INTERNAL ATTENTION: GROWING TOKEN REPRESENTATIONS WITHOUT GROWING THE FFN OR THE KV CACHE

Anonymous authors
Paper under double-blind review

ABSTRACT

Increasing the width of a token's representation is expensive: parameters grow quadratically, dominated by the feed-forward network (FFN), and the KV cache grows linearly. We combine two ideas into an architecture that avoids both costs. First, inspired by Hyper-Connections, we split the token representation into a compute subspace and a memory subspace, where the FFN block is applied to the compute subspace only. Following Multi-Head Latent Attention, we also restrict the attention's key and value projections to the compute width. Quadratic costs in the number of parameters are therefore tied to the compute width and decoupled from the memory width. Second, we introduce a mechanism different from Hyper-Connections for moving information between the memory and compute subspaces: an attention internal to the token representation, which separates it into a fixed number of internal tokens and applies attention among them rather than across the sequence. An internal attention on the full vector followed by an FFN on the compute part forms a block that can be stacked, increasing depth without increasing the KV cache. Experiments on small autoregressive language models and integer multiplications show that the resulting architecture improves on both standard transformers and Manifold-Constrained Hyper-Connections (mHC) at equal KV cache, including in parameter efficiency.

1 INTRODUCTION

Language models based on transformers do not separate memory from computation. Each layer operates on a single vector per token. This vector must therefore hold not only the values currently being computed on, but also a memory: intermediate results that later layers may need and whatever should be exposed to future tokens through the keys and values. We argue that these roles should be separated.

To see why, we propose a computer analogy. The token representation plays the role of the CPU registers, and the feed-forward network (FFN) that of the compute unit acting on them. In turn, past tokens play the role of the main memory, and the attention mechanism that of a read operation moving data from main memory to the registers[1]. Note that writing to main memory can only be done by emitting new tokens, making a chain of thought necessary whenever more memory than the registers provide is required. As in a CPU, accessing the registers is cheap, but having many registers is costly, while reading the main memory is expensive. Traditional CPUs solve this by adding extra levels of memory, be it CPU cache or RAM. Several transformer variants can be read as adding such a level, although they were not framed this way.

A first family of approaches works on the main memory itself: prepending dedicated memory tokens to the sequence (Burtsev et al., 2021; Darcet et al., 2024), inserting summary tokens every few steps (Mohtashami & Jaggi, 2023; Chevalier et al., 2023; Bulatov et al., 2022), restricting attention to a

[1]The analogy is not exact. Attention is not a pure read, and attention-only models can compute non-trivial functions (Sukhbaatar et al., 2019; Elhage et al., 2021); conversely, feed-forward layers have been shown to behave as key–value memories (Geva et al., 2021). The analogy describes where each component's *cost* lies, not a strict division of labor.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

local window plus a few global tokens (Beltagy et al., 2020; Zaheer et al., 2021), or compressing past tokens (Rae et al., 2019; Yuan et al., 2025).

A second family, more directly tied to the analogy, adds extra vectors to the representation (Zhu et al., 2025a) and the Manifold-Constrained Hyper-Connections (mHC) variant (Xie et al., 2025) as well as (Zhu et al., 2025b). While it was initially introduced to extend the residual connection and solve the post-norm vs. pre-norm trade-off, this line of work can be reframed as adding an extra memory level. Indeed, the token representation goes from one vector to n vectors. Before each layer, those n vectors are mixed to give a single vector. The output of the layer, once again a single vector, is dispatched to those n vectors, which are additionally mixed among themselves by a learned $n \times n$ matrix. The n vectors therefore act as RAM, with the combination and dispatch weights acting as read and write operations. This approach grows neither the KV cache nor the FFN.

Our paper builds upon this idea. The token representation is grown to include a memory component, upon which the FFN does not act. A token representation x can therefore be divided into (x_c, x_m), where x_c is a compute component that feeds into the FFN, and x_m is a memory component. To move information between the compute and memory components, we depart significantly from Hyper-Connections and use an internal attention mechanism: the vector x is divided into k chunks of dimension c, each chunk being treated as an "internal" token. Like Hyper-Connections, this grows neither the FFN nor the KV cache: following Multi-Head Latent Attention (DeepSeek-AI et al., 2024), the keys and values of the temporal attention are projected to the dimension of the compute component.

Mixing the components of a representation is not new, but existing mechanisms differ from ours in what the mixing ranges over. A first line of work attends along the channel axis, treating each channel as a token whose features are the whole spatial or temporal extent of the sequence (El-Nouby et al., 2021; Ding et al., 2022; Zhang et al., 2020; Liu et al., 2024); this is the transpose of standard attention, and as such it is a global, cross-position operation, incompatible with causal decoding. Ours is strictly position-wise: the internal tokens are chunks of a single token's representation, so the mechanism is causal by construction and adds nothing to the KV cache. A second line replaces the FFN with attention over learned parameter tokens (Wang et al., 2025; Lample et al., 2019; Berges et al., 2024); there the keys and values are weights, whereas our chunks attend to one another. Closest in structure is Transformer-in-Transformer (Han et al., 2021), which runs an inner transformer over the sub-patches of each image patch; its inner tokens, however, are input-level sub-patches carried in a separate residual stream, while ours are a reshaping of the residual vector itself. Finally, in the terminology of Tolstikhin et al. (2021), our internal attention is a channel-mixing operation, but a content-dependent one: where MLP-Mixer and the standard FFN mix channels through a fixed learned map, attention mixes them through weights computed from the representation itself.

Section 2 describes the architecture in detail. Section 3 is dedicated to experiments, where we show at a small scale that our architecture outperforms both standard transformers and mHC.

2 Internal Attention Transformer (IAT)

A standard transformer layer is composed of a temporal attention, followed by an FFN. An internal attention transformer (IAT) layer introduces a subdivision between compute and memory, by splitting the token representation x into (x_c, x_m), having d and m dimensions respectively. At a high level, an internal attention transformer layer is then composed of a rank-d temporal attention following DeepSeek-AI et al. (2024), followed by an internal attention that mixes the compute and memory parts, and finally an FFN acting on x_c only. Figure 1 summarizes a layer.

2.1 Detailed description

We give here a detailed description of the components of an internal attention transformer layer.

Embedding and Head We initialize the memory part of the embedding to $x_m = 0$ rather than to a learned vector. The output head only reads x_c.

Internal Attention The internal attention mechanism assumes that x is a vector of size $c\,k$, where we call c the internal dimension. The vector is then split into chunks $x_1, \ldots, x_k$ of size c, where

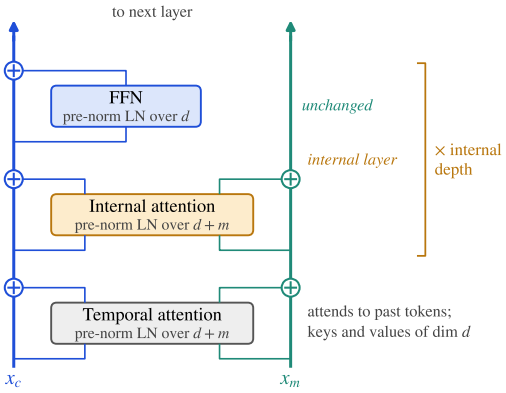

Figure 1: One internal attention transformer layer

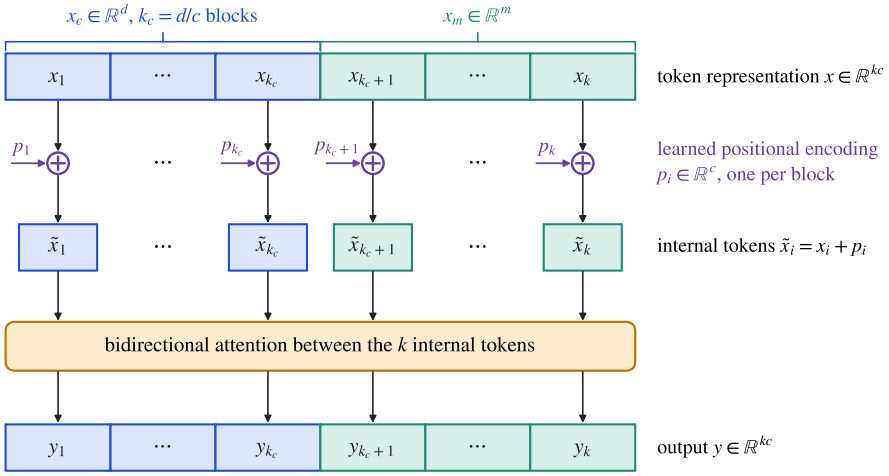

Figure 2: Internal attention on a single token. The representation $x \in \mathbb{R}^{kc}$ is split into k chunks of size c, the first $k_c = d/c$ in the compute part and the last $k - k_c$ in the memory part.

each x_i is seen as an internal token. The internal attention is therefore a standard bidirectional attention on the internal tokens $x_1, \ldots, x_k$: queries, keys and values are computed with projections $W_Q, W_K, W_V \in \mathbb{R}^{c \times c}$, applied to every internal token as in standard attention, split into n_c heads as in standard multi-head attention, and the output is projected back by $W_O \in \mathbb{R}^{c \times c}$ before the chunks are concatenated again into a vector of size $c\,k$. While it is not strictly necessary, we also assume that m and d are multiples of c, so that each chunk belongs entirely to either the compute or the memory part.

In addition, since the internal sequence length is fixed at k, we use a learned positional encoding: a vector $p_i \in \mathbb{R}^c$ is added to each chunk x_i before the projections. Since the mechanism acts on each token independently, it is causal by construction and does not contribute to the KV cache.

Figure 2 illustrates the mechanism on a single token.

Remark 1. *With properly chosen positional encodings, internal attention can fetch memory from a target "address". Informally, if one wants to add the content of chunk j into chunk i, one can write the positional encoding p_j of chunk j into chunk i (acting as a kind of pointer to j), and, with properly chosen attention matrices, the internal attention will give chunk i a very high attention score to chunk j.*

Parameters	Transformer, width d	Internal attention transformer
Temporal attention	$4d^2$	$4d(d+m)$
Internal attention	—	$4c^2 + d + m$
FFN	$8d^2$	$8d^2$

Table 1: Parameters of one layer (biases and LayerNorms omitted), for an FFN of hidden size $4d$. With internal depth L_{int}, the internal attention and FFN rows are multiplied by L_{int}.

Low-rank temporal attention Following DeepSeek-AI et al. (2024), we use a low-rank temporal attention: the queries, keys and values are projected from the full representation $x \in \mathbb{R}^{d+m}$ to $\mathbb{R}^d$, split into heads of total dimension d, and the output of the attention is projected from $\mathbb{R}^d$ back to $\mathbb{R}^{d+m}$. The temporal attention can therefore read the memory of past tokens and write into the memory of the current token, but the map from values to outputs has rank at most d (Bhojanapalli et al., 2020). Since the stored keys and values have dimension d, it does not increase the KV cache.

We also tried another option, which is to have the temporal attention act on the compute part x_c only. It performs worse than the baseline (see Appendix B).

FFN The feed-forward layer is standard, except that it acts only on x_c, keeping the compute cost in check.

Residual connections and layer norms Each of the three sublayers is preceded by LayerNorm in a pre-norm setting, and has a residual connection. LayerNorm and residual connections are taken over the dimensions a given sublayer acts on: over the full $d + m$ dimensions before the temporal and internal attentions, and over the d dimensions of x_c before the FFN, whose output is added to x_c only. A LayerNorm over $d + m$ followed by a restriction to x_c would make the scale of the FFN input depend on the magnitude of the memory, which is an undesirable behavior. A layer therefore computes

$$\begin{aligned} x &\leftarrow x + \mathrm{TAttn}\big(\mathrm{LN}_{d+m}(x)\big), \\ x &\leftarrow x + \mathrm{IAttn}\big(\mathrm{LN}_{d+m}(x)\big), \\ x_c &\leftarrow x_c + \mathrm{FFN}\big(\mathrm{LN}_d(x_c)\big), \qquad x_m \text{ unchanged.} \end{aligned}$$

The final LayerNorm and the output head are applied to x_c only.

Stacking internal layers An internal attention followed by an FFN can be seen as an internal layer. Since it is almost a transformer itself, it is natural to stack them, to increase depth. We call the number of stacked internal layers the internal depth, which is 1 by default. Increasing the internal depth increases the compute cost, but not the KV cache.

Remark 2 (Application to MoE)**.** *This architecture can be used to extend what experts can do in MoE architectures. In a traditional MoE, only the FFN can change. Here, an expert can be an internal attention followed by the FFN, or even multiple layers of internal attention + FFN. This allows for much more powerful experts, potentially doing some complicated reasoning.*

2.2 Parameters

Table 1 compares the parameters of one internal attention transformer layer with those of a standard transformer layer of width d, i.e. with the same KV cache.

The internal attention is cheap: its projections are $c \times c$ matrices shared across the k chunks. The main overhead comes from the temporal attention, whose input and output projections span the full width $d + m$.

3 Experiments

We evaluate the architecture on two autoregressive language modeling tasks. The first uses the TinyStories dataset (Eldan & Li, 2023). The second is the multiplication of two integers, a task

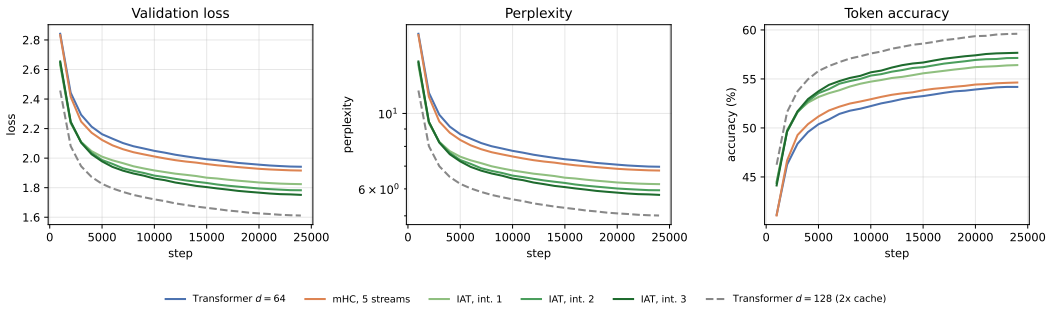

Figure 3: Validation loss, perplexity, and token accuracy over training for the runs of Table 2.

known to be difficult for transformer-based language models (Dziri et al., 2023; Deng et al., 2024). The arithmetic experiment is motivated by the fact that computing a product usually requires intermediate values, making memory valuable. The TinyStories experiment, on the other hand, shows that the architecture improves results even when intermediate values are not explicitly required.

We compare the performance of our architecture with a standard transformer and mHC. Furthermore, the training of the multiplication experiment turns out to be very unstable, requiring multiple seeds to get an accurate reading. Therefore the study of the impact of the different parameters of our architecture is done on the TinyStories dataset only.

3.1 TinyStories

The TinyStories dataset (Eldan & Li, 2023) is composed of short children's stories generated by GPT-3.5 and GPT-4, and has been used to train and benchmark small language models. We use TinyStories V2, a version of the training split regenerated entirely with GPT-4 to fix grammatical and logical errors present in the original GPT-3.5/GPT-4 mix.

Dataset We take all stories in the training and validation datasets. Stories are tokenized with the GPT-Neo encoding, restricted to its 10,000 most frequent tokens, then separated into non-overlapping blocks of 512 tokens.

Architectures For all architectures, we fix the depth (or external depth) at 4 and the width of the token representation at $d = 64$. The dimension d is chosen low because models with too high d are limited by the dataset rather than the model. For the baseline configuration of our architecture, we chose $m = 256$, $c = 16$ with 2 attention heads for the internal attention, and internal depth 1. For mHC, we use 5 streams, matching the overall dimension of our model.

Training parameters We train for 24,000 steps with batch size 64 and context length 512. We use AdamW ($\beta = (0.9, 0.95)$, weight decay 0.1), with a peak learning rate of 10^{-3}, 500 steps of linear warmup, and cosine decay to a floor of 10^{-4}. Gradient clipping is applied at norm 1. Each configuration is trained with a single seed.

Results Table 2 shows that our architecture outperforms both the standard transformer and mHC. It also shows that increasing the internal depth improves results. Figure 3 shows the corresponding validation curves over training; the ranking between architectures is stable throughout training. While our architecture clearly performs better than the standard transformer at equal KV cache, the question remains whether the gains, especially over mHC, are due to an increase in the parameter count. The next paragraph shows that they are not.

Parameter efficiency and the effect of m and c Starting from our baseline model, we did two sweeps in addition to increasing the internal depth. Sweeps are done starting from the baseline IAT ($d = 64$, $m = 256$, $c = 16$, internal depth 1) and change only one parameter at a time. We try $m = 0, 32, 64, 128, 256, 512$, and note that $m = 0$ is not the standard transformer as it adds internal attention. For c we try $c = 8, 16, 32, 64$, as we keep $c \leq d$. Detailed tables are available in

Model	Val. loss	Perplexity	Token acc. (%)	Params (M)	KV cache
Transformer $d = 64$	1.942	6.97	54.2	1.51	512
mHC, 5 streams $\times$ 64	1.916	6.79	54.6	1.61	512
IAT $d = 64$, $m = 256$, int. 1	**1.824**	**6.20**	**56.4**	1.79	512
IAT $d = 64$, $m = 256$, int. 2	1.782	5.94	57.1	1.93	512
IAT $d = 64$, $m = 256$, int. 3	1.751	5.76	57.7	2.07	512
Transformer $d = 128$	1.612	5.01	59.6	3.42	1024

Table 2: TinyStories v2, 24,000-step training run. KV cache in scalars per token. IAT is the Internal Attention Transformer and "int." is the internal depth.

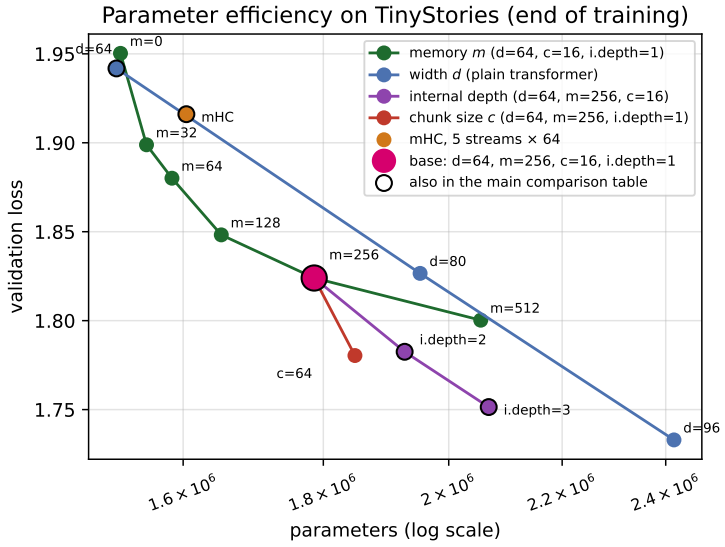

Figure 4: Final validation loss (step 24,000) versus parameter count (log scale). The large pink point denotes the baseline IAT configuration ($d = 64$, $m = 256$, $c = 16$, internal depth 1); the memory, internal-depth, and chunk-size series start from it, each changing only a single parameter from the baseline. Black contours indicate points that appear in Table 2.

Appendices C and D, but the overall effect can be summarized in Figure 4. From this figure we can draw several conclusions:

1. our architecture is more parameter efficient than the standard transformer and mHC,
2. increasing the internal depth does not seem to modify parameter efficiency,
3. in this case, the optimal value for c seems to be $c = d$,
4. adding internal attention with no memory ($m = 0$) decreases parameter efficiency compared to standard transformers,
5. the sweet spot in parameter efficiency is $m = 128$, and the parameter efficiency starts to be worse than that of the standard transformer at $m = 512$.

3.2 Arithmetic

As noted before, learning how to multiply numbers is a difficult problem for LLMs. We believe the possibility to store intermediate values should help in this task. To test this, we need a representation width d low enough that a standard transformer does not have enough memory capacity. That can be done by either taking a high number of digits, or taking a small d. Considering that the training runs are very unstable, we chose a low number of digits, paired with a small d.

Model	Params	lr	Overall	2×2	2×3	3×2	3×3
Transformer $d = 16$	17,120	3e-3	0.325	0.458	0.291	0.442	0.121
mHC, 5 streams × 16	45,906	3e-3	0.463	0.718	0.526	0.505	0.134
IAT $d = 16$, $m = 64$	45,200	1e-3	**0.572**	**0.754**	**0.604**	**0.655**	**0.283**

Table 3: Exact match on the full validation set (5,000 pairs) by operand lengths at the end of training (step 500k). For each architecture, the run shown is the one with the highest final exact match as measured during training, on the first 1,000 validation pairs.

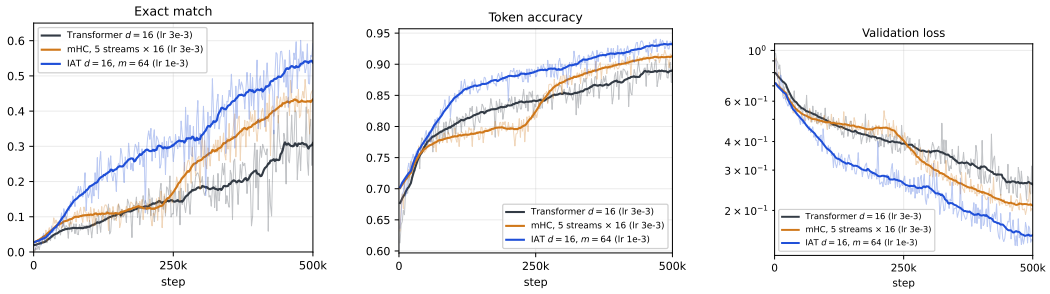

Figure 5: Best run for each architecture. The thin lines show the raw validation metric (measured every 2k steps), while the thick lines of the same colour show a centred moving average over 21 evaluations (42k steps).

Dataset For simplicity, we look at multiplications between numbers of 2 or 3 digits. We generate both a training set of 500,000 multiplications $a \times b$ and a validation set. For both a and b the number of digits is chosen uniformly in $\{2, 3\}$, and the numbers are then drawn uniformly in 10–99 or 100–999. The validation set is well balanced between the number of digits, but since we do not select duplicated pairs, the training set distribution is 67% of 3×3 pairs, 16% of 2×3, 16% of 3×2 and 1.4% of 2×2. We also ensure that the validation set is disjoint from the training set. We use a custom tokenizer containing the ten digits, '*' and '=', plus padding and end-of-sequence tokens.

Architectures We chose the representation width $d = 16$ for all architectures. mHC uses 5 streams, while our model is augmented by a memory of size $m = 64$, $c = 16$ with 2 attention heads for the internal attention, matching the overall dimension of mHC.

Training parameters We compare the models on a fixed number of 500,000 iterations, using a batch size of 256. The optimizer is AdamW with $\beta = (0.9, 0.95)$. We use 200 steps of warmup, and do gradient clipping at norm 1. Runs are done in FP32. We noticed a high variability, hence we run each configuration with 3 different seeds, and we sweep 3 different learning rates: 3e-4, 1e-3, 3e-3, for a total of 9 runs per architecture.

Results Table 3 shows results for the best run of each architecture, by class. We can see a clear ranking, with our architecture having a clear lead, followed by mHC, and finally the standard transformer. In particular, our architecture significantly outperforms the two others for 3×3 multiplications. Figure 5 shows the training trajectory of the best run of each architecture. Looking at the thin lines, which show raw values without the moving average, it is clear that the variability, even in a single run, is high. In addition, all runs are displayed per architecture in Figure 6, which once again emphasizes the variability, this time across seeds.

4 Conclusion

This paper introduced a new variation of the transformer architecture by explicitly adding a memory component to the token representation which is excluded from the FFN computation. Memory and compute are mixed through an internal attention mechanism, and an internal attention followed by an FFN forms an internal layer that can be stacked, adding depth without adding to the KV cache.

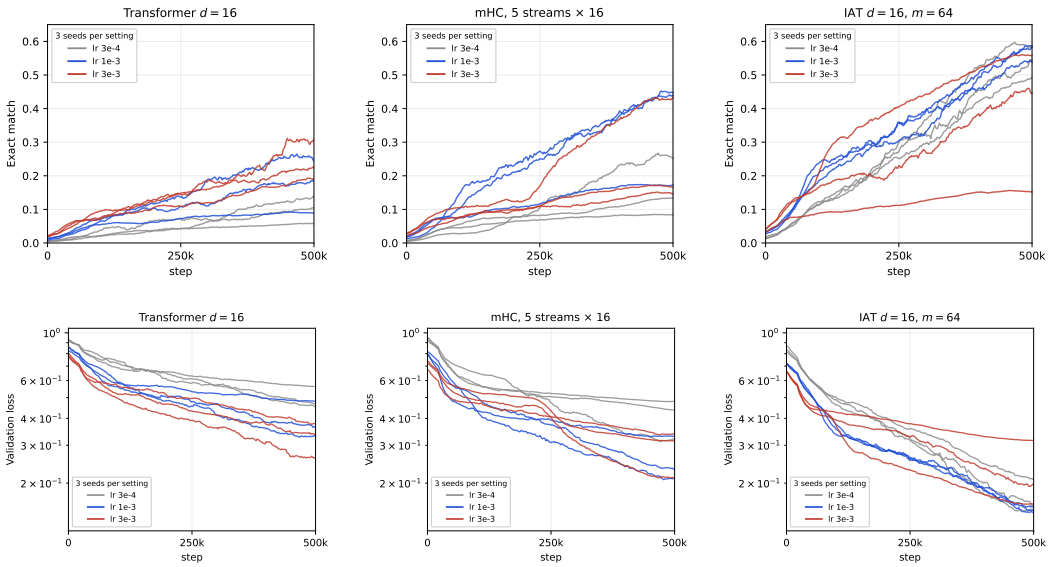

Figure 6: Run-to-run variability on the multiplication task. Each panel shows all nine runs of one architecture. Top: exact match; bottom: validation loss. Curves are moving averages over 21 evaluations (42k steps).

Overall, the experiments, while small scale, are promising and show that, at equal KV cache, it outperforms both the standard transformer and mHC on two very different tasks: language modeling and multiplication. In particular, we show that our architecture is more parameter efficient on TinyStories.

Finally, we think this work opens the door to richer Mixture of Experts models: instead of a single FFN, an expert could consist of several internal layers, allowing it to carry out multi-step computations on its own.

Reproducibility statement

We tried to include the relevant parameters used for training so that the experiments can be reproduced. The code will be available on GitHub after the review period has passed.

AI use statement

In this work, we used generative AI tools for implementing the architectures and experiments in PyTorch. We have not used generative AI tools for the design of the original architecture, and the rest of the required disclosure tasks are not applicable to this work.

Additionally, we used generative AI tools for creating and modifying figures, editing code, summarizing or analyzing existing literature, identifying relevant literature, formatting references, and editing the paper to improve readability. We have reviewed all AI-assisted work, in particular the code of the implementation of the different models. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

References

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020. URL https://arxiv.org/abs/2004.05150.

Vincent-Pierre Berges, Barlas Oğuz, Daniel Haziza, Wen tau Yih, Luke Zettlemoyer, and Gargi Ghosh. Memory layers at scale, 2024. URL https://arxiv.org/abs/2412.09764.

Srinadh Bhojanapalli, Chulhee Yun, Ankit Singh Rawat, Sashank J. Reddi, and Sanjiv Kumar. Low-rank bottleneck in multi-head attention models, 2020. URL https://arxiv.org/abs/2002.07028. ICML 2020.

Aydar Bulatov, Yuri Kuratov, and Mikhail S. Burtsev. Recurrent memory transformer, 2022. URL https://arxiv.org/abs/2207.06881. NeurIPS 2022.

Mikhail S. Burtsev, Yuri Kuratov, Anton Peganov, and Grigory V. Sapunov. Memory transformer, 2021. URL https://arxiv.org/abs/2006.11527.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts, 2023. URL https://arxiv.org/abs/2305.14788. EMNLP 2023.

Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. Vision transformers need registers, 2024. URL https://arxiv.org/abs/2309.16588. ICLR 2024.

DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Yang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jin Chen, Jingyang Yuan, Junjie Qiu, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruizhe Pan, Runxin Xu, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Size Zheng, T. Wang, Tian Pei, Tian Yuan, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Liu, Xin Xie, Xingkai Yu, Xinnan Song, Xinyi Zhou, Xinyu Yang, Xuan Lu, Xuecheng Su, Y. Wu, Y. K. Li, Y. X. Wei, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Zheng, Yichao Zhang, Yiliang Xiong, Yilong Zhao, Ying He, Ying Tang, Yishi Piao, Yixin Dong, Yixuan Tan, Yiyuan Liu, Yongji Wang, Yongqiang Guo, Yuchen Zhu, Yuduan Wang, Yuheng Zou, Yukun Zha, Yunxian Ma, Yuting Yan, Yuxiang You, Yuxuan Liu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhewen Hao, Zhihong Shao, Zhiniu Wen, Zhipeng Xu, Zhongyu Zhang, Zhuoshu Li, Zihan Wang, Zihui Gu, Zilin Li, and Ziwei Xie. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024. URL https://arxiv.org/abs/2405.04434.

Yuntian Deng, Yejin Choi, and Stuart Shieber. From explicit cot to implicit cot: Learning to internalize cot step by step, 2024. URL https://arxiv.org/abs/2405.14838.

Mingyu Ding, Bin Xiao, Noel Codella, Ping Luo, Jingdong Wang, and Lu Yuan. Davit: Dual attention vision transformers, 2022. URL https://arxiv.org/abs/2204.03645. ECCV 2022.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and fate: Limits of transformers on compositionality, 2023. URL https://arxiv.org/abs/2305.18654. NeurIPS 2023.

Alaaeldin El-Nouby, Hugo Touvron, Mathilde Caron, Piotr Bojanowski, Matthijs Douze, Armand Joulin, Ivan Laptev, Natalia Neverova, Gabriel Synnaeve, Jakob Verbeek, and Hervé Jegou. Xcit: Cross-covariance image transformers, 2021. URL https://arxiv.org/abs/2106.09681. NeurIPS 2021.

Ronen Eldan and Yuanzhi Li. Tinystories: How small can language models be and still speak coherent english?, 2023. URL https://arxiv.org/abs/2305.07759.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. Transformer Circuits Thread, 2021. URL https://transformer-circuits.pub/2021/framework/index.html.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories, 2021. URL https://arxiv.org/abs/2012.14913. EMNLP 2021.

Kai Han, An Xiao, Enhua Wu, Jianyuan Guo, Chunjing Xu, and Yunhe Wang. Transformer in transformer, 2021. URL https://arxiv.org/abs/2103.00112. NeurIPS 2021.

Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large memory layers with product keys, 2019. URL https://arxiv.org/abs/1907.05242. NeurIPS 2019.

Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. itransformer: Inverted transformers are effective for time series forecasting, 2024. URL https://arxiv.org/abs/2310.06625. ICLR 2024.

Amirkeivan Mohtashami and Martin Jaggi. Landmark attention: Random-access infinite context length for transformers, 2023. URL https://arxiv.org/abs/2305.16300. NeurIPS 2023.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling, 2019. URL https://arxiv.org/abs/1911.05507. ICLR 2020.

Sainbayar Sukhbaatar, Edouard Grave, Guillaume Lample, Herve Jegou, and Armand Joulin. Augmenting self-attention with persistent memory, 2019. URL https://arxiv.org/abs/1907.01470.

Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. Mlp-mixer: An all-mlp architecture for vision, 2021. URL https://arxiv.org/abs/2105.01601. NeurIPS 2021.

Haiyang Wang, Yue Fan, Muhammad Ferjad Naeem, Yongqin Xian, Jan Eric Lenssen, Liwei Wang, Federico Tombari, and Bernt Schiele. Tokenformer: Rethinking transformer scaling with tokenized model parameters, 2025. URL https://arxiv.org/abs/2410.23168. ICLR 2025.

Zhenda Xie, Yixuan Wei, Huanqi Cao, Chenggang Zhao, Chengqi Deng, Jiashi Li, Damai Dai, Huazuo Gao, Jiang Chang, Kuai Yu, Liang Zhao, Shangyan Zhou, Zhean Xu, Zhengyan Zhang, Wangding Zeng, Shengding Hu, Yuqing Wang, Jingyang Yuan, Lean Wang, and Wenfeng Liang. mhc: Manifold-constrained hyper-connections, 2025. URL https://arxiv.org/abs/2512.24880.

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. Native sparse attention: Hardware-aligned and natively trainable sparse attention, 2025. URL https://arxiv.org/abs/2502.11089. ACL 2025.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences, 2021. URL https://arxiv.org/abs/2007.14062. NeurIPS 2020.

Shuai Zhang, Peng Zhang, Xindian Ma, Junqiu Wei, Ningning Wang, and Qun Liu. Tensorcoder: Dimension-wise attention via tensor representation for natural language modeling, 2020. URL https://arxiv.org/abs/2008.01547.

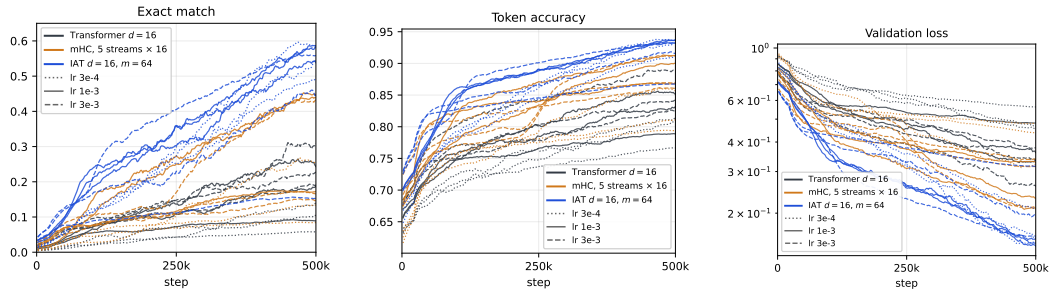

Figure 7: All 27 multiplication runs (three architectures × three learning rates × three seeds), at equal KV cache (160 scalars per token). Colour encodes the architecture, line style the learning rate; the three curves sharing a colour and a style are three seeds. Curves are moving averages over 21 evaluations (42k steps).

Defa Zhu, Hongzhi Huang, Zihao Huang, Yutao Zeng, Yunyao Mao, Banggu Wu, Qiyang Min, and Xun Zhou. Hyper-connections, 2025a. URL https://arxiv.org/abs/2409.19606. ICLR 2025.

Defa Zhu, Hongzhi Huang, Jundong Zhou, Zihao Huang, Yutao Zeng, Banggu Wu, Qiyang Min, and Xun Zhou. Frac-connections: Fractional extension of hyper-connections, 2025b. URL https://arxiv.org/abs/2503.14125.

APPENDIX

A EXTRA RESULTS ON THE ARITHMETIC EXPERIMENT

See Figure 7 for all runs in the same plot.

B TEMPORAL ATTENTION RESTRICTED TO THE COMPUTE PART

We compared the low-rank temporal attention with a variant that attends to x_c only, the memory being then reachable exclusively through the internal attention. On TinyStories, with the same protocol as Table 2 ($d = 64$, $m = 256$, $c = 16$), the restricted variant is worse than a standard transformer with the same KV cache, while the full-width variant is clearly better:

Model	Val. loss	Perplexity	Token acc. (%)
Transformer $d = 64$	1.942	6.97	54.2
Temporal attention on x_c only ($m = 256$)	1.955	7.06	54.0
Low-rank temporal attention ($m = 256$)	1.824	6.20	56.4

Table 4: TinyStories, validation at the end of the same 24,000-step runs as Table 2.

C INFLUENCE OF THE MEMORY SIZE m

We swept the memory size $m \in \{0, 32, 64, 128, 256, 512\}$ on TinyStories, keeping $d = 64$, $c = 16$, internal depth 1, and the rest of the protocol identical to Table 2. Internal attention with nothing to mix ($m = 0$) is very slightly worse than the plain transformer (1.950 vs. 1.942) despite a handful more parameters, but validation loss then decreases monotonically as m grows, with diminishing but still clearly positive returns even at $m = 512$ (Figure 8).

Model	Val. loss	Perplexity	Token acc. (%)	Params (M)	KV cache
Transformer $d = 64$ (no internal attention)	1.942	6.97	54.2	1.51	512
IAT $d = 64$, $m = 0$, $c = 16$	1.950	7.03	54.1	1.52	512
IAT $d = 64$, $m = 32$, $c = 16$	1.899	6.68	55.0	1.55	512
IAT $d = 64$, $m = 64$, $c = 16$	1.880	6.55	55.3	1.59	512
IAT $d = 64$, $m = 128$, $c = 16$	1.848	6.35	56.0	1.65	512
IAT $d = 64$, $m = 256$, $c = 16$	1.824	6.20	56.4	1.79	512
IAT $d = 64$, $m = 512$, $c = 16$	1.800	6.05	56.9	2.05	512

Table 5: TinyStories v2, validation at the end of a 24,000-step training run, with internal depth 1. The $m = 256$ row is the same run as "IAT $d = 64$, $m = 256$, int. 1" in Table 2, and the plain transformer row is the same run as "Transformer $d = 64$" there.

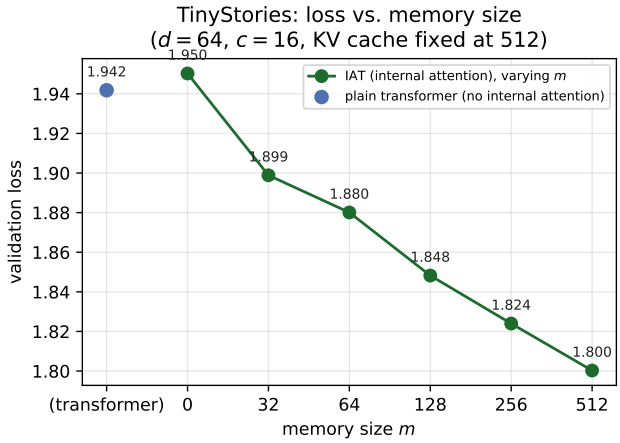

Figure 8: Final validation loss (step 24,000) as a function of the memory size m, for the runs in Table 5.

D Influence of the internal chunk size c

We swept the internal chunk size $c \in \{8, 16, 32, 64\}$ on TinyStories, keeping $d = 64$, $m = 256$, internal depth 1, and the rest of the protocol identical to Table 2. Validation loss decreases monotonically as c grows over this range, with a marked improvement at $c = 64$ (Figure 9).

Model	Val. loss	Perplexity	Token acc. (%)	Params (M)	KV cache
IAT $d = 64$, $m = 256$, $c = 8$	1.836	6.27	56.2	1.78	512
IAT $d = 64$, $m = 256$, $c = 16$	1.824	6.20	56.4	1.79	512
IAT $d = 64$, $m = 256$, $c = 32$	1.816	6.15	56.5	1.80	512
IAT $d = 64$, $m = 256$, $c = 64$	1.780	5.93	57.1	1.85	512

Table 6: TinyStories, validation at the end of a 24,000-step training run, with internal depth 1. The $c = 16$ row is the same run as "IAT $d = 64$, $m = 256$, int. 1" in Table 2.

E Parameter efficiency: full range, including $d = 112$ and $d = 128$

Figure 10 reproduces Figure 4 from the main text with two more points on the width-d curve, $d = 112$ and $d = 128$ (2.91M and 3.42M parameters).

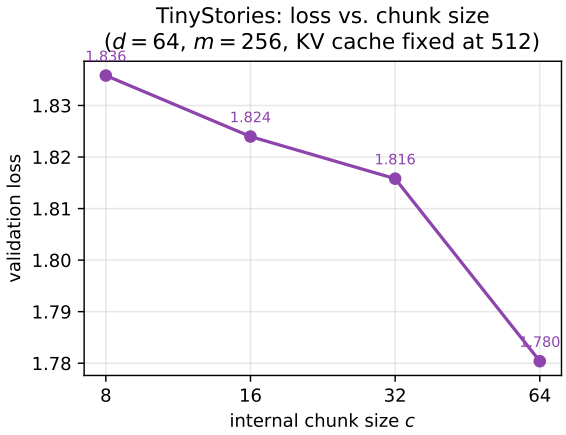

Figure 9: Final validation loss (step 24,000) as a function of the internal chunk size c (log scale), for the runs in Table 6. KV cache is constant across all four points.

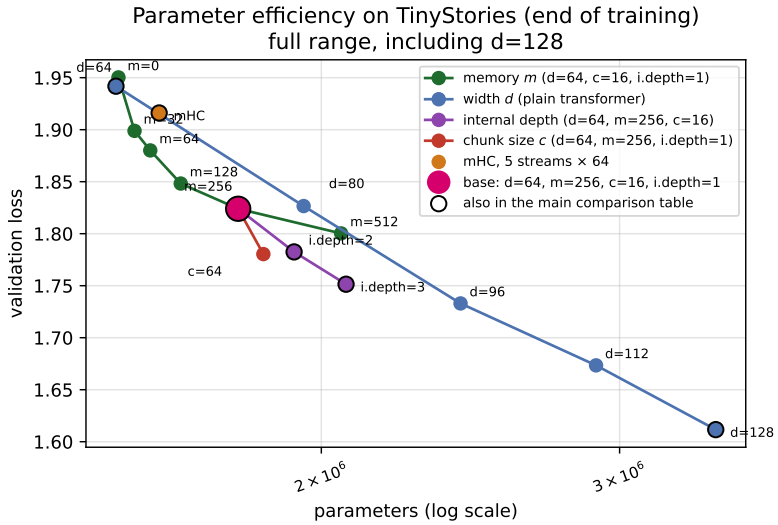

Figure 10: Same data as Figure 4, with $d = 112$ (2.91M parameters, KV cache 896) and $d = 128$ (3.42M parameters, KV cache 1024) added back to the width-d curve.