Synergy: End-to-end Concept Model

Keli Zheng
Institute of Software, Chinese Academy of Science
zhengkl@ios.ac.cn
and **Zerong Xie**
The University of Hong Kong
zerong@connect.hku.hk

Abstract

In this paper, we present *Synergy*, a language model that bridges different levels of abstraction in an end-to-end fashion through a learned routing mechanism. Focusing on low-level linguistic abstraction, we trained our model as a byte-level language model. Our model spontaneously learns to tokenize bytes, producing fewer concept tokens than Byte-level Byte Pair Encoder (BBPE) tokenizers whiling keeping the comparable performance. By comparing with *Llama3*, we observed an advantage of *Synergy* under the same model scale and training dataset size. Further studies show that the middle part (the higher abstraction part) of our model performs better when positional encodings are removed, suggesting the emergence of position-independent concepts. These findings demonstrate the feasibility of tokenizer-free architectures, paving the way for more robust and flexible pipelines.

1 Introduction

In the last few years, large language models (LLMs) have shown dazzling capabilities and enormous potential to solve a wide range of tasks. Many companies and research organizations have released their own large language models, such as ChatGPT, Llama, DeepSeek, etc., which have profoundly changed the way we work and live. Most of these products share a similar principle: a decoder-only-transformer-like model fed on a large number of tokens and trained with the predict-next-token task. Although researchers today have further developed sophisticated model structures and cleverer training approaches, the aforementioned principle still remains the core of most LLMs.

However, as LCM-team et al. (2024) pointed out, such a core principle results in a drawback: it cannot efficiently process information at different levels of abstraction. Since the model is trained at the token level, it tends to "think" at the token level. However, many high-level abstract concepts are difficult to accurately describe in existing languages. It may prevent the model from performing better in scenarios where higher-level abstractions are critical, such as outlining a presentation, planning a tour, engineering a complicated program, etc. Many researchers believe that the ability to handle high-level abstract concepts can be spontaneously developed through training due to the multilayer structure of the transformer model. But a properly designed neural network structure can lead to a more efficient processing of high-level abstract concepts.

Creatively, in the Large Concept Model (LCM) (LCM-team et al., 2024) an autoencoder-based sentence embedding model is used to abstract token-level embeddings into sentence-level embeddings, and an encoder-only transformer model is fed with sentence-level embeddings and trained to predict next sentence embedding. Although their experiments have shown initial success, their solutions have an unavoidable disadvantage: The sentence embedding model is trained separately from the transformer model and have a quite different training target (in this case, as an autoencoder, to restore the original input), therefore the abstracted information may not be useful for the ultimate goal (in this case, to predict the next sentence embedding), leading to a low efficiency.

To overcome this disadvantage, we propose an end-to-end model *Synergy*, where the encoding/decoding part and the processing part are trained together. The main challenge to train an end-to-end model is that the aforementioned two parts are working on token sequences with different lengths, which does not fit in a conventional decoder-only transformer. An encoder-decoder transformer is also not applicable due to the temporal dependency within/between these two sequences. Inspired by Mixture of Depths (MoD) (Raposo et al., 2024), we bridge those se-

quences using a router mechanism. Further, we configured those parts correspondingly to encourage a "division of labor" across different levels of abstraction, resulting in better efficiency.

Tokens are also concepts at a low level of abstraction, except that they have already been symbolized in our language. Using statistic-based tokenizers like Byte-level Byte Pair Encoding (BBPE) tokenizers, we already achieved a quite efficient abstraction from characters/bytes to tokens/words. However, with the attempts to process multimodal data such as images, audio, video, etc. with LLMs, the ability to process data in low-level of abstractions becomes important.

Considering the fact that low-level abstraction is relatively easier, and therefore consumes affordable resources, our experiments focused on low-level abstraction tasks. We compared our model *Synergy* with *Llama3*, and observed an advantage under the same model scale and training dataset size. Further studies show that *Synergy* can generate fewer tokens than BBPE tokenizers. In addition, we observed an unexpected decrease of loss when removing positional encoding from the middle part, suggesting the emergence of position-independent concepts.

In short, our contributions are:

- We proposed an end-to-end model *Synergy* that bridges different levels of abstraction.
- We conducted experiments to show the efficiency and feasibility of tokenizer-free *Synergy* as a byte-level language model.
- We uncovered the emergence of position-independent concepts in the middle part of *Synergy* by removing positional encoding.

2 Methodology

The main idea of *Synergy* is to reduce the number of tokens using a routing mechanism, then process the resulting shorter sequence using a decoder-only transformer. Like a normal decoder-only transformer, the whole model consumes a sequence of tokens as input, produces a corresponding sequence of tokens as output, and is trained by predict-next-token tasks.

As it shows in fig. 1a, the model is split into three parts — encoder, middle, and decoder, which are all decoder-only transformers. The output from the encoder is passed to the decoder. For a portion of tokens, the encoder output also passes through the middle part and then adds onto the decoder input.

A router is responsible to determine which tokens can go through the middle part. As it shows in figs. 1b and 1c, the routing mechanism is similar to that in MoD (Raposo et al., 2024). A weight factor w_i is computed from the encoder output x_i through a simple linear layer for each token (indexed by i). A portion of tokens are picked to pass through the middle part via a top-k operation on the weight factor. The picking result is represented as a mask m_i. For the picked tokens, the middle output $\text{Middle}(x_i)$ is multiplied by a gating factor σ_i before adding to the decoder input. The routing mechanism can be formally described as the following formulas:

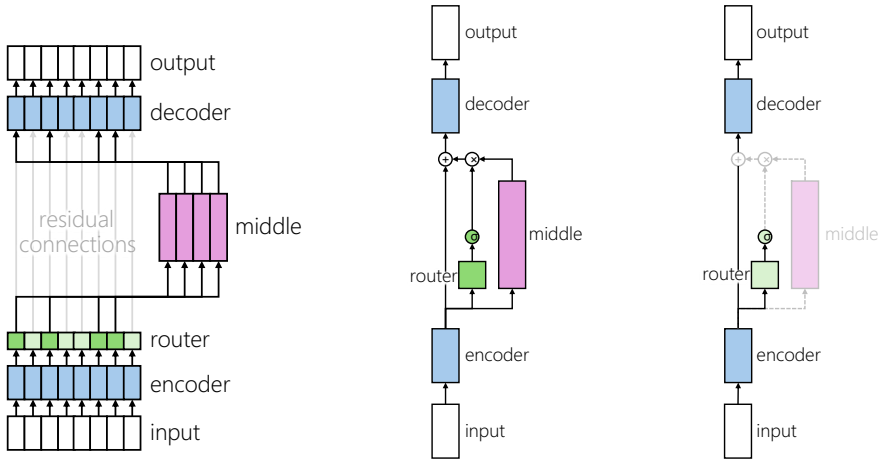
$$w_i = \text{Router}(x_i) \tag{1}$$
$$m_i = \text{topkmask}(w_i, k) \tag{2}$$
$$\sigma_i = \text{sigmoid}(w_i) \tag{3}$$
$$y_i = x_i + m_i\ \sigma_i\ \text{Middle}(x_i) \tag{4}$$

Where,

- x_i — The encoder output of the i-th token.
- w_i — The weight factor of the i-th token computed by the router.
- k — A hyperparameter that controls the number of tokens that can pass through the middle part.
- m_i — The mask that represents whether the i-th token can pass through the middle part.
- σ_i — The gating factor of the i-th token computed from w_i.
- y_i — The decoder input of the i-th token.

Here we can consider the weight factor w_i as an importance indicator of each token. While a token is important, it is reasonable to allocate more computation to process it, so it activates the middle part. As a result, the middle part processes fewer tokens than the encoder/decoder part, achieving our goal of token number compressing.

We assume that the encoder/decoder and middle parts work in different levels of granularity, for example, the encoder/decoder part should be responsible for concrete tasks, focusing on few tokens, while the middle part should take on abstract tasks, involving long-range context. So we used different configurations for them respectively. On the one hand, we use local attention in the encoder/decoder part. On the other hand, we removed the positional encoding in the middle part.



(a) The overview (b) Routing (active) (c) Routing (inactive)

Figure 1: The architecture of Synergy

3 Experiments

3.1 Experimental Setup

3.1.1 Dataset Configurations

In all experiments, we use the English Wikipedia text corpus [1], where 70% randomly sampled rows are used for training, remaining the rest for evaluation and testing. We chose this dataset because we found it best highlights the characteristics of our model.

To train our model *Synergy* as a byte-level language model, we tokenize the dataset using a simple UTF8 tokenizer, which encodes any text into bytes according to the UTF-8 standard and then treats each byte as a token.

3.1.2 Model Hyperparameters

In all experiments, our model *Synergy* has totally 32 layers, with 4 layers in the encoder part, 24 layers in the middle, and 4 layers in the decoder part. The embedding dimension and the hidden dimension of all transformer layers, is 1024. Every transformer layer consists of a 16-head self-attention layer with 64-dimensional Q, K, V, and an MLP layer with SwiGLU activation (Shazeer, 2020) and 4096-dimension intermediate [2].

By default, the model is trained with sequences of length 1024, and the corresponding number of tokens in the middle part is 224. For positional encoding, we choose the widely used RoPE (Su et al., 2023). Recall that we deliberately removed the positional encoding in the middle part, so it only affects the encoder and decoder.

3.1.3 Hardware and Costs

We finished all experiments on a single machine with one GPU (NVIDIA H20 96GB). The whole training set consists of about 12G bytes. Using our experimental configurations, the whole training phase of *Synergy* lasts for about 62 hours (while it lasts for about 40 hours for *Llama3*).

3.2 Comparison between *Synergy* and *Llama3*

To evaluate the ability of our model to abstract word-level concepts, we arranged a comparison between *Synergy* and *Llama3*. For *Llama3* we use the official tokenizer with 128k words/phrases in its vocabulary, while *Synergy* is equipped with a simple UTF8 tokenizer as mentioned above.

To ensure a fair comparison, a number of special configurations are used:

- Since two models are using different tokenizers, the perplexity is no longer a suitable metric for this comparison. Follow previous works (Xue et al., 2022; Yu et al., 2023; Wang et al., 2024; Pagnoni et al., 2024), we use Bits-Per-Byte (BPB), which is independent of tokenizer, as the metric for comparison.
- Since two models are using different tokenizers, it becomes tricky to ensure an equal context size for both models. To avoid this unfairness, we configured a long enough context size for both models correspondingly (*Synergy*: 1024 bytes; *Llama3*: 224 tokens). Then we clipped the training text samples into shorter independent segments so that one seg-

[1] To be exact, we use the subset 20231101.en from https://huggingface.co/datasets/wikimedia/wikipedia/viewer/20231101.en

[2] The $\frac{2}{3}$ factor introduced by SwiGLU is **not** multiplied.

ment can be fit into the context after tokenization for both models. It is fair for both models because they are trained with the same text segments without clipping.

- To ensure the fairness in model scale, we use 32 layers in the *Llama3*, which is consistent with the total number of layers in *Synergy*, and use the same set of hyperparameters for all transformer layers. However, there is still a difference in the number of parameters (*Llama3*: 0.8B, *Synergy*: 0.5B), which is caused by the embedding layer: while *Synergy* has a tiny vocabulary (256 plus special tokens), *Llama3* has a large vocabulary (upto 128k), requiring extra 0.3B parameters for token embedding. Despite this difference, we still consider two models matched, as they have the same number of transformer layers.

As shown in fig. 2, our model yields a better result when the training data exceeds 0.6T bytes, which indicates an advantage of our model compared to *Llama3*. However, the price of getting this advantage is about 1.5 times more computation time, see section 5.2 for more details.

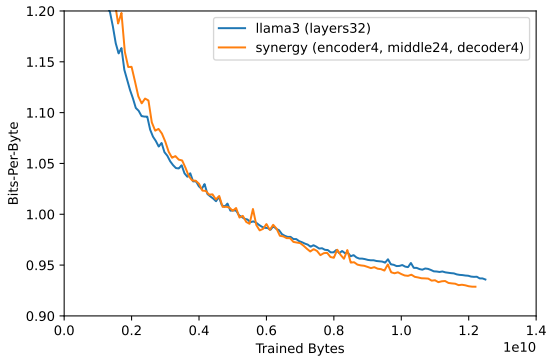


Figure 2: BPB curves of both models

3.3 Positioning Modes

One of the special designs of our model is the removal of positional encoding for the middle part. It originates from a series of experiments with different positioning modes for the middle part. We can roughly divide them into three categories:

- Original: use the original position picked from the input.
- Sigma: use the accumulated value of the gating factor cumsum(σ_i) as the position.
- None: do not use any positional encoding.

For "Sigma", there are variants. We can accumulate gating factors of all tokens or only those of picked tokens. And we can remain or cut the gradient trace of σ_i when computing the positional encodings. In total, we have 4 variants of "Sigma".

Positioning Mode	BPB (at 5.5T token)
original	1.0164
sigma	1.0747
sigma_grad	1.0523
sigma_all	1.0251
sigma_all_grad	1.0630
none	**0.9906**

Table 1: BPB values of different positioning modes

We trained our model correspondingly with all 6 modes. Considering the training cost, we stopped most training experiments at 6T tokens, where the trend is stable enough. As we can see in fig. 3, the "None" mode is the best among them. We collected BPB values and listed them in table 1.

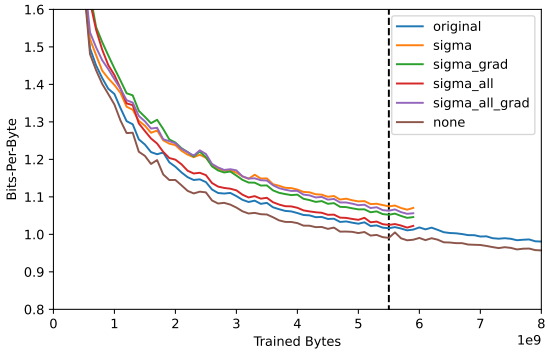


Figure 3: BPB curves of different positioning modes

This phenomenon is quite above our expectations. We assume that this is because the concept in the encoder output is actually independent of concrete positions, and thus, in contrast, it is interfered with by positional encodings. It does not mean that the concept is completely independent of position. The position information about can be extracted in the encoder part and absorbed into the concept. Such a phenomenon also reveals a potential — as the middle part does not rely on the positional encoding, it may have better extrapolation performance.

3.4 Concept Tokens Number

To inspect how Synergy tokenizes bytes, we visualized the output of the router. As we can see in fig. 4, the weight factor shows an obvious relation with word separations, which means that our model somehow spontaneously learns the ability to

segment the bytes into words.

In addition, we observed that a number of bytes are picked with low weight factors. This indicates a redundancy in tokenization. These bytes have low weight factors and correspondingly have low gating factors, so that we can actually drop these tokens from the middle part without significant impact on the final output.

To further investigate this redundancy, we conduct a series of experiments to test the influence of the concept token count (i.e., the number of tokens processed in the middle part) on the performance. As shown in fig. 5, the performance does not decrease until the number of concept tokens falls below 192, while the number of tokens from *Llama3* tokenizer is about $1024/4.25 = 240.94$. This result shows the potential of our model to tokenize bytes more efficiently, yielding fewer concept tokens than BBPE tokenizers.

4 Related Works

Prior to our work, there have been a number of works related to the abstraction of information, some of which has inspired our design. We organized the related works in the following subsections classified by their approaches.

4.1 Sparse Attention

Sparse attention is a powerful technique to reduce the computational cost of attention, especially in long-context scenarios.

In the early stages, there exists a group of sparse attention methods which rely on predefined structures like fixed strides (Child et al., 2019), dilated sliding window (Beltagy et al., 2020) and attention sinks (Xiao et al., 2024). The sparsity in these methods has little correlation with the content, which may limit their efficiency.

To address this issue, a group of dynamic sparse attention mechanisms were proposed. The representative ones are H2O (Zhang et al., 2023), Quest (Tang et al., 2024), Minference (Jiang et al., 2024) and RetrievalAttention (Liu et al., 2024). These methods significantly reduced the computational cost by maintaining a subset of KV pairs in long-context inference. But these methods cannot be applied to the training phase.

In recent years, new approaches, such as NSA (Yuan et al., 2025) and MoBA (Lu et al., 2025), introduce content-dependent block-wise sparsity to reduce the computational cost of attention. These methods can speed up both training and inference phases. However, they still cannot easily handle abstract information that diffuses throughout the long sequence (e.g., ambient gradients, style transformation).

Moreover, the improvement of attention does not change the fact that the whole model still operates at the token level, and is challenging to process abstract concepts that cannot be clearly described in words.

4.2 RNN-like language model

Except for the aforementioned efficient attention, there are other approaches that break the fundamental principle of attention, resulting in a number of RNN-like language models. Starting with Linformer (Wang et al., 2020), a number of attempts like RWKV (Peng et al., 2023, 2024), Mamba (Gu and Dao, 2024; Dao and Gu, 2024), and RetNet (Sun et al., 2023) have made significant progress.

However, despite the high efficiency of these models in dealing with long-context tasks, they remain confined to a single level of abstraction, in most cases, the level of words. We note that in some of these rnn-like models, state transition with different frequencies is deliberately involved to handle information of different granularity, which corresponds to our idea. But instead of processing everything in a sequence at the finest granularity, it would be more efficient to separately process sequences of granularity.

4.3 Hierarchical transformers

Hierarchical transformers, as a promising approach to process long sequences, have been studied in many previous works. Hourglass (Nawrot et al., 2022), MegaByte (Yu et al., 2023) and Block Transformer (Ho et al., 2024) are proposed with a hierarchical transformer architecture with fixed-size pooling and upsampling mechanisms. However, fixed block size limits its adaptability to various situations.

MrT5 (Kallini et al., 2025) introduces an efficient variant of ByT5 that integrates a dynamic token deletion mechanism, but it is not applicable to decoder-only transformers. In the meantime, Dynamic-Pooling Transformer (Nawrot et al., 2023; Ahia et al., 2024) introduced a boundary predictor trained by Gumbel-sigmoid stochastic reparameterization. Their idea corresponds to ours, but uses a different approach to train the router.

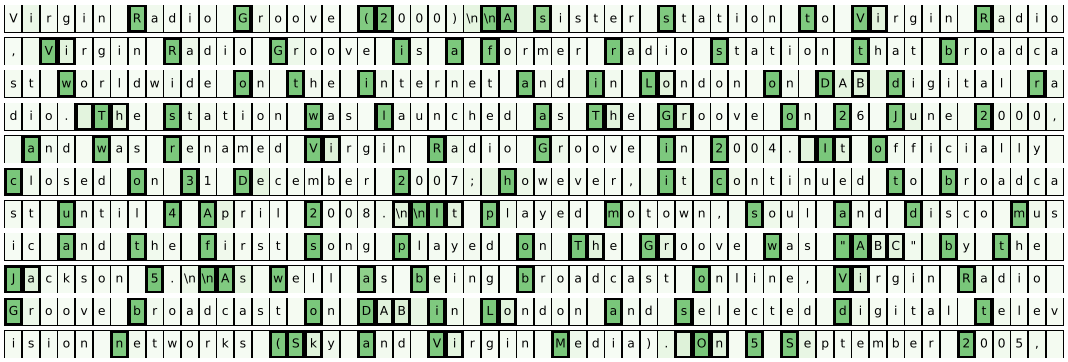


Figure 4: Visualization of router output, where green color represents the weight factor (the darker green, the greater weight) and the black border represents the routing mask (bold border means passing through the middle part).

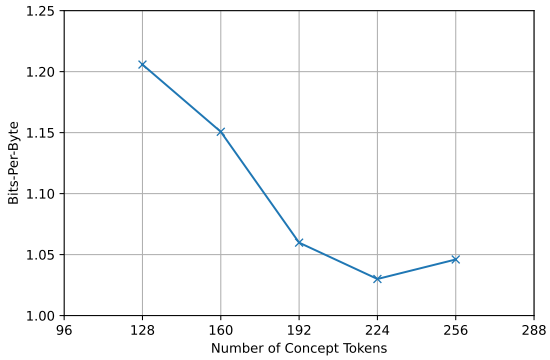


Figure 5: The influence of concept tokens counts to BPB (collected at 4T tokens).

And another difference is that our model has no deliberate pooling/upsampling because we do not presuppose the task to be word segmenting, but token filtering/compressing.

Moreover, BLT (Pagnoni et al., 2024) employed an entropy-based segmenting mechanism to segment the long sequence into patches (this approach is also mentioned in Dynamic-Pooling Transformer (Nawrot et al., 2023)). LCM (LCM-team et al., 2024) uses a sentence embedding model to extract sentence-level embeddings from long sequences and then process them with a predict-next-sentence-embedding decoder-only transformer, making good progress in abstracting sentence-level concepts. These methods have initially inspired our designs.

5 Limitations

5.1 Instability

We observed the instability during the training of our model. For example, at the beginning phase of the training, we observe glitches (a sudden spike in BPB) from time to time. In some cases, the accuracy converges to a higher value after the glitch, resulting in a poor training result. Even if there is no glitch, each training session may also yield very different BPB values, even using the same hyperparameters.

We hypothesize that this is caused by the imperfect router training method. Since there is no direct gradient connection from token sampling probabilities to the loss (recall that the sampling is operated by topK), the change of picking rule is unstable. On the one hand, breaking changes can be committed during training, resulting in glitches. On the other hand, beneficial changes may fail to occur, causing a poor result. One of our future works is to explore better ways to train the router both stably and cost efficiently.

Considering this limitation, the experimental results presented in this paper are partial results after being filtered out the poor outliers. Due to the high costs of pretraining a model from scratch, yet we were unable to repeat all experiments multiple times. But all results are confirmed to be reproducible.

5.2 Extra FLOPs

In our experiment, *Synergy* consumes about 1.5 times more FLOPs than *Llama3*. But it is worth noting that the extra FLOPs do not come from attention over long sequences, but from the MLP layers in encoder/decoder that inevitably run on each byte. Considering that every token has about 4.25 bytes on average under our experimental setting, such extra FLOPs are reasonable. In long-context scenarios, where the attention dominates the FLOPs consumption, this disadvantage will be mitigated. On the other hand, considering that the encoder and decoder are using local attention, a proper cache-aware implementation may greatly

improve its performance.

6 Conclusion

In this paper, we proposed an end-to-end model *Synergy* that bridges different levels of abstraction through a learned routing mechanism. We trained *Synergy* as a byte-level language model and compared it with *Llama3*, which uses a BBPE tokenizer, and observed an advantage of our model over *Llama3* under the same model scale and training dataset size. Further studies show that our model can compress bytes into fewer concept tokens than BBPE tokenizers, which reflects its efficiency in abstracting word-level concepts. Moreover, experiments on different positioning options for the middle part yield an unusual result: the model works best when the positional encoding is removed. This phenomenon shows that the concepts processed in the middle part are somehow position-independent. Our findings demonstrate the feasibility of tokenizer-free architectures, paving the way for more robust and flexible pipelines.

However, our model is far from perfect. It suffers from instability in training and extra computational cost. It leaves much to explore for the future, such as exploring better ways to train the router, fine-tuning the layers of each part for better efficiency and using specialized structures for encoder, middle, and decoder parts. In addition, it is worth investigating the extrapolation performance of the middle part in the absence of positional encoding. It is also worth a try to use our model to abstract sentence-level concepts.

References

Orevaoghene Ahia, Sachin Kumar, Hila Gonen, Valentin Hofmann, Tomasz Limisiewicz, Yulia Tsvetkov, and Noah A. Smith. 2024. Magnet: Improving the multilingual fairness of language models with adaptive gradient-based tokenization. *Preprint*, arXiv:2407.08818.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *Preprint*, arXiv:2004.05150.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *Preprint*, arXiv:1904.10509.

Tri Dao and Albert Gu. 2024. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *Preprint*, arXiv:2405.21060.

Albert Gu and Tri Dao. 2024. Mamba: Linear-time sequence modeling with selective state spaces. *Preprint*, arXiv:2312.00752.

Namgyu Ho, Sangmin Bae, Taehyeon Kim, Hyunjik Jo, Yireun Kim, Tal Schuster, Adam Fisch, James Thorne, and Se-Young Yun. 2024. Block transformer: Global-to-local language modeling for fast inference. *Preprint*, arXiv:2406.02657.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2024. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *Preprint*, arXiv:2407.02490.

Julie Kallini, Shikhar Murty, Christopher D. Manning, Christopher Potts, and Róbert Csordás. 2025. Mrt5: Dynamic token merging for efficient byte-level language models. *Preprint*, arXiv:2410.20771.

LCM-team, Loïc Barrault, Paul-Ambroise Duquenne, Maha Elbayad, Artyom Kozhevnikov, Belen Alastruey, Pierre Andrews, Mariano Coria, Guillaume Couairon, Marta R. Costa-jussà, David Dale, Hady Elsahar, Kevin Heffernan, João Maria Janeiro, Tuan Tran, Christophe Ropers, Eduardo Sánchez, Robin San Roman, Alexandre Mourachko, and 2 others. 2024. Large concept models: Language modeling in a sentence representation space. *Preprint*, arXiv:2412.08821.

Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, Chen Chen, Fan Yang, Yuqing Yang, and Lili Qiu. 2024. Retrievalattention: Accelerating long-context llm inference via vector retrieval. *Preprint*, arXiv:2409.10516.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, Zhiqi Huang, Huan Yuan, Suting Xu, Xinran Xu, Guokun Lai, Yanru Chen, Huabin Zheng, Junjie Yan, Jianlin Su, and 6 others. 2025. Moba: Mixture of block attention for long-context llms. *Preprint*, arXiv:2502.13189.

Piotr Nawrot, Jan Chorowski, Adrian Lancucki, and Edoardo Maria Ponti. 2023. Efficient transformers with dynamic token pooling. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6403–6417, Toronto, Canada. Association for Computational Linguistics.

Piotr Nawrot, Szymon Tworkowski, Michał Tyrolski, Łukasz Kaiser, Yuhuai Wu, Christian Szegedy, and Henryk Michalewski. 2022. Hierarchical transformers are more efficient language models. *Preprint*, arXiv:2110.13711.

Artidoro Pagnoni, Ram Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason Weston, Luke Zettlemoyer,

Gargi Ghosh, Mike Lewis, Ari Holtzman, and Srinivasan Iyer. 2024. Byte latent transformer: Patches scale better than tokens. *Preprint*, arXiv:2412.09871.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, Kranthi Kiran GV, Xuzheng He, Haowen Hou, Jiaju Lin, Przemyslaw Kazienko, Jan Kocon, Jiaming Kong, Bartlomiej Koptyra, Hayden Lau, and 15 others. 2023. Rwkv: Reinventing rnns for the transformer era. *Preprint*, arXiv:2305.13048.

Bo Peng, Daniel Goldstein, Quentin Anthony, Alon Albalak, Eric Alcaide, Stella Biderman, Eugene Cheah, Xingjian Du, Teddy Ferdinan, Haowen Hou, Przemysław Kazienko, Kranthi Kiran GV, Jan Kocoń, Bartłomiej Koptyra, Satyapriya Krishna, Ronald McClelland Jr., Jiaju Lin, Niklas Muennighoff, Fares Obeid, and 11 others. 2024. Eagle and finch: Rwkv with matrix-valued states and dynamic recurrence. *Preprint*, arXiv:2404.05892.

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. *Preprint*, arXiv:2404.02258.

Noam Shazeer. 2020. Glu variants improve transformer. *Preprint*, arXiv:2002.05202.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. 2023. Roformer: Enhanced transformer with rotary position embedding. *Preprint*, arXiv:2104.09864.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. 2023. Retentive network: A successor to transformer for large language models. *Preprint*, arXiv:2307.08621.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. Quest: Query-aware sparsity for efficient long-context llm inference. *Preprint*, arXiv:2406.10774.

Junxiong Wang, Tushaar Gangavarapu, Jing Nathan Yan, and Alexander M. Rush. 2024. Mambabyte: Token-free selective state space model. *Preprint*, arXiv:2401.13660.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-attention with linear complexity. *Preprint*, arXiv:2006.04768.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. *Preprint*, arXiv:2309.17453.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2022. Byt5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306.

Lili Yu, Dániel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. 2023. Megabyte: Predicting million-byte sequences with multiscale transformers. *Preprint*, arXiv:2305.07185.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y. X. Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. 2025. Native sparse attention: Hardware-aligned and natively trainable sparse attention. *Preprint*, arXiv:2502.11089.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. 2023. H_2o: Heavy-hitter oracle for efficient generative inference of large language models. *Preprint*, arXiv:2306.14048.