FlowHN: Adaptive Token Routing for Efficient Parallel Hybrid Networks

Mohammad Mahdi Moradi[1,2], Walid Ahmed[2], Shuangyue Wen[2]
Sudhir Mudur[1], Weiwei Zhang[2], Yang Liu[2]

[1]Department of Computer Science, Concordia University
[2]Huawei Technologies, Toronto Research Center
Correspondence: mohammad.mahdi.moradi@h-partners.com

Abstract

Production LLMs must balance modeling quality with predictable latency, stable accelerator utilization, and cost-efficient scaling—constraints that remain difficult for existing architectures. Transformers provide strong reasoning but incur quadratic complexity, while state-space models (SSMs) scale efficiently yet lack fine-grained interactions; prior hybrids either introduce sequential bottlenecks or rely on learned routing that complicates deployment. We present FlowHN, a deployment-oriented parallel hybrid architecture that enables deterministic conditional computation via FLOP-aware token circulation across attention and SSM branches. Instead of dynamic expert routing, FlowHN performs hardware-aligned token scheduling that balances workloads, reduces synchronization stalls, and preserves full parameter utilization. Across 135M–1B models, FlowHN achieves up to 4× higher throughput and 15% higher MFU than strong Transformer, SSM, and hybrid baselines while maintaining competitive accuracy on reasoning, coding, and long-context tasks up to 32K tokens. FlowHN is designed to integrate directly into existing Hybrid pipelines without changes to optimizers, training stacks, or inference serving infrastructure, making it practical for real-world deployment.

1 Introduction

Transformer-based language models underpin many production NLP systems, yet their quadratic complexity with respect to sequence length introduces substantial memory, latency, and cost challenges in real-world deployments (Zeng et al., 2025). To address these limitations, researchers have explored more scalable alternatives such as Linear Attention (Katharopoulos et al., 2020), Gated Linear Attention (Yang et al., 2023), and, more recently, State Space Models (SSMs) (Fu et al., 2022; Gu and Dao, 2023). SSMs provide hardware-friendly implementations and linear-time recurrence, enabling efficient long-range modeling. However, despite their favorable scaling properties, SSM-based models consistently lag behind Transformers on tasks requiring precise token-level interactions and in-context retrieval.

This complementary trade-off between efficiency (SSMs) and expressivity (Transformers) has motivated hybrid architectures that combine both mechanisms (Lieber et al., 2024; Ren et al., 2024; Glorioso et al., 2024). Sequential hybrids interleave attention and SSM layers in series, but their sequential dependencies introduce idle periods that reduce throughput and increase latency. Parallel hybrids attempt to address this limitation by processing tokens concurrently through attention and SSM branches, eliminating idle computation. However, they introduce new challenges, including workload imbalance between branches with different FLOP profiles and synchronization latency when faster branches must wait for slower ones before fusion.

To date, Hymba (Dong et al., 2024) represents one of the most prominent parallel hybrid designs, applying both branches to every token and averaging their outputs. While effective, its static allocation and uniform dual-branch processing limit efficiency gains and exacerbate branch imbalance, highlighting the need for more hardware-aware scheduling strategies.

In large-scale enterprise deployments, runtime variance, synchronization stalls, and uneven compute utilization often dominate infrastructure cost. Architectures that rely on stochastic routing or dynamically activated experts can introduce unpredictable latency profiles and workload imbalance, complicating scheduling, capacity planning, and accelerator provisioning. Consequently, efficiency improvements must be predictable, stable, and aligned with hardware characteristics. Learned routing mechanisms such as mixture-of-experts (Zhang et al., 2025) provide conditional compu-

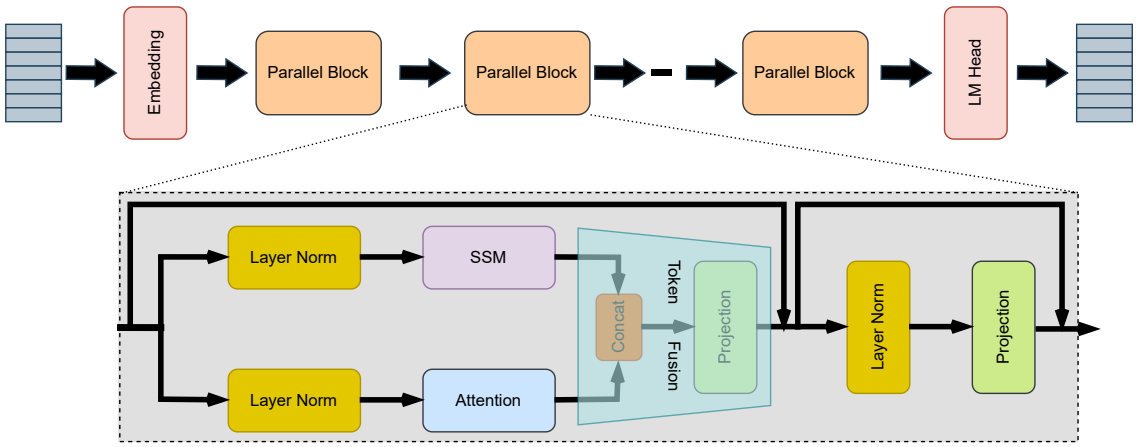

Figure 1: The overall architecture of our proposed parallel hybrid model, FlowHN

tation but frequently introduce non-deterministic execution behavior. These practical constraints motivate architectures that enable conditional computation while preserving deterministic execution, balanced workloads, and full parameter utilization. FlowHN is designed from this deployment-driven perspective rather than purely algorithmic optimization.

We introduce FlowHN, a FLOP-aware parallel hybrid network that integrates attention and SSM branches through deterministic token circulation. Instead of dynamic expert selection, FlowHN performs hardware-aligned token scheduling that balances branch workloads, minimizes synchronization stalls, and preserves progressive token coverage across layers. Designed as a deployment-oriented architecture, FlowHN emphasizes predictable latency, stable utilization, and drop-in compatibility with hybrid block, achieving strong efficiency–accuracy trade-offs across multiple model scales and long-context benchmarks.

In summary, our contributions are:

- We propose FlowHN, a hybrid attention–SSM architecture with deterministic, FLOP-aware conditional computation designed for stable and predictable deployment.
- We introduce FLOP-Aware Circulating Split (FAC-Split), a deterministic token splitting and rotation scheme that balances computation across parallel branches.
- We show that progressive token circulation ensures full token coverage across layers, mitigating information loss introduced by partial computation.
- We achieve the best efficiency–accuracy trade-off among evaluated baselines under identical compute budgets.

FlowHN is well-suited for long-context production workloads such as document understanding, compliance analysis, and large-scale code intelligence, where predictable latency and sustained throughput are critical operational requirements.

2 Proposed Method

Figure 1 shows the overall FlowHN architecture, composed of parallel hybrid blocks that combine Attention and SSM branches. Tokens are deterministically allocated between branches to balance FLOP usage while preserving representational coverage. An illustrative comparison of allocation strategies is included in Appendix A (Figure 5). After parallel processing, outputs are fused through a lightweight projection layer.

2.1 No_Split

In the No_Split configuration, the full token sequence is processed by both branches at every block. This maximizes cross-branch information exchange and typically yields the highest accuracy, but incurs higher computational cost. The fused output is obtained by concatenating branch representations and projecting them through a learnable matrix:

$$Y = W[H_{attn}; H_{ssm}], \tag{1}$$

which integrates local (attention) and global (SSM) dependencies at the cost of redundant FLOPs.

2.2 Alternating_Equal_Split (AE_Split)

AE_Split divides tokens evenly between the attention and SSM branches within each block, reducing per-block computation and improving throughput. To maintain information coverage, token as-

signments alternate deterministically across consecutive blocks: tokens processed by the attention branch in block i are routed to the SSM branch in block $i+1$, and vice versa. This strategy ensures eventual exposure of all tokens to both branches, though synchronization latency may persist when branch runtimes differ.

2.3 FLOP_Aware_Split (FA_Split)

FA_Split allocates tokens to each branch in proportion to its estimated FLOPs-per-token cost. The computationally heavier branch receives fewer tokens, while the lighter branch processes a larger subset, yielding improved runtime balance and hardware utilization compared with AE_Split. However, because token assignments remain fixed across layers, each token is processed by only a single branch, which can limit cross-branch interaction and representational expressivity.

2.4 FLOP_Aware_Circulating_Split (FAC_Split)

FAC_Split extends FA_Split by introducing deterministic circulation of token assignments across successive parallel blocks. Tokens are initially allocated based on relative branch FLOP costs, but their assignments are rotated across layers so that every token is processed by both the attention and SSM branches over depth. This progressive circulation preserves the efficiency benefits of FLOP-aware allocation while ensuring full representational coverage and mitigating information loss. As a result, FAC_Split combines the computational efficiency of FA_Split with the representational richness of No_Split, achieving balanced workloads, high throughput, and stable accuracy without increasing overall FLOPs. A formal specification and illustrative examples of FAC_Split are provided in Appendix A.

2.5 Token Fusion

After both branches complete processing, their hidden states are concatenated and projected back to the model dimension:

$$Y = W_f[H_{attn}; H_{ssm}], \tag{2}$$

where W_f is a learnable fusion matrix. This design efficiently merges representations while keeping parameter overhead minimal. Empirically, concatenation-projection achieves a stronger efficiency–accuracy trade-off than other linear fusion alternatives.

2.6 Deployment-Oriented Conditional Computation

FlowHN performs deterministic conditional computation through FLOP-aware token allocation across attention and SSM branches. Unlike mixture-of-experts approaches that rely on learned routing, FlowHN schedules computation using fixed, hardware-aligned policies, enabling predictable execution behavior while preserving full parameter utilization. Tokens are distributed according to branch FLOP cost and circulated across layers to maintain representational coverage without introducing stochastic latency variance.

From a deployment perspective, this design provides several advantages. First, deterministic routing ensures stable latency profiles, simplifying scheduling and capacity planning in production systems. Second, balanced token allocation reduces synchronization stalls between parallel branches, sustaining high model FLOPs utilization (MFU). Third, because all parameters remain active during training and inference, FlowHN avoids the memory fragmentation and load imbalance often associated with sparse expert activation. Finally, the architecture operates as a drop-in hybrid block, requiring no modification to optimizers, training pipelines, or serving infrastructure.

Overall, FlowHN reframes conditional computation as a hardware-aware scheduling problem, enabling efficient parallel execution while maintaining compatibility with existing deployment stacks. A formal analysis of stability under deterministic circulation is provided in Appendix B.

3 Results and Discussion

Table 1 presents the performance of FlowHN with the four token allocation strategies and three model sizes (135M, 350M, and 1B parameters), compared against LLaMA, Mamba-1, Mamba-2, Jamba, and Hymba. The table reports accuracy, throughput measured in Tokens per Second (TPS), and Model FLOPs utilization (MFU). MFU (Chowdhery et al., 2023) quantifies how effectively available accelerator compute (e.g., GPUs or TPUs) is utilized during training and inference, with higher values indicating reduced idle capacity and better hardware efficiency. Across all evaluated model scales, FlowHN consistently achieves higher MFU than both attention-only and hybrid baselines, reflecting more effective utilization of accelerator compute through balanced parallel execution—an im-

portant consideration in cost-sensitive and latency-constrained production environments.

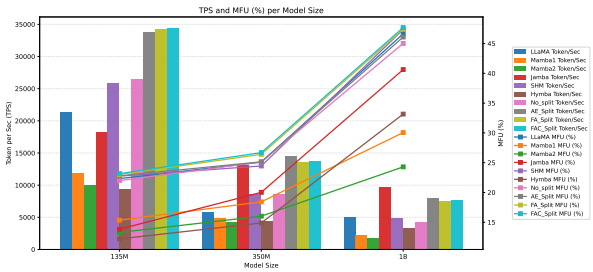

Figure 2: Effect of model scale on TPS and MFU. Bars (left axis) show TPS, while lines (right axis) trace MFU for each model at 135M, 350M, and 1B parameters.

Table 1 and Figure 2 highlight the trade-offs between efficiency and accuracy across state-of-the-art baselines and our FlowHN variants at 135M, 350M, and 1B parameters. No_Split attains the highest accuracy but at the cost of efficiency, while AE_Split and FA_Split improve TPS and MFU over baselines yet remain below FAC_Split. In contrast, FAC_Split consistently provides the best overall balance, reaching 34K TPS at 135M, 14K TPS at 350M, and 7.9K TPS at 1B, with competitive accuracy and the highest MFU at all scales. The only throughput exception is Jamba-1B, whose Mixture-of-Experts design activates only two of eight experts per token, reducing active parameters to 483M and boosting speed, though at the cost of lower MFU. As shown in Figure 3, FAC_Split also exhibits a steeper accuracy scaling trend than hybrid baselines such as Hymba and Jamba, continuing to improve with model size rather than saturating, indicating more effective utilization of additional capacity. At larger scales, FlowHN's throughput gains translate into clear superiority in both MFU and speed. Overall, FAC_Split establishes the most favorable efficiency–accuracy frontier across model sizes (Figures 2 and 3).

3.1 Information Loss Analysis

We evaluate information loss (KL to No_Split) on a FlowHN with five Attention–SSM blocks (Fig. 4). FA_Split plateaus/oscillates because tokens are permanently pinned to a single branch, preventing cross-branch information exchange. AE_Split decreases KL monotonically but with diminishing returns: its alternating 50/50 routing is task-agnostic and exposes tokens to the other branch only every second block, slowing error transport. FAC_Split contracts fastest and deepest by combining per-

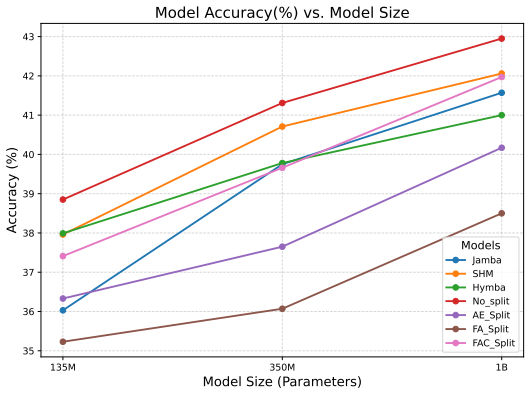

Figure 3: Accuracy as a function of model scale (135 M, 350 M, and 1 B parameters) comparing our approach against state-of-the-art hybrid models.

layer rotation (immediate cross-branch exposure) with FLOP-aware token allocation matched to task demands (more attention for local structure, more SSM for long-range coherence), yielding the lowest final-block KL (0.11) and the strongest alignment at the logits.

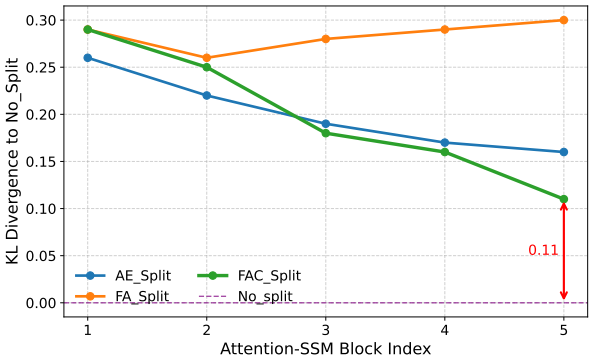

Figure 4: Information Loss analysis.

3.2 Long-Context Deployment Evaluation

The results in Table 2 illustrate the complementary strengths of attention and SSMs under a unified training paradigm (Appendix C.2). LLaMA-1B achieves peak accuracy at 8K (34.5%) but scales poorly, as quadratic complexity drives throughput down to 822 TPS at 32K with MFU saturating near 48%. Mamba2-1B sustains 2K TPS and rising MFU (35–41%) due to linear recurrence but trails attention by 2–3 points in accuracy across tasks. Jamba-1B narrows this gap through sequential hybridization, reaching maximum accuracy, though dependencies cap MFU below 47%, while Hymba-1B parallelizes branches but incurs synchronization latency and redundant compute, yielding both lower throughput and weaker accuracy. In contrast, FAC_Split-1B consistently se-

Table 1: Comparative analysis of TPS, MFU, and accuracy across various state-of-the-art models, including our proposed FlowHN, evaluated at three model scales: 135M, 350M, and 1B parameters.

Model Name	Tokens	TPS	MFU	arc_e	arc_c	wino	piqa	hella	boolq	Avg
135M										
Llama	1B	21353	22.3	27.2	22.9	47.9	52.4	25.1	41.2	36.1
Mamba-1	1B	11854	15.3	26.8	24.6	49.5	53.0	26.0	40.9	36.8
Mamba-2	1B	10094	13.2	28.1	23.8	48.5	53.1	25.4	41.0	36.7
Jamba	1B	18324	13.8	27.7	22.4	37.8	25.7	53.0	49.3	36.0
SHM	1B	25900	22.7	29.6	22.7	40.7	26.5	55.2	50.6	37.9
Hymba	1B	9393	12.2	29.8	23.8	49.0	55.0	25.5	44.5	37.9
No_split$_{ours}$	1B	26443	21.9	29.4	24.2	49.9	56.0	27.0	43.4	**38.8**
AE_Split$_{ours}$	1B	33846	22.8	26.2	23.8	48.6	52.5	25.9	40.8	36.3
FA_Split$_{ours}$	1B	34348	23.0	25.2	23.9	47.8	50.5	25.2	38.4	35.2
FAC_Split$_{ours}$	1B	**34430**	**23.1**	28.7	23.2	48.9	54.3	26.8	42.3	37.4
350M										
Llama	1B	9223	25.0	29.1	23.4	50.1	54.6	26.41	48.3	38.3
Mamba-1	1B	4966	18.4	29.5	24.0	51.0	55.2	26.9	49.0	39.3
Mamba-2	1B	4264	16.0	29.8	23.7	50.6	54.9	26.7	48.7	38.9
Jamba	1B	13202	20.0	29.0	22.6	52.1	26.7	56.7	51.2	39.7
SHM	1B	8547	24.4	29.1	23.5	58.5	26.8	55.7	50.4	40.7
Hymba	1B	4452	14.8	30.5	24.4	49.4	54.2	26.1	53.8	39.7
No_split$_{ours}$	1B	8587	25.1	30.8	25.8	49.3	57.5	27.0	57.2	**41.3**
AE_Split$_{ours}$	1B	13697	25.1	27.4	23.4	49.5	55.4	25.7	44.3	37.6
FA_Split$_{ours}$	1B	14246	26.4	25.4	22.5	48.0	53.0	26.0	40.8	36.0
FAC_Split$_{ours}$	1B	**14385**	**26.6**	29.3	25.7	47.9	56.2	27.4	56.5	40.5
1B										
Llama	2B	5069	46.6	33.9	24.7	52.0	56.1	27.1	52.4	41.0
Mamba-1	2B	2189	30.0	34.2	25.0	52.9	56.8	27.4	53.0	41.6
Mamba-2	2B	1768	24.3	33.6	24.5	52.4	56.3	27.0	52.6	41.1
Jamba	2B	**9758**	40.6	34.6	23.2	49.5	58.2	26.9	56.7	41.5
SHM	2B	4923	47.3	33.5	23.8	49.7	55.4	27.4	56.2	42.0
Hymba	2B	3283	33.1	37.0	21.9	49.2	59.3	27.1	60.1	41.0
No_split$_{ours}$	2B	4325	45.0	36.1	22.9	49.5	59.3	28.2	61.4	**42.9**
AE_Split$_{ours}$	2B	7752	46.1	33.3	22.5	49.1	55.4	26.8	53.7	40.1
FA_Split$_{ours}$	2B	7901	47.4	32.0	23.4	49.4	54.1	25.8	46.1	38.5
FAC_Split$_{ours}$	2B	7926	**47.6**	36.0	23.8	50.9	57.5	26.5	56.7	41.9

Table 2: Comparison of Accuracy (%), Throughput, and MFU across long-context benchmarks—LongFQA (8K), NQ (16K), and CUAD (32K)—evaluated in a 3-shot setting on 8×V100 (32GB) GPUs.

Model	**LongFQA (8K)**			**NQ (16K)**			**CUAD (32K)**			**Average**		
	Acc	TPS	MFU	Acc	TPS	MFU	Acc	TPS	MFU	Acc	TPS	MFU
Mamba2-1B	32.7	2120	35.6	32.2	2044	38.2	29.2	1956	41.4	31.4	2040	38.4
LLaMA-1B	**34.5**	1949	45.1	33.1	1302	47.4	30.0	822	48.7	32.5	1358	47.1
Jamba-1B	34.2	2050	42.6	**33.9**	1763	44.4	**30.8**	1308	46.9	**33.0**	1707	44.6
Hymba-1B	33.0	1588	38.8	32.6	1263	40.5	29.4	1064	42.7	31.7	1305	40.7
FAC_Split-1B	34.0	**3932**	**51.3**	33.4	**3824**	**53.6**	30.6	**3588**	**55.4**	32.7	**3781**	**53.4**

cures the second-highest accuracy overall while delivering 2–3× higher throughput and 53.4% MFU, confirming that FLOP-aware routing and progressive token rotation effectively balance branch workloads and maintain representational coverage at scale. Moreover, the advantage widens with context length: from 8K→32K, FAC_Split's throughput benefit grows from 1.91× to 2.94× over Jamba and from 2.47× to 3.37× over Hymba. These findings demonstrate that while LLaMA excels at short-range reasoning and Mamba at efficiency, FAC_Split uniquely combines both properties, pre-

Table 3: Performance of FlowHN and baselines on math, coding, and summarization benchmarks: Accuracy (Acc) for GSM8K, HumanEval, MBPP, ROUGE-L (R-L) for GovReport and SQuALITY, along with TPS and MFU, at 1B scale.

Model	**GSM8K**			**HumanEval**			**MBPP**			**GovReport**			**SQuALITY**		
	Acc	TPS	MFU	Acc	TPS	MFU	Acc	TPS	MFU	R-L	TPS	MFU	R-L	TPS	MFU
Mamba2-1B	18.0	2124	18.8	17.6	2059	25.8	34.2	2115	21.3	18.4	1884	43.8	18.9	2026	37.4
LLaMA-1B	21.5	6843	40.8	**21.9**	6608	48.1	**37.2**	6340	43.6	19.3	1732	47.2	20.8	2598	46.6
Jamba-1B	21.8	9171	35.6	20.8	8731	34.8	36.6	8815	37.6	20.4	1822	43.5	**22.5**	2733	42.6
Hymba-1B	18.5	4430	28.2	18.9	4267	27.5	34.8	4111	30.2	19.5	1411	44.8	19.9	2177	40.9
FAC_Split-1B	**22.2**	**10712**	**42.6**	20.4	**9910**	**49.1**	35.7	**10338**	**44.6**	**21.1**	**3495**	**53.2**	21.5	**5242**	**52.7**

serving near-attention accuracy while scaling robustly to long contexts.

3.3 Task Generalization Across Reasoning, Coding, and Summarization

Table 3 summarizes accuracy, TPS, and MFU across math (GSM8K), coding (HumanEval, MBPP), and summarization (GovReport, SQuALITY) tasks, showing how task-specific token allocation affects performance. In math reasoning and coding tasks, where fine-grained symbolic manipulation and syntactic precision are critical, a larger proportion of tokens is routed to the attention branch—enabling higher computational efficiency without compromising accuracy. For summarization, where long-range coherence dominates, most tokens flow to the SSM branch, allowing FAC_Split-1B to attain comparable ROUGE-L scores while delivering over 2× the throughput of Jamba and Hymba. Compared to Mamba2's loss of fine-grained accuracy and LLaMA's quadratic inefficiency, FAC_Split's deterministic routing achieves the best balance between expressivity and scalability, adapting token flow per task to maximize utilization without sacrificing representational fidelity.

4 Conclusion

We presented FlowHN, a FLOP-aware parallel hybrid architecture that integrates attention and state-space mechanisms through deterministic token circulation and hardware-aligned scheduling. By balancing branch workloads and rotating token assignments across layers, FlowHN increases throughput and MFU while preserving strong performance on reasoning, coding, summarization, and long-context tasks. Unlike learned sparse routing approaches, FlowHN ensures predictable execution and full parameter utilization, making it well suited for latency-sensitive deployment. More broadly, our results suggest that viewing conditional computation as a deterministic scheduling problem provides a practical and infrastructure-compatible path toward efficient hybrid language models that can serve as drop-in components in real-world systems.

5 Limitations

While FlowHN demonstrates strong efficiency–accuracy trade-offs, several directions remain for future work. First, evaluating deterministic token circulation at larger model scales and in fully distributed serving environments would further validate its deployment potential. Second, the current FLOP-aware scheduling is based on static compute estimates; exploring adaptive yet deterministic allocation strategies could further improve task-specific efficiency. Finally, extending the proposed hybrid scheduling framework to broader heterogeneous architectures and multimodal settings represents a promising avenue for future research.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Mohammad Bavarian, Heewoo Jun, Nikolas Tezak, John Schulman, Christine McLeavey, Jerry Tworek, and Mark Chen. 2022. Efficient training of language models to fill in the middle. *arXiv preprint arXiv:2207.14255*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, and 1 others. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Xin Dong, Yonggan Fu, Shizhe Diao, Wonmin Byeon, Zijia Chen, Ameya Sunil Mahabaleshwarkar, Shih-Yang Liu, Matthijs Van Keirsbilck, Min-Hung Chen, Yoshi Suhara, and 1 others. 2024. Hymba: A hybrid-head architecture for small language models. *arXiv preprint arXiv:2411.13676*.

Daniel Y Fu, Tri Dao, Khaled K Saab, Armin W Thomas, Atri Rudra, and Christopher Ré. 2022. Hungry hungry hippos: Towards language modeling with state space models. *arXiv preprint arXiv:2212.14052*.

Paolo Glorioso, Quentin Anthony, Yury Tokpanov, and 1 others. 2024. Zamba: A compact 7b ssm hybrid model. *arXiv preprint arXiv:2405.XXXXX*.

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

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and 1 others. 2021. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. 2021. Efficient attentions for long document summarization. *arXiv preprint arXiv:2104.02112*.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 1909. Codesearchnet challenge: Evaluating the state of semantic code search.(2019). *arXiv preprint arXiv:1909.09436*.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR.

Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, and 1 others. 2024. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *arXiv preprint arXiv:1705.04146*.

Shen-Yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2021. A diverse corpus for evaluating and developing english math word problem solvers. *arXiv preprint arXiv:2106.15772*.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*.

Liliang Ren, Yang Liu, Yadong Lu, Yelong Shen, Chen Liang, and Weizhu Chen. 2024. Samba: Simple hybrid state space models for efficient unlimited context language modeling. *arXiv preprint arXiv:2406.07522*.

Eva Sharma, Chen Li, and Lu Wang. 2019. Bigpatent: A large-scale dataset for abstractive and coherent summarization. *arXiv preprint arXiv:1906.03741*.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. 2023. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama. https://www.cerebras.net/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama.

Alex Wang, Richard Yuanzhe Pang, Angelica Chen, Jason Phang, and Samuel R Bowman. 2022. Squality: Building a long-document summarization dataset the hard way. *arXiv preprint arXiv:2205.11465*.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. 2023. Gated linear attention transformers with hardware-efficient training. *arXiv preprint arXiv:2312.06635*.

Qiuhao Zeng, Jerry Huang, Peng Lu, Gezheng Xu, Boxing Chen, Charles Ling, and Boyu Wang. 2025. Zeta: Leveraging z-order curves for efficient top-k attention. *arXiv preprint arXiv:2501.14577*.

Danyang Zhang, Junhao Song, Ziqian Bi, Xinyuan Song, Yingfang Yuan, Tianyang Wang, Joe Yeong, and Junfeng Hao. 2025. Mixture of experts in large language models. *arXiv preprint arXiv:2507.11181*.

A FLOP-Aware Circulating Split Details

Figure 5 visualizes how FAC-Split differs from prior token allocation strategies. While FA-Split statically assigns tokens to branches, FAC-Split rotates assignments across blocks, ensuring full cross-branch exposure. Algorithm 1 formally specifies this deterministic circulation schedule.

Algorithm 1 FLOP-Aware Circulating Split (FAC_Split) strategy

Input: T = $[t_1, t_2, ..., t_L]$
Output: Processed results from SSM and Attention branches
L: Number of tokens in the input sequence
T: Sequence of input tokens
F_a: Number of Attention FLOPs
F_s: Number of SSM FLOPs
block_index Number specifying block index
$block_size \leftarrow \frac{L}{\frac{F_s}{F_a}+1}$ ▷ SSM block processes $\frac{L}{\frac{F_s}{F_a}+1}$ tokens
$block_index \leftarrow 0$
for $i = 0$ to N-1 **do** ▷ N is Number of Parallel Blocks
  $start_index \leftarrow (block_index) \times block_size$
  $end_index \leftarrow (block_index + 1) \times block_size$
  if $end_index < L$ **then**
    $T_{SSM} \leftarrow T[start_index : end_index)$ ▷ block_size tokens to SSM
    $T_{Attention} \leftarrow T[: start_index \& end_index :]$ ▷ Rest of the tokens to Attention
    $SSM_{Output} \leftarrow SSM(T_{SSM})$
    $Attention_{Output} \leftarrow Attention(T_{Attention})$
    Merge SSM_{Output} and $Attention_{Output}$ ▷ Merge results of SSM and Attention
    $block_index \leftarrow block_index + 1$
  else
    $block_index \leftarrow 0$
  end if
end for

B Theoretical Properties of FLOP-Aware Circulating Split

Proposition : Bounded Representation Deviation under FAC-Split
Let $\mathbf{X} = (x_1, \ldots, x_L)$ be an input token sequence and consider a FlowHN architecture composed of B identical parallel blocks. Each block consists of an attention branch $\mathcal{A}$, a SSM branch $\mathcal{S}$, and a fusion operator $\mathcal{F}$.

Assume that:

1. The mappings $\mathcal{A}$ and $\mathcal{S}$ are Lipschitz continuous with constant K.
2. The fusion operator $\mathcal{F}$ is linear and Lipschitz continuous with constant K_F.
3. Tokens are allocated using the FAC_Split strategy such that, over any sequence of B consecutive blocks, every token is processed by both branches at least once.

Let $h^{(B)}_{\text{FAC}}(\mathbf{X})$ denote the representation produced after B FAC-Split blocks, and let $h^{(B)}_{\text{full}}(\mathbf{X})$ denote the representation produced by an otherwise identical model in which both branches process all tokens in every block.

Then there exists a constant $C > 0$ such that

$$\left\| h^{(B)}_{\text{FAC}}(\mathbf{X}) - h^{(B)}_{\text{full}}(\mathbf{X}) \right\| \;\leq\; C \sum_{t=1}^{B} (KK_F)^{B-t} \max_i \left\| \delta_i^{(t)} \right\|,$$

where $\delta_i^{(t)}$ denotes the per-token representation discrepancy induced when token x_i is omitted from one branch at block t.

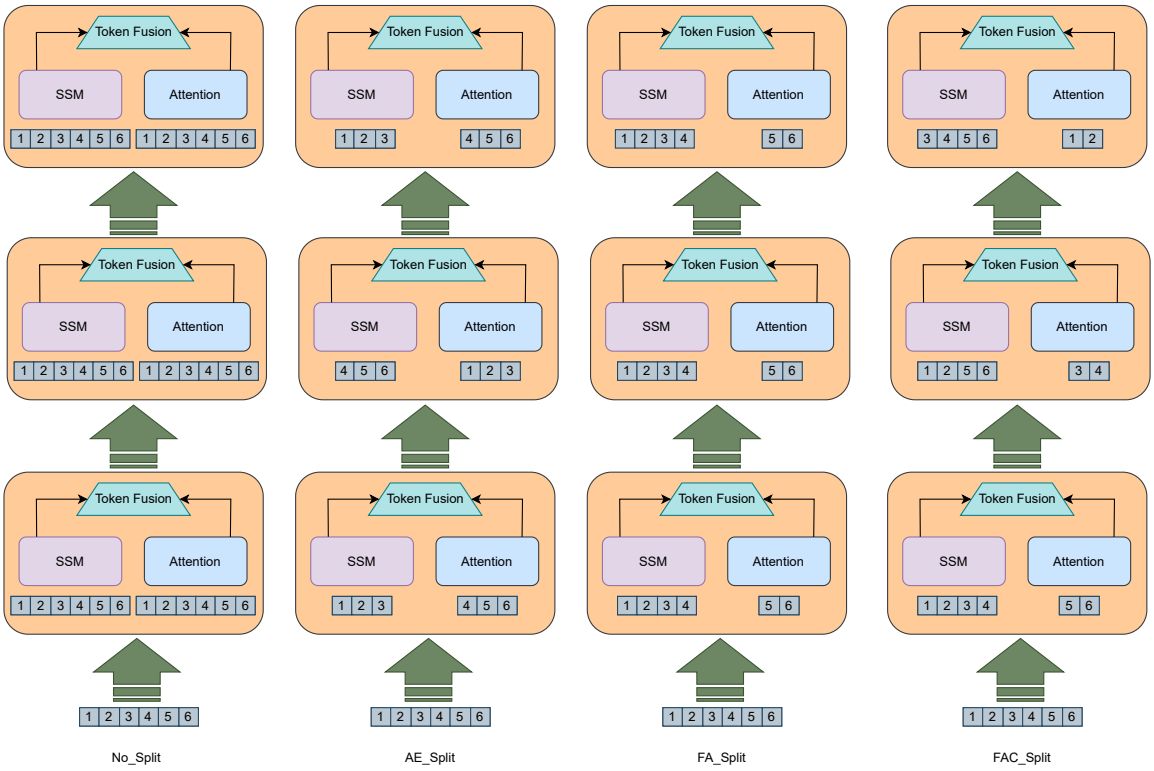

Figure 5: Illustrative example of our four token-splitting strategies in action, showing how input tokens are partitioned and processed under each method.

Proof. We compare the evolution of representations produced by the FAC-Split model and the full-computation model block by block.

Consider a single block t. Under FAC-Split, a token x_i may be omitted from either the attention or SSM branch, while under full computation it is processed by both. This induces a local discrepancy $\delta_i^{(t)}$ in the intermediate representation prior to fusion. By assumption, this discrepancy is bounded.

Since both branches are Lipschitz continuous with constant K, and the fusion operator $\mathcal{F}$ is Lipschitz with constant K_F, the resulting perturbation in the block output is bounded by $KK_F\|\delta_i^{(t)}\|$.

By Assumption 3, each token is processed by both branches at least once over the B blocks, ensuring that branch-specific features omitted at earlier blocks can be reintroduced in subsequent blocks. Therefore, no token is permanently excluded from either branch.

Perturbations introduced at block t propagate through subsequent blocks. Since each block is Lipschitz with constant KK_F, the contribution of a perturbation introduced at block t to the final representation is bounded by $(KK_F)^{B-t}\|\delta_i^{(t)}\|$.

Summing the contributions of perturbations over all blocks and tokens yields

$$\left\|h_{\text{FAC}}^{(B)}(\mathbf{X}) - h_{\text{full}}^{(B)}(\mathbf{X})\right\| \leq C \sum_{t=1}^{B} (KK_F)^{B-t} \max_i \|\delta_i^{(t)}\|,$$

for some constant C absorbing architecture-dependent factors. This establishes the stated bound. □

C Experimental Setup and Reproducibility Details

C.1 Settings

Datasets. We pretrain on the SlimPajama-627B (Soboleva et al., 2023) corpus (Cerebras, 2023). For the compute-controlled comparison in Table 1, we cap training at 1B tokens for the 135M/350M models and 2B tokens for the 1B model, sampled from SlimPajama without replacement. For Table 2, we scale training to 50B tokens from the same corpus.

Baseline Models. To assess the effectiveness of FlowHN, we benchmarked it against several state-of-the-art baselines—including LLaMA, Mamba-1, Mamba-2, Jamba, Sequential Hybrid Model (SHM), and Hymba—across three model scales: 135M, 350M, and 1B parameters. SHM serves as the sequential counterpart to our proposed FlowHN, interleaving LayerNorm + SSM + LayerNorm + MLP with LayerNorm + Attention + LayerNorm + MLP in sequence.

C.2 Implementation details

All experiments in Table 1 were conducted on a single NVIDIA Tesla V100 PCIe GPU with 32GB of memory. All models were trained using a single GPU. The average training times for models with 135M, 350M, and 1B parameters were around 17 hours, 3 days, and 11 days, respectively. Due to limited resources, we were unable to train our proposed model, FlowHN, on massive datasets like those used for large-scale models such as LLaMA, Mamba, Jamba, and Hymba. For a fair and rigorous comparison among all state of the art models, we trained all models, FlowHN and the baselines, from scratch under identical experimental conditions, even if these conditions may not be individually optimal, and not for FlowHN either. Specifically, each configuration was trained using exactly 1,024 tokens. For the 135M and 350M parameter models, we used a batch size of 16 with gradient accumulation over 8 steps, an initial learning rate of 3×10^{-4}, weight decay of 0.1, and AdamW with $\beta_1 = 0.9$ and $\beta_2 = 0.95$. We applied a cosine learning-rate schedule with 10% linear warmup and trained for a total of 1B tokens. For the 1B parameter models, we reduced the batch size to 4 and increased gradient accumulation to 16 to accommodate memory constraints, set the learning rate to 1×10^{-4}, maintained the 0.1 weight decay, and used AdamW with $\beta_1 = 0.9$ and $\beta_2 = 0.999$ under the same cosine schedule and 10% warmup, training across 2B tokens.

In Table 2, All models were trained on 8× V100-32GB GPUs using FP16, ZeRO-3/FSDP, activation checkpointing with micro-batch size fixed at 1 and setting the Gradient Accumulation to the maximum possible value. Training used 50B SlimPajama tokens following a length curriculum (4K→8K→16K), and attention-based models were evaluated at 32K with RoPE scaling (factor=2.0) plus a short 0.5B-token continuation for adaptation.

In Table 3, after pretraining all models on 50B SlimPajama tokens with a length curriculum (4K→8K→16K), we applied light, task-specific supervised instruction fine-tuning using small, publicly available datasets. For Mathematical reasoning, we used 70% GSM8K (Cobbe et al., 2021), 10% SVAMP (Patel et al., 2021), 10% ASDiv (Miao et al., 2021), and 10% qua_rat (Ling et al., 2017) training data. For code generation, we used 50% MBPP (Austin et al., 2021), 40% APPS (Hendrycks et al., 2021), 15% CodeSearchNet (Husain et al., 1909), and 15% FIM (Bavarian et al., 2022) trainign data. For summarization, we fine-tuned on 40% GovReport (Huang et al., 2021), 30% SQuALITY (Wang et al., 2022), and 50% BigPatent (Sharma et al., 2019) training data. For each task, multiple benchmarks were employed for fine-tuning to ensure broader coverage and robustness, as training on a single dataset can lead to domain overfitting and bias the model toward that benchmark's specific distribution. All models were fine-tuned under identical settings (FP16, ZeRO-3/FSDP, activation checkpointing, 8×V100-32GB, micro-batch size 1 ensuring a fair comparison without evaluation data leakage.

D Extended Evaluation and Analysis

D.1 Training Loss

Loss-curve analysis. Figure 6 plots training loss (nats/token) vs. seen tokens for our FlowHN variants (No_Split, AE_Split, FA_Split, FAC_Split) alongside a parallel hybrid (Hymba) and a sequential hybrid (Jamba). AE_Split and FA_Split fall quickly at first yet exhibit diminishing returns, plateauing near 2. This is consistent with their task-agnostic routing and and absence of cross-branch correction. No_Split attains the lowest loss floor because every token is processed by both mechanisms in every layer, but this comes at a cost in MFU/TPS relative to split variants (though it still converges below Jamba and Hymba). FAC_Split is the preferred trade-off: it achieves a No_Split-like floor with better efficiency, owing to per-layer rotation (immediate cross-branch exposure) and FLOP-aware token allocation (balanced finish times). Among baselines, Jamba generally tracks closer to FAC_Split than Hymba and converges to a

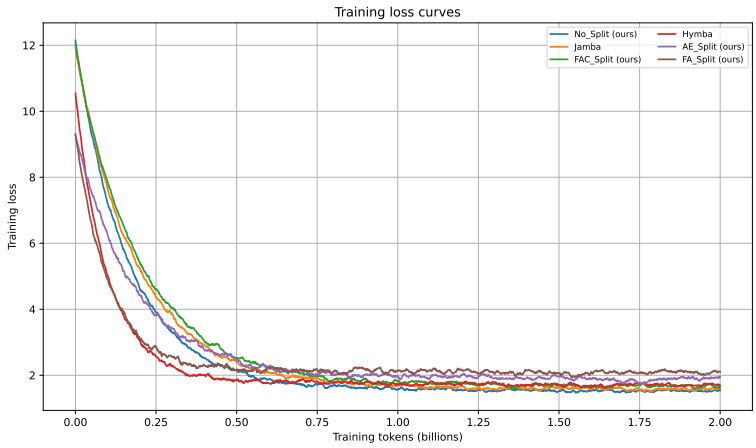

Figure 6: Training loss vs. training tokens for FlowHN-1B variants and hybrid baselines on 2B SlimPajama.

relatively lower loss, yet FAC_Split delivers the best Pareto point—near-optimal loss with materially higher efficiency.

D.2 Evaluation of Token Fusion Strategies

Table 4 compares different token fusion strategies used in FlowHN. Simple Mean or direct Concatenation does not fully reconcile the heterogeneous representations produced by the attention and SSM branches. Introducing a lightweight linear projection substantially improves feature alignment, with Concatenation + Linear achieving the best accuracy (37.41%). Importantly, the linear projection maintains computational efficiency while enabling effective integration of fine-grained attention features and the global state representations learned by the SSM branch.

Table 4: Average results for the 135M-parameter model on commonsense and general reasoning benchmarks.

Token Fusion Strategy	Accuracy (%)
Mean	32.68
Concatenation	34.02
Mean + Linear	35.87
Concatenation + Linear	**37.41**

D.3 Evaluation of Task-Adaptive Token Allocation

Table 5 examines how FlowHN distributes tokens between the attention and SSM branches across different tasks. The FAC_Split mechanism naturally adjusts computation according to task requirements. Precision-sensitive domains such as commonsense reasoning, mathematical reasoning, and code generation route a larger fraction of tokens to the attention branch (64–68%), reflecting their dependence on fine-grained token interactions, while maintaining roughly one-third of tokens within the SSM branch to preserve global context. In contrast, summarization workloads shift toward greater SSM utilization (61%), leveraging its efficiency in modeling long-range structure and coherence. For long-context scenarios, the SSM branch processes the majority of tokens (72%), while deterministic circulation ensures that attention layers continue refining local details across depth. These trends indicate that FLOP-aware allocation aligns computation with task characteristics, enabling FAC_Split to maintain strong efficiency and stable accuracy across diverse deployment settings.

Table 5: Proportation of the tokens assigned to Attention and SSM branches across different Tasks

Task	**Attn (%)**	**SSM (%)**
Commonsense Reasoning	68	32
Math Reasoning	64	36
Coding	65	35
Summarization	39	61
Long-Context	28	72