ATTENTION IS ALL YOU NEED 2: SERIOUSLY, ALL WE ARE USING IS ATTENTION

Ohad Rubin
The Blavatnik School of Computer Science
Tel Aviv University
{ohad.rubin}@cs.tau.ac.il

ABSTRACT

Existing methods for conditional computation in language models either focus on conditional context retrieval or dynamic application of feed-forward layers (e.g., Mixture of Experts), but not both simultaneously. We propose a simple modification of attention that allows combining these approaches, allowing a flexible computational budget for each token. This allows for a dynamic compute allocation of either activated parameters, context or even bypassing computation entirely. Our unified layer introduces a budget prediction network that shares capacity between attention and feed-forward layers across all tokens in the input, which in turn allows leveraging the learning signals from both conditional context retrieval and feed-forward routing, resulting in improved overall model quality. We demonstrate that our approach not only improves perplexity and reduces computational cost compared to baseline models using either conditional context or MoE layers separately, but also exhibits enhanced generalizability to out-of-distribution context lengths and varying computational budgets. We validate these improvements on standard long-context language modeling datasets, including books, scientific papers and code.

1 INTRODUCTION

Large Language Models (LLMs) excel in diverse NLP tasks but suffer from inefficiencies due to a fixed computational budget per token, regardless of task complexity, leading to high costs and limited optimization potential.

Recent research challenges the fixed computation paradigm by exploring adaptive mechanisms like speculative decoding, which uses smaller models for token generation and larger ones for verification. Mixture of Experts (MoE) models activate specific parameters to reduce computation while maintaining performance. Dynamic models adjust computation based on input complexity, and layer bypassing reduces costs without losing accuracy. These methods recognize that different tokens and tasks require varying computational resources.

Additionally, as current approaches to test-time scaling of compute concentrate on post-training techniques, we suggest that a complementary method of utilizing additional computational resources involves architectural modifications. For instance, consider a scenario where we need to predict a sentence in which most words have appeared previously in the context. In such cases, the model only needs to recall and reproduce these tokens, requiring reduced computational effort.

Sparse models have gone a recent renaissance, with Mixture-of-Experts models decoupling the parameter count from the compute per example, and retrieval-based models allowing attending only to the most relevant context of a large (equivalently long) external datastore. This paper introduces a new, simple method for conditional computation in language modeling that allows for both dynamic feed-forward and dynamic long-context processing. Our approach consists of two key contributions: First, we propose the All Attention Mixture of Experts architecture, that allows for a shared computational budget to dynamically allocate resources. By combining both dynamic feed-forward and dynamic long-context into a single differentiable layer, our method leverages the learning signal of conditional context retrieval to enhance the performance of feed-forward layers and vice versa, resulting in a higher quality model that improves perplexity and reduces computational cost compared to baseline models of similar size.

2 METHOD

Problem Setup For an input sequence of tokens $(x_1, \ldots, x_L)$ with their corresponding representations $(h_1, \ldots, h_L)$, where h_i is the representation of token x_i, the following sections describe the components of our model.

Feed-Forward Layer A feed-forward layer consists of two linear transformations separated by a non-linear activation function, typically ReLU. For an input vector $\mathbf{x}$, the output of the feed-forward layer is expressed as (excluding bias terms): $\text{FF}(\mathbf{x}) = \mathbf{V}^{\text{FF}}\,\text{ReLU}(\mathbf{x}\mathbf{K}^{\text{FF}})$, where $\mathbf{K}^{\text{FF}}$ and $\mathbf{V}^{\text{FF}}$ are learnable matrices with shape $d \times d_{\text{ff}}$.

Single-Head Attention Given a query matrix $\mathbf{Q} \in \mathbb{R}^{m \times d_k}$, a set of keys $\mathbf{K} \in \mathbb{R}^{n \times d_k}$, and values $\mathbf{V} \in \mathbb{R}^{n \times d_k}$, single-head attention is described by: $\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$, where m is the number of queries, n is the number of keys, and d_k is the dimension of the head.

Multi-Head Attention Multi-head attention involves projecting the input $\mathbf{X} \in \mathbb{R}^{L \times d}$ into query $\mathbf{Q}^{\text{ATT}}$, key $\mathbf{K}^{\text{ATT}}$, and value $\mathbf{V}^{\text{ATT}}$ matrices using learned weights $\mathbf{W}^Q$, $\mathbf{W}^K$, and $\mathbf{W}^V \in \mathbb{R}^{d \times d}$. Specifically, these projections are given by $\mathbf{Q}^{\text{ATT}} = \mathbf{X}\mathbf{W}^Q$, $\mathbf{K}^{\text{ATT}} = \mathbf{X}\mathbf{W}^K$, and $\mathbf{V}^{\text{ATT}} = \mathbf{X}\mathbf{W}^V$.

These are reshaped to $\mathbb{R}^{H \times L \times d_k}$ for H heads, where $d_k = d/H$. Attention is computed for each head i as $\text{head}_i = \text{Attention}(\mathbf{Q}_i^{\text{ATT}}, \mathbf{K}_i^{\text{ATT}}, \mathbf{V}_i^{\text{ATT}})$. The heads are concatenated and transformed back to $\mathbb{R}^{L \times d}$ using $\mathbf{W}^O$, yielding $\text{MHA}(\mathbf{X}) = \mathbf{W}^O[\text{head}_1; \ldots; \text{head}_H]$.

A Transformer Layer A parallel transformer layer includes multi-head self-attention and feed-forward sublayers, both with layer normalization, and uses a residual connection.

$$\mathbf{X}_{\text{out}} = \mathbf{X} + \text{MHA}(\text{LN}(\mathbf{X})) + \text{FF}(\text{LN}(\mathbf{X})) \tag{1}$$

All Attention Layer In the All-Attention layer, we first define the persistent memory matrices $\mathbf{M}_k$ and $\mathbf{M}_v$, which are analogous to $\mathbf{K}^{\text{FF}}$ and $\mathbf{V}^{\text{FF}}$ and have dimensions $\mathbb{R}^{d \times d_{\text{ff}}}$. These matrices are then used to form $\mathbf{K}^{\text{ALL}}$ and $\mathbf{V}^{\text{ALL}}$ by combining context-dependent vectors with persistent memory vectors. Specifically, in Multi-Head Attention, we use $\mathbf{K}^{\text{ALL}}$ and $\mathbf{V}^{\text{ALL}}$ instead of $\mathbf{K}^{\text{ATT}}$ and $\mathbf{V}^{\text{ATT}}$:

$$\mathbf{K}^{\text{ALL}} = [\mathbf{K}^{\text{ATT}}; \mathbf{M}_k] \qquad \mathbf{V}^{\text{ALL}} = [\mathbf{V}^{\text{ATT}}; \mathbf{M}_v] \tag{2}$$

$$= [\mathbf{x}_1\mathbf{W}^K; \ldots; \mathbf{x}_L\mathbf{W}^K; \mathbf{M}_k] \qquad = [\mathbf{x}_1\mathbf{W}^V; \ldots; \mathbf{x}_L\mathbf{W}^V; \mathbf{M}_v] \tag{3}$$

Where the concatenation is done along the sequence dimension, resulting in $\mathbf{K}^{\text{ALL}}, \mathbf{V}^{\text{ALL}} \in \mathbb{R}^{(L+d_{\text{ff}}) \times d}$. This allows the All-Attention layer to replace the feed-forward sublayer with persistent memory, thereby simplifying the architecture without compromising performance.

2.1 AA-MOE ARCHITECTURE

Problem Setup Given an input sequence of L tokens $(x_1, \ldots, x_L)$, we partition it into $\ell = \frac{L}{m}$ chunks $\mathcal{C} = (c_1, \ldots, c_\ell)$ of length m. Each token x_i belongs to chunk $\frac{i}{m}$ and can attend to chunks $\mathcal{C}^{\leq \frac{i}{m}} = \left(c_1, \ldots, c_{\frac{i}{m}}\right)$. We also partition memory into E experts $\mathcal{E} = (e_1, \ldots, e_E)$, each of size m. For token x_i, available resources (we use this term to collectively refer to both context chunks and experts) are $\mathcal{R}(x_i) = \mathcal{C}^{<\frac{i}{m}} \cup \mathcal{E}$, totaling $E + \frac{i}{m}$ options.

We present our method in two parts. First, our architecture extends the All-Attention layer by incorporating a Mixture-of-Experts (MoE) mechanism within the attention framework, followed by a describe the budget allocation mechanism, which dynamically allocates computational resources (both context and expert parameters) to each token based on its complexity, using a shared computational budget. Second, we describe the method in which we propogate signal through the discrete budget allocation mechanism.

2.2 Model Architecture

Router Score Calculation The router score calculation for context chunks and experts is the same in principle, but differs in implementation. For context chunks, the score is computed as a dot product between the token representation and the last token representation in the context chunk, while for experts, it is computed based on a learned expert embedding. Additionally, we apply rotary positional embeddings to the context chunk representations to encode relative positions. We compute router scores for both context chunks and experts:

$$r_{i,j} = h_i^T W_c \text{RoPE}(c_j) \text{ for context chunks} \qquad r_{i,l} = h_i^T e_l \text{ for experts} \tag{4}$$

where e_l is the l-th expert embedding, W_c is a learned weight matrix for context chunks, and $\text{RoPE}(\cdot)$ applies rotary positional embeddings to encode relative positions between tokens and chunks.

Budget Allocation Our budget allocation network operates on a total fixed computational budget B. Each token's representation predicts an importance score relative to the entire sequence, determining its proportional share of resources. Based on this relative importance prediction, the budget allocated to each token is its relative importance times B rounded-down. This budget is distributed jointly across experts and the context chunks it precedes (to maintain causality). Formally, at each layer, each token x_i receives a budget of $B_i = \lfloor B \cdot b_i \rfloor$, where $b_i \propto \exp(w_B^T h_i)$ and w_B is learned linear projection.

Representation Tensor We define the representation tensor by organizing token representations from context chunks and expert parameters into a unified structure:

$$\mathbf{X} \in \mathbb{R}^{(\ell+E)\times m\times d} \tag{5}$$

where:

- $\ell = L/m$ is the number of context chunks
- E is the number of expert parameters
- m is the chunk size
- d is the hidden dimension

The tensor is organized as:

$$\mathbf{X} = \begin{bmatrix} [h_1^1, \dots, h_1^m] \\ \vdots \\ [h_\ell^1, \dots, h_\ell^m] \\ [e_1^1, \dots, e_1^m] \\ \vdots \\ [e_E^1, \dots, e_E^m] \end{bmatrix} \tag{6}$$

$$\mathbf{X} \in \mathbb{R}^{(\ell+E)\times m\times d} = \begin{bmatrix} \text{Context Chunk Representations} \in \mathbb{R}^{\ell\times m\times d} \\ \text{Expert Parameters} \in \mathbb{R}^{E\times m\times d} \end{bmatrix} \tag{7}$$

Allocated Resources For each token representation h_i (from token x_i), we attend to the top-B_i resources according to the routing scores. Let $\mathcal{R}_i = \{j_1, \dots, j_{B_i}\}$ be the indices of the B_i resources with highest routing scores $r_{i,j}$ for representation h_i, where each $j_k \in [1, \ell + E]$ indexes into our representation tensor $\mathbf{X}$. The attention weights for token representation h_i are only computed for resources in $\mathcal{R}_i$, with all other attention weights set to zero. Formally:

$$\mathcal{R}_i = \text{argtop-}B_i(\{r_{i,j} : j \in \mathcal{R}(h_i)\}) \tag{8}$$

where $\mathcal{R}(h_i)$ contains indices of context chunks preceding the token representation h_i and all expert parameters.

For example, if token representation h_i is in chunk c_k, then $\mathcal{R}(h_i)$ contains indices $\{1, ..., k-1\}$ for previous context chunks and $\{\ell+1, ..., \ell+E\}$ for expert parameters, ensuring causal attention.

Token/Neuron-Level K/V Assignments Given resource-level assignments $\mathcal{R}_i$, we define the fine-grained token/neuron-level key vectors $\mathcal{R}_i^K$ by extracting the corresponding m-sized chunks from our representation tensor:

$$\mathcal{R}_i^K = \bigcup_{j \in \mathcal{R}_i} \begin{cases} \{(W^K \mathbf{X}_{j,1}, \ldots, W^K \mathbf{X}_{j,m})\} & \text{if } j \leq \ell \text{ (chunk)} \\ \{(\mathbf{X}_{j,1}, \ldots, \mathbf{X}_{j,m})\} & \text{if } j > \ell \text{ (expert)} \end{cases} \tag{9}$$

where $\mathbf{X}_{j,k}$ refers to the k-th vector in the j-th row of our representation tensor. $\mathcal{R}_i^V$ is defined analogously with W^V. We apply rotary positional embeddings to the chunk representations after projection. For each token i, this results in a matrix of size $m \cdot B_i \times d$.

To illustrate with an example:

- For a chunk ($j \leq \ell$): If chunk j is selected, we project all m vectors in that chunk using W^K and include them in $\mathcal{R}_i^K$
- For an expert ($j > \ell$): If expert j is selected, we directly include all m vectors from that expert's row in the representation tensor

This formulation allows us to:

- Process chunks and experts uniformly through the same attention mechanism
- Maintain the full context within each chunk/expert (all m vectors)
- Apply different transformations to chunks vs experts while keeping the tensor structure

Attention Calculation Using the above notation, the result of AA-MoE for token i is simply defined as applying multi-head attention to the token/neuron-level key and value vectors. Specifically, the key and value vectors are reshaped to $\mathbb{R}^{H \times m \cdot B_i \times d_k}$ for H heads, where $d_k = d/H$. For each head h, we compute $\text{head}_h = \text{Attention}(W_h^Q \mathbf{h}_i, \mathcal{R}_{i,h}^K, \mathcal{R}_{i,h}^V)$, where $\mathcal{R}_{i,h}^K, \mathcal{R}_{i,h}^V$ are the key and value vectors for head h from the allocated resources. The heads are concatenated and transformed back to $\mathbb{R}^{m \cdot B_i \times d}$ using $\mathbf{W}^O$, yielding $\text{MHA}_i = \mathbf{W}^O[\text{head}_1; \ldots; \text{head}_H]$.

2.2.1 Signal Propogation

Probability of Allocation We can define the probability that token i is allocated resource j using the existing variables and allocation mechanism in our algorithm, while taking into account the budget constraint and allowing gradient flow to all neural networks. Let $P(i, j)$ be the probability that token i is allocated resource j.

Motivation By using conditional probabilities, we can create a differentiable approximation of the discrete allocation process, allowing for end-to-end training of the neural networks while respecting budget constraints and capturing the competitive nature of the auction. To do this we can decompose the probability into two parts:

$$P(i, j) = P(i \text{ can afford } j) \cdot P(i \text{ chooses } j \mid i \text{ can afford } j) \tag{10}$$

Sanity Check As a sanity check, for a budget of 2.5, We should expect that the $k = 1$ item will get low gradients and the $k = 2$ item will get slightly higher gradients, because it closer to the edge of the budget.

The affordability probability is given by:

$$P(i \text{ can afford } k \text{ items}) = P(i \text{ can afford } j_k) = \sigma(B_i - k) \tag{11}$$

Where we assume that the price of each item is 1. and j_k is the index of the k-th item of buyer i. For our budget of 2.5 example we can compute the gradients for each item:

$$P(i \text{ can afford } k = 1 \text{ items}) = \sigma(2.5 - 1) = \sigma(1.5) \tag{12}$$

$$P(i \text{ can afford } k = 2 \text{ items}) = \sigma(2.5 - 2) = \sigma(0.5) \tag{13}$$

And since sigmoid saturates away from 0, we can see that the gradients for the first item will be much lower than the gradients for the second item.

The probability of allocating a resource (context chunk or expert) to a token is influenced by the affordability probability, assuming all prices are set to 1. This is derived from the auction mechanism, where we sum over all the item avialable to the buyer:

$$P(i, j_k) = \sigma(B_i - k) \cdot \frac{\exp(r_{i,j_k})}{\sum_l \exp(r_{i,j_l})} \tag{14}$$

where σ is the sigmoid function.

This formulation allows the auction mechanism to directly influence the attention weights by incorporating the allocation probabilities into the attention calculation. By assuming all prices are 1, we simplify the model.

Attention Calculation We incorporate the allocation probabilities into the attention calculation by adding their log values to the attention matrix. This mechanism allows the model to learn which resources (context chunks or experts) are most relevant for each token while maintaining differentiability. The attention scores then reflect both the content-based similarity between tokens and resources, as well as the auction-based allocation decisions.

This enables end-to-end training while allowing the model to dynamically balance between attending to context and utilizing experts.

We now formalize this mathematically:

$$\log P(i, j_k) = \log \sigma(B_i - k) + r_{i,j_k} - \log \sum_l \exp(r_{i,j_l}) \tag{15}$$

$$A_{i,j_k} = \frac{Q_i K_{j_k}^T}{\sqrt{d_k}} + \log P(i, j_k) \tag{16}$$

where A_{i,j_k} is the pre-softmax attention score between token i and resource j_k (which can be either a context chunk or an expert), Q_i is the query vector for token i, K_{j_k} is the key vector for resource j_k, and d_k is the dimension of the key vectors.

One key advantage of this approach is that it allows sharing the learning signal between context routers and expert routers, as we will later see in the experiments.

Additional advantage, in contrast with MoE layers, where there is no separate gating or weighting mechanism that determines the relative contribution of experts for each token as the attention mechanism already takes care of this.

Impresively, while there is no mechanism to prevent over-reliance on attention or MoE, it appears that the network learns to allocate budget to the most important tokens and deprioritizes the rest demonstrating that the model learns to balance between attending to context and utilizing experts.

Concequences of Softmax Budget Allocation In our formulation, using the floor function allows the model to set $B_i = 0$ for less important tokens, where the token is not allocated any resources. This allows the model to focus on more challenging tokens requiring more compute. Equivalently, a single token can use the entire budget.

Non-causal Budget Allocation A consequence of computing an importance score relative to the entire sequence is that the budget decisions for a given token depend on information from future tokens, which is not available during autoregressive procedures like sampling or perplexity calculation. To address this issue, we follow MoD and during training train a predictor to mimic the budget B_i for each token, based only on the current and past tokens.

References

A Appendix