is 788 tokens, and the median label length is 66 tokens. The maximum context length used for training is 2048, and the split used for this dataset is approximately 92% train, 4% validation, and 4% test.

A.2.2 Fine-tuned model training

We use the OPT-decoder-only architecture [14], and perform standard fine-tuning on Alpaca and CNN-DM starting from the instruction-tuned checkpoint of OPT-1.3B by Iyer et al. [27]. All experiments are performed using 8X Nvidia P100 as compute. For Alpaca, we fine-tune for 6 epochs with early stopping and a patience of 1. The best evaluation ROUGE-L score is obtained after 3 epochs. We use a learning rate of $5e^{-5}$ which decays linearly over the training procedure. We use a batch size of 64, and an AdamW optimizer.

For CNN-DM, we fine-tune for 2 epochs with early stopping and a patience of 1. The best evaluation is obtained at 2 epochs. The same learning rate and optimizer is used with a batch size of 48. In both cases, the prompt is masked away from the loss, so the model is only trained on the completion task.

A.3 Hidden state similarity details

A.3.1 Uniform layer skipping formal definition

We define the uniform layer skipping network with cost c out of a total of L layers as $T_{ULS, c}$. The execution of layer l is determined by $G^l := U(l, c)$ using the recursion

$$U(l, c) = \mathbb{1} \left(\left(\sum_{k < l} U(k, c) \right) \leq (l - 1) \frac{c}{L} \right), U(1, c) := 1. \quad (2)$$

Figure 3 illustrates the produced layer skipping pattern for different computational costs c .

A.3.2 Random layer skipping when executing the first layer is not enforced

To justify the rule to enforce the execution of layer $l = 1$ for all methods in Section 3.1, we include in Figure 4 the results of $T_{RLSwo/1}$, random layer skipping when layer 1 is permitted to be skipped. We observe that the performance is consistently worse than T_{RLS} , confirming the importance of executing the first layer in a frozen model.

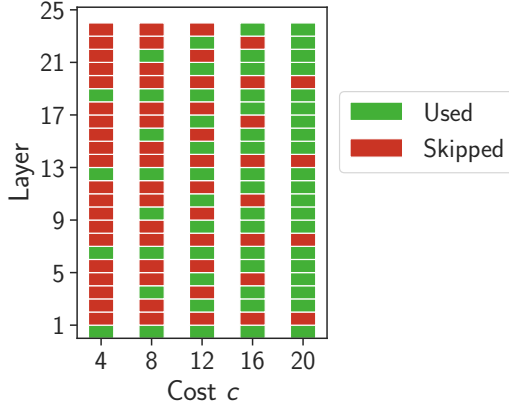


Figure 3: Uniform layer skipping strategy of a 24 layer model following Equation 2 for different computational cost c .

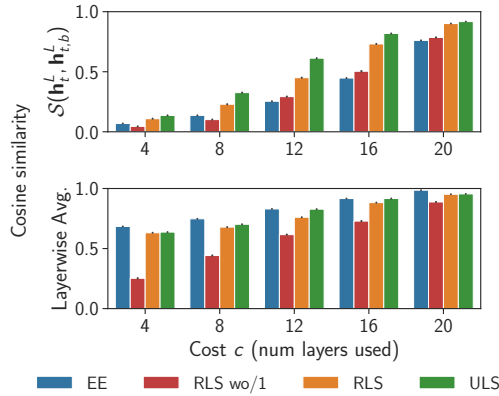


Figure 4: Using OPT-1.3B (24 layers) on the Alpaca dataset. We report the mean and 95% confidence interval on the test set. Average cosine similarity of the final hidden states \mathbf{h}_t^L (top) and layerwise intermediate hidden states $\mathbf{h}_t^l, 1 \leq l \leq 23$ (bottom) obtained from each dynamic route strategy compared to the full model execution. We include RLS wo/1, random layer skipping without enforcing the execution of layer 1, and see that it performs consistently worse than RLS.

A.3.3 CNN-DM results for hidden state similarity

We present in Figure 5 a repetition of the Section 3.1 experiment on the CNN-DM dataset. We observe a similar pattern, with the exception of early exiting having a higher cosine similarity of intermediate hidden states than uniform layer skipping at cost $c = 12$.

A.4 Hidden state versus fixed input layer skipping details

A.4.1 Training setup

We start with a standard fine-tuned model following Appendix A.2. After the fine-tuning is complete, the skip controller training phase begins and follows the procedure of Zeng et al. [9]. Controllers – consisting of a single linear layer followed by a Gumbel softmax [16] – are added to every layer of the fine-tuned model except the first and last, and are jointly fine-tuned with the backbone. The KV-cache is propagated following the approach described in the CALM paper [6] and the prompt is propagated through the full network. We train the controller-augmented network for a maximum of 3 epochs, running evaluation every $\frac{1}{3}$ epoch and using early stopping with a patience of 1. The fixed input Gumbel training uses the same parameters as the hidden state Gumbel training.

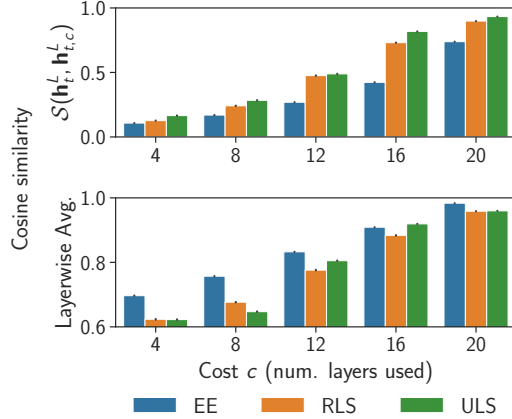


Figure 5: Using OPT-1.3B (24 layers) on the CNN-DM dataset. We report the mean and 95% confidence interval on the test set. The cosine similarity of the final hidden states \mathbf{h}_t^L (top) and layerwise intermediate hidden states \mathbf{h}_t^l , $1 \leq l \leq 23$ (bottom) obtained from early exit (EE), random and uniform layer skipping (RLS/ULS) is shown compared to the full model execution.

A.4.2 Per-layer skip ratios of the hidden state and fixed input skipping approaches

In this section, we provide additional information about the experiment of Section 3.2. This experiment showed that there was no significant performance difference when using the hidden state compared to a fixed input to the skip controllers. In order to further investigate the skipping strategies adopted by each of the approaches, we plot the percentage of skips across all generated tokens for varying values of α . The results are displayed in Figure 6.

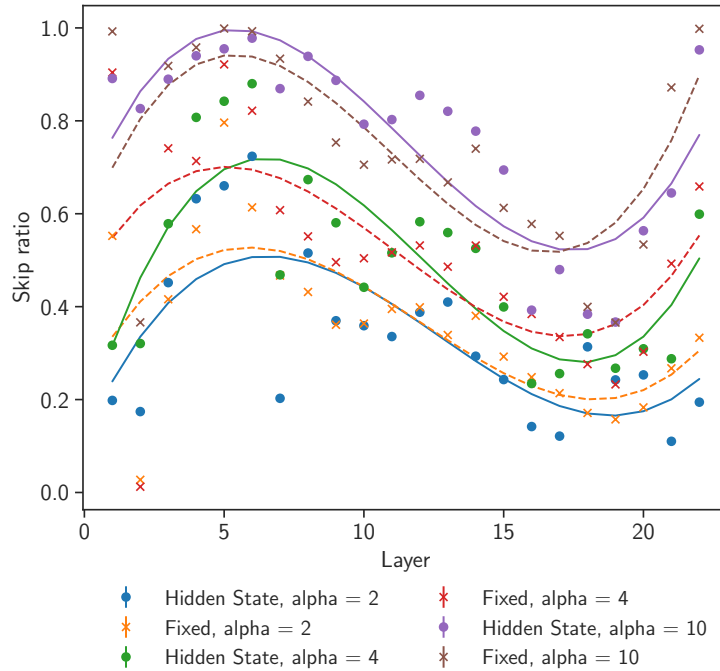


Figure 6: Per-layer skip ratios as a function of α and the layer number when using the hidden state or a fixed tensor as input to the skip controllers. The two approaches have very similar trends across different values of α , showing that the learnt skip strategy depends more on the layer than the token.

The results show very similar trends for both the hidden state and fixed input approaches across different values of α , showing again that the two models learn a similar skipping strategy on average per layer.

A.4.3 Ablation study of controller performance on token hidden states

The surprising results of Section 3.2 prompted us to further investigate why skip controllers using a token’s hidden state \mathbf{h}^l perform on-par with controllers fed a fixed input when using the loss formulation in Equation (1). One plausible explanation is that training the backbone jointly with the controllers as in Zeng et al. [9] causes a distribution shift in \mathbf{h}^l making the controller learning task more challenging.

To test this hypothesis we first fine-tune the backbone with random layer dropout following Fan et al. [8], freeze the backbone and subsequently train the controllers following Equation (1). As in Section 3.2, we compare the cost-performance trade-off between hidden-state and fixed input training for varying values of α . We perform this experiment on OPT-350M [14] trained on the Alpaca dataset. OPT-350M consists of 24 layers. As before, we always execute the first and last layers. We use a batch size of 20 and a maximum sequence length of 512. Figure 7 shows that even in this setting where the backbone is frozen, the controllers fail to extract meaningful information from \mathbf{h}^l with the exception of the high cost regime (around 16 layers).

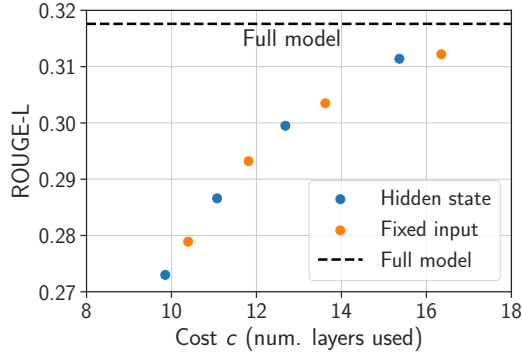


Figure 7: Comparison of cost-performance curves on OPT-350M [14] when training skip controllers to use the hidden states \mathbf{h}^l or a fixed input. For this ablation, we fix the fine-tuned backbone to prevent distribution shifts in \mathbf{h}^l . We see that using the hidden state brings no significant advantage over the fixed input.

A.5 Oracle algorithm details

In this section, we provide additional information about the experiment of Section 3.3. Given the 4 prediction sets $\mathcal{Y} = \sigma(T(\mathcal{X}))$, $\mathcal{Y}_{12} = \sigma(T_{12}(\mathcal{X}))$, $\mathcal{Y}_8 = \sigma(T_8(\mathcal{X}))$, $\mathcal{Y}_4 = \sigma(T_4(\mathcal{X}))$ obtained by using the 24 (full), 12, 8 and 4-layer cost uniform layer skipping OPT-1.3B respectively, the oracle algorithm assigns to each input \mathbf{x}_i a model T_c so as to maximize the average performance metric (ROUGE-L score) given a maximum average computational budget β .

The problem can then be framed as a multiple choice knapsack problem [28] where each sequence \mathbf{x}_i has to be assigned to exactly one of four models T_c , $c \in \{4, 8, 12, 24\}$ to maximize average ROUGE-L score subject to a cost constraint. The solution to the algorithm for budget β is an assignment of costs $\mathbf{S}_{1:n}^\beta \in \{4, 8, 12, 24\}^n$, where S_i^β denotes the cost c of the model T_c selected for input \mathbf{x}_i . We can write the oracle objective P as maximizing the average ROUGE-L score obtained given β :

$$\begin{aligned}
 P = \arg \max_{\mathbf{S}_{1:n}^\beta} & \frac{1}{n} \sum_{i=1}^n \text{ROUGE}(T_{S_i^\beta}(\mathbf{x}_i)) \\
 \text{subject to} & \frac{1}{n} \sum_{i=1}^n S_i^\beta \leq \beta
 \end{aligned} \tag{3}$$

where $n = 7800$ is the total number of samples considered. Using dynamic programming, we obtain the optimal assignment $\mathbf{S}_{1:n}^{*\beta}$ and report the average ROUGE-L score P for different β in Section 3.3.

A.5.1 Quantifying the sampling advantage

Although $\mathbf{S}_{1:n}^{*\beta}$ is a valid solution achievable in theory by a perfect controller module, its advantage over the ULS models T_4, T_8, T_{12} and T cannot be solely attributed to the oracle assigning different computational paths based on a sequence’s difficulty, as was hypothesized in Section 3.3. Because of the perfect information available to the oracle, there is an intrinsic sampling advantage present due to the natural variance in the ROUGE-L scores of each model. Even though the output quality of each model is correlated to its cost on average, stochasticity from the model’s training and inference mean that, for all input \mathbf{x}_i , we cannot assume that

$$ROUGE(T_4(\mathbf{x}_i)) \leq ROUGE(T_8(\mathbf{x}_i)) \leq ROUGE(T_{12}(\mathbf{x}_i)) \leq ROUGE(T(\mathbf{x}_i)).$$

Generally, the advantage obtained in expectation from taking the maximum of multiple samples $\mathbb{E}[\max(X_1, X_2, \dots, X_k)] - \mathbb{E}[\frac{1}{k} \sum_{i=1}^k X_i]$ depends on the nature of the distributions, their variance and the number of samples considered. We refer to this advantage as the sampling advantage. As part of maximizing the objective P in Equation 3, the oracle will always choose the maximally performing model whose cost is within the budget available for a given input. The variance in the ROUGE-L scores of each model therefore introduces a sampling advantage, which in our case depends on the arbitrary number of models considered for the experiment.

Although it is not feasible to construct an oracle benefiting only from the dynamic budget allocation advantage, we instead propose a greedy oracle which only benefits from sampling advantage to serve as a comparative baseline to the original oracle, in an effort to better approximate the dynamic budget allocation advantage. As suggested by its name, the greedy oracle is given perfect information regarding the performance of each model on each sequence, but finds an assignment $\mathbf{S}_{1:n}^\beta$ by selecting the locally optimal S_i^β for each input, according to the objective

$$P = \frac{1}{n} \sum_{i=1}^n \arg \max_{S_i^\beta} ROUGE(T_{S_i^\beta}(\mathbf{x}_i)) \quad (4)$$

subject to $S_i^\beta \leq \beta \quad \forall i.$

Given an average computational budget β , the greedy oracle equally distributes the budget to all sequences. Unlike the original oracle which can reduce the budget given to one sequence to increase it for another who needs it more, the greedy oracle makes an isolated decision for every input, and therefore has no dynamic budget allocation advantage. However, by selecting the best model within the budget for each input, it is given a sampling advantage. While this sampling advantage may not perfectly align with that of our initial oracle, it serves as a reasonably close approximation.

We evaluate the greedy oracle at $\beta = 4, 8, 12, 24$, which corresponds to selecting the best model from $\{T_4\}$, $\{T_4, T_8\}$, $\{T_4, T_8, T_{12}\}$ and $\{T_4, T_8, T_{12}, T\}$ respectively for each input. The performance difference shown in Figure 8 between the ULS and greedy oracle plots corresponds to the sampling advantage obtained by selecting the best of the models with layer cost $c \leq \beta$ for each sequence, instead of only choosing the most costly one. As expected, there is no advantage at $\beta = 4$, since only the T_4 model can be used. The advantage grows with β as more models can be considered. We can then compare the original oracle performance to the greedy oracle for the approximate marginal advantage obtained by dynamic budget allocation. We see that at $\beta = 24$, there is no marginal advantage since dynamically allocating the budget has no effect when there already is the full $c = 24$ available to every sequence. Crucially, we observe that there remains a notable marginal advantage in the budget-constrained interval $4 < \beta < 12$. For example, at $\beta = 8$, the dynamic budget allocation advantage of the original oracle over the greedy oracle is greater than the sampling advantage of the greedy oracle over the individual ULS model. Furthermore, the dynamic budget allocation alone is sufficient for the oracle to outperform the full model using only $\beta = 8$, which validates the hypothesis that there is potential to achieve impressive computational savings from sequence-level budget allocation.

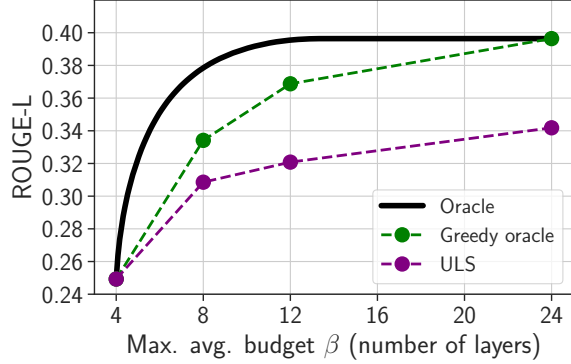


Figure 8: ROUGE-L of ULS models T_4, T_8, T_{12} and T compared to an oracle skip controller which selects the optimal model per sequence given an maximum average budget β in number of layers. The greedy oracle cannot trade off increased budget for some sequence at the expense of others, but selects the model within the budget with the best performance.

A.5.2 Accounting for the generation length in the computational cost

We note that the second line of Equation (3), which measures the total computational cost over the entire dataset, does not account for the length the label for \mathbf{x}_i and only depends on the number of layers of the network selected for \mathbf{x}_i . Ignoring the label length allows to significantly speed up the knapsack solver by reducing the search space. Nonetheless, we verify that the model selection is independent of the label length by performing a chi-squared test of homogeneity at two different average budgets. Specifically, we bin the label lengths into 11 bins $\mathcal{B}_{i \in \{1, \dots, 11\}}$ of width 15. That is, \mathcal{B}_i corresponds to generations whose length is in $[15(i-1) + 1, 15i]$ for $i \in \{1, \dots, 10\}$. The 11th bin contains all sequences whose label length is above 150. For a given average budget β , we obtain the proportion of samples routed on each compressed network for each bin \mathcal{B}_i denoted by $p_{\beta, 24, \mathcal{B}_i}, p_{\beta, 12, \mathcal{B}_i}, p_{\beta, 8, \mathcal{B}_i}, p_{\beta, 4, \mathcal{B}_i}$. We then conduct a chi-square homogeneity test to determine whether these proportions significantly vary from one bin to another. For $\beta = 6$, corresponding to the yellow star in Figure 2, we obtain $\chi^2 = 29.02$ and a p-value of 0.51. We conclude that there isn't sufficient evidence to reject the null-hypothesis that the network selection across bins follows the same distribution using a significance level $\alpha = 0.05$. For $\beta = 24$, we obtain $\chi^2 = 40.57$ a p-value of 0.09. Similarly, we conclude that there isn't enough evidence to reject the null hypothesis at $\alpha = 0.05$. Figure 9 provides a visualization of the distribution of network selections across bins for these two average budgets.

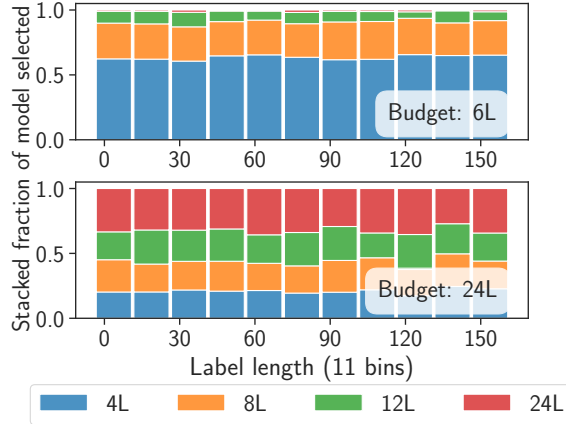


Figure 9: OPT-1.3B, Alpaca: Network selection for binned label lengths for average budgets β of 6 (top) and 24 (bottom). We see that there is no significant variation of network selection distribution across bins.

A.5.3 CNN-DM results for the sequence-level oracle

We present in Figure 10 the oracle results for OPT-1.3B trained on CNN-DM following the experimental setup in Section 3.3. The test set contains $n = 11490$ prompts. We obtain a similar performance curve and model selection percentage to Alpaca, with the exception that the full model T is used with a higher percentage relative to T_4 for large values of β .

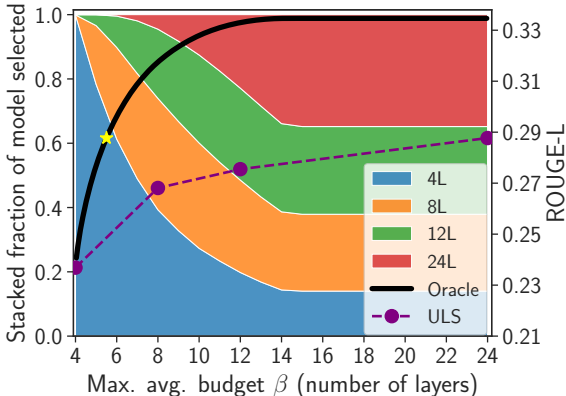


Figure 10: CNN-DM dataset: ROUGE-L (right axis) of T , $T_{\text{ULS}, 12}$, $T_{\text{ULS}, 8}$, and $T_{\text{ULS}, 4}$ compared to an oracle skip controller that selects the optimal model per sequence given a global budget. For each global budget, the selection percentage of each model by the oracle is shown as a stacked plot (left axis). The yellow star denotes where the oracle can match the performance of the full static model, using an average of 5.5/24 layers (22.9%).

A.6 Limitations

There are several limitations that should be considered when interpreting the results. First, due to computational constraints, the scope of architectures and datasets we used for our experiments is limited. We made an effort to select models and tasks that are generally representative, but our findings may not be broadly generalizable to different architectures and tasks. Second, the performance of our gate controller is likely influenced by its design, and we only presented one design architecture. Therefore, it is possible that a better gate controller design could potentially utilize the hidden state more effectively. However, we do stress that our design aligns with architectures used in previous work, and that the computational budget dictates that a simple controller must be employed. Last, as the oracle used in our analysis in Figure 2 directly uses test ROUGE scores for assignment, it can select the best outcome from multiple samples, even if they originate from the same distribution. As a result, the oracle achieves far beyond the “attainable performance” and surpasses a Bayes optimal model. The goal of the experiment is to highlight that there is a sequence-level selection that achieves a dramatic performance benefit. This motivates the development of a technique that can make a decision based on the prompt, without the benefit of hindsight. While we cannot provide the exact Bayes optimal performance, in Appendix A.5.1 we approximated the effect of the sampling advantage and showed that there remains a notable improvement .