FlashFFN: Multi-Head Decomposition Enables I/O-Aware Feed-Forward Networks

Minshen Zhang*
ShanghaiTech University
zhangmsh1@shanghaitech.edu.cn
miz055@ucsd.edu

Xiang Hu*
Ant International, Ant Group
shawnxxxhu@tencent.com

Jianguo Li
Ant International, Ant Group

Wei Wu
Ant International, Ant Group
wuwei19850318@gmail.com

Kewei Tu
ShanghaiTech University
tukw@shanghaitech.edu.cn

Abstract

The Feed-Forward Network (FFN) dominates the activation memory of modern Transformers because the wide intermediate tensor is materialized in off-chip HBM between the two matmul stages. FlashAttention-style I/O-aware tiling avoids the analogous cost in attention, but the same one-pass fusion does not fit a monolithic FFN. We identify the reason as a hardware-geometry constraint: under one-pass tiled accumulation a fused FFN kernel must keep an output accumulator spanning the full hidden dimension in on-chip SRAM, which is infeasible for hidden dimensions of 2048 or more on currently shipping NVIDIA datacenter GPUs (A100/H100/B200) under one-pass tiled execution with sequence tiles of 128 or more tokens. We formalize this as an SRAM-feasibility proposition and prove a corollary: *multi-head decomposition is the simplest architectural transform that brings FFN inside this SRAM-feasible regime*, because it makes the per-head accumulator width (the hidden dimension divided by the number of FFN heads) a free parameter that can be driven below the GPU's SRAM cutoff. Naïve multi-head FFN, however, blows up the per-head expansion ratio as the hidden dimension grows, drifting away from the well-validated SwiGLU $8/3$ optimum and degrading quality at scale; we resolve this with a dense MoE-like sub-network correction that restores the optimal ratio while preserving SRAM-feasibility. The resulting architecture, **FlashFFN**, is an SRAM-feasible FFN block that replaces SwiGLU under matched parameter and FLOPs budgets and derives directly from the SRAM constraint. Across 128M–1.3B parameter models trained on 60–100B tokens of The Pile, FlashFFN achieves a Pareto improvement over SwiGLU at fixed compute: **up to** $2.7\times$ **lower** peak activation memory at long sequences, inference latency that matches or improves over SwiGLU (up to $1.18\times$ *faster* end-to-end prefill and $1.19\times$ faster decode on H100), and consistent quality gains (up to -0.84 in evaluation perplexity (PPL) and $+1.60$ average accuracy points across six downstream benchmarks at 1.3B).[2]

*Equal contribution.

[2]Reference implementation: `https://github.com/alexzms/FlashMHF`

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

1 Introduction

The Transformer architecture is the dominant backbone of modern Large Language Models [Vaswani et al., 2017b], with multi-head self-attention and the position-wise Feed-Forward Network (FFN) contributing roughly equal shares of parameters and activation memory [Gerber, 2025]. The FFN is memory-bound at scale: its intermediate activation tensor of shape $L \times d_{\text{ff}}$ (with d_{ff} in the thousands) is materialized in HBM between the two matmul stages, dominating activation memory in standard implementations. FlashAttention [Dao et al., 2022] reshaped attention by recognizing the analogous bottleneck and removing it through I/O-aware tiling. Tiling and kernel fusion are also standard for the FFN: Liger Kernel fuses its element-wise operations [Hsu et al., 2024], and concurrent work uses deeply fused kernels for long-context inference [Zhang et al., 2026]. These kernels optimize the monolithic FFN as it is. We ask why a FlashAttention-style one-pass kernel, which keeps the $L \times d_{\text{ff}}$ intermediate on chip, does not fit a monolithic SwiGLU FFN at large d_{model}, and what minimal architectural change makes it fit.

The SRAM-feasibility frontier of FFN computation. I/O-aware tiling requires the on-chip working set to fit in SRAM during the inner accumulation loop. We formalize this in Section 3: under one-pass tiled execution the binding term is an output accumulator of width d_{model}, and the per-tile working set scales as $\Theta(B_q \cdot d_{\text{model}})$ in the monolithic case. With modern $d_{\text{model}} \geq 2048$, monolithic FFN sits outside the SRAM-feasible regime on every currently shipping NVIDIA datacenter GPU we considered (A100/H100/B200). The obstacle is the accumulator width under this schedule, not a lack of engineering effort; other schedules exist but pay other costs (Section 6).

Multi-head decomposition is the simplest fix. Decomposing the model dimension into H_{ffn} FFN heads (written H below; this count is set independently of the attention heads) of width $d_h = d_{\text{model}}/H$ drops the per-tile working set to $\Theta(B_q \cdot d_h)$, fitting in SRAM whenever d_h falls below the GPU-specific cutoff (Cor. 3.3). Multi-head structure for FFN is therefore not an expressivity convenience but the architectural property that makes FFN I/O-aware-tilable at all; architecture and kernel are mutually dependent: without the kernel, MH-MoE-style splits [Wu et al., 2024] pay linear-in-H HBM cost; without multi-head, no kernel fits. A second-order failure follows: holding d_h fixed forces the per-head ratio d_{ff}/d_h to grow as H ($16\times$ at 128M to $45\times$ at 1.3B), drifting from the SwiGLU $8/3$ optimum. We resolve this with E MoE-like sub-networks per head of width $d_e \approx (8/3)\, d_h$, restoring the optimum while preserving SRAM-feasibility.

Contributions.

- **A formal SRAM-feasibility framework for FFN computation** (Section 3): a Proposition characterizing when a fused tiled FFN kernel is physically realizable on a given GPU, showing that monolithic FFN is infeasible under one-pass tiled execution on currently shipping datacenter GPUs.
- **Multi-head decomposition derived as the simplest architectural transform** that satisfies the SRAM-feasibility constraint (Section 3, Corollary 3.3); the second-order ratio-imbalance failure resolved via MoE-like sub-networks (Section 4). Neither multi-head splitting nor tiling is new; our contribution is their composition on the FFN. Tiling alone does not fit on chip (Cor. 3.2), and the split alone is quality-neutral (MH-FFN control) and, without a fused kernel, costs memory linear in H; composed, each removes what blocks the other.
- **FlashFFN: a constraint-derived FFN with empirical Pareto improvement.** At matched parameters and FLOPs, the dense FlashFFN architecture improves on the dense SwiGLU baseline on every axis we measure. Across 128M–1.3B (Section 5), it reaches up to $2.7\times$ lower per-layer peak FFN memory, up to $1.18\times$ and $1.19\times$ higher model-level prefill and decode throughput on H100, and consistently lower evaluation perplexity and higher downstream accuracy than a strong SwiGLU baseline.

2 Preliminaries

Notation. L denotes the sequence length, d_{model} the model dimension, and d_{ff} the FFN intermediate dimension. We write $\mathbf{X} \in \mathbb{R}^{L \times d_{\text{model}}}$ for the FFN input. Standard SwiGLU [Touvron

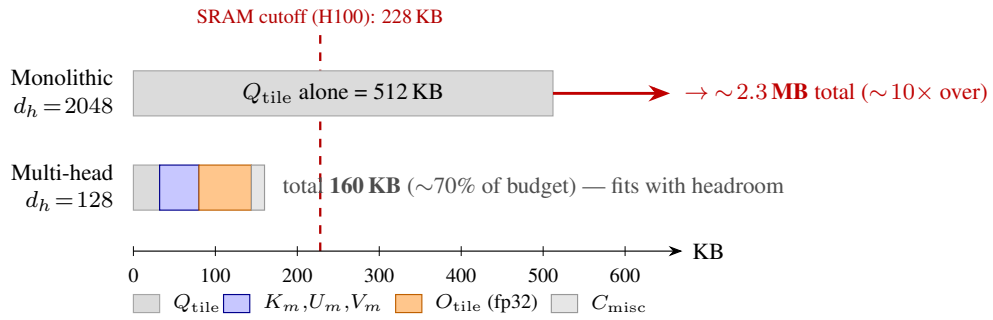


Figure 1: **Per-SM SRAM budget on H100** ($B_q = 128$, $B_k = 64$, bf16 inputs, fp32 accumulator). Multi-head FFN at $d_h = 128$ keeps all resident tiles ($Q_{\text{tile}}, K_m, U_m, V_m, O_{\text{tile}}$, plus the C_{misc} margin) inside the 228 KB cutoff with headroom. Monolithic FFN at $d_h = d_{\text{model}} = 2048$ overflows by an order of magnitude: Q_{tile} alone (512 KB) already exceeds the cutoff, and the full set sums to ∼2.3 MB. This illustrates Cor. 3.2.

et al., 2023, Shazeer, 2020] computes $\text{SwiGLU}(\mathbf{X}) = \big((\mathbf{X}\mathbf{W}_{\text{up}}) \odot \text{SiLU}(\mathbf{X}\mathbf{W}_{\text{gate}})\big)\mathbf{W}_{\text{down}}$, with $\mathbf{W}_{\text{up}}, \mathbf{W}_{\text{gate}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$ and $\mathbf{W}_{\text{down}} \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$.

Key-value form for tiling. For the per-head analysis in Sections 3–4 we use a query/key/up/value form with symbols echoing attention. For an input $\mathbf{Q} \in \mathbb{R}^{L \times d}$ and parameters $\mathbf{K}, \mathbf{U}, \mathbf{V} \in \mathbb{R}^{d_{\text{ff}} \times d}$, define

$$\widetilde{\text{FFN}}(\mathbf{Q}; \mathbf{K}, \mathbf{U}, \mathbf{V}) \;=:\; \big(\text{SiLU}(\mathbf{Q}\,\mathbf{K}^\top) \;\odot\; (\mathbf{Q}\,\mathbf{U}^\top)\big)\,\mathbf{V}. \tag{1}$$

Setting $\mathbf{Q} = \mathbf{X}$, $\mathbf{K} = \mathbf{W}_{\text{gate}}^\top$, $\mathbf{U} = \mathbf{W}_{\text{up}}^\top$, $\mathbf{V} = \mathbf{W}_{\text{down}}$ recovers SwiGLU exactly; this reparameterization makes the structural symmetry [Vaswani et al., 2017a, Geva et al., 2020] between FFN and attention explicit, and lets us tile FFN computation along d_{ff} analogously to how FlashAttention tiles attention along the key dimension.

Headwise operations. Let H ($= H_{\text{ffn}}$) be the number of FFN heads, which is chosen independently of the number of attention heads (16 FFN heads vs. 32 attention heads at 1.3B), and $d_h = d_{\text{model}}/H$ the per-head width. We write $\text{split}_H(\cdot)\colon \mathbb{R}^{L \times d_{\text{model}}} \to \mathbb{R}^{L \times H \times d_h}$ for the model-dimension split and $\text{concat}_H(\cdot)$ for its inverse.

3 The Hardware Geometry of FFN Computation

3.1 Tiled, I/O-Aware FFN Computation

A FlashAttention-style tiled kernel for $\widetilde{\text{FFN}}$ (Eq. (1)) partitions $\mathbf{K}, \mathbf{U}, \mathbf{V} \in \mathbb{R}^{d_{\text{ff}} \times d_h}$ along d_{ff} into M blocks of size B_k, partitions $\mathbf{Q} \in \mathbb{R}^{L \times d_h}$ along L into outer-loop tiles of size B_q, and accumulates the per-tile output $\mathbf{O}_{\text{tile}} \in \mathbb{R}^{B_q \times d_h}$ on chip:

$$\mathbf{O}_{\text{tile}} \;+=\; \big(\text{SiLU}(\mathbf{Q}_{\text{tile}}\mathbf{K}_m^\top) \;\odot\; (\mathbf{Q}_{\text{tile}}\mathbf{U}_m^\top)\big)\,\mathbf{V}_m, \quad m = 1, \ldots, M. \tag{2}$$

The intermediate $\mathbf{A}_m = \text{SiLU}(\mathbf{Q}_{\text{tile}}\mathbf{K}_m^\top) \odot (\mathbf{Q}_{\text{tile}}\mathbf{U}_m^\top) \in \mathbb{R}^{B_q \times B_k}$ is register-resident and never reaches HBM. The remaining question is when this loop fits on a given GPU.

3.2 The SRAM-Feasibility Proposition

Let S be the per-SM SRAM capacity, s the activation byte width ($s = 2$ for bf16), and $s_{\text{acc}} = 4$ the fp32 accumulator byte width.

Proposition 3.1 (SRAM-Feasibility of Fused Tiled FFN)**.** *The kernel of Eq.* (2) *is SRAM-feasible iff*

$$\underbrace{B_q \cdot d_h \cdot s}_{\mathbf{Q}_{\text{tile}}} + \underbrace{3 \cdot B_k \cdot d_h \cdot s}_{\mathbf{K}_m, \mathbf{U}_m, \mathbf{V}_m} + \underbrace{B_q \cdot d_h \cdot s_{\text{acc}}}_{\mathbf{O}_{\text{tile}}} \;\le\; S - C_{\text{misc}}, \tag{3}$$

where C_{misc} covers register-spill margin, software-pipelining stage buffers, and synchronization primitives. Equivalently,

$$d_h \;\le\; d_h^*(B_q, B_k) \;=:\; \frac{S - C_{\text{misc}}}{B_q(s + s_{\text{acc}}) \;+\; 3\,B_k\,s}. \tag{4}$$

Proof. The three under-braced terms (covering five resident tiles: $\mathbf{Q}_{\text{tile}}$, $\mathbf{K}_m$, $\mathbf{U}_m$, $\mathbf{V}_m$, $\mathbf{O}_{\text{tile}}$) are the only ones resident in SRAM ($\mathbf{A}_m$ is register-only). Summing them and bounding by $S - C_{\text{misc}}$ gives Eq. (3); solving for d_h gives Eq. (4). □

Worked example (H100). With $S = 228$ KB, $B_q = 128$ (smallest tile saturating Hopper tensor cores), $B_k = 64$, $s = 2$, $s_{\text{acc}} = 4$, $C_{\text{misc}} \approx 16$ KB, Eq. (4) gives the strict bound $d_h^* \approx 184$. Empirically $d_h = 128$ runs with headroom; $d_h = 256$ marginally exceeds the bound and runs at reduced occupancy; $d_h = 512$ either spills or forces B_q below the tensor-core threshold (Fig. 1).

3.3 Corollary: Monolithic FFN is SRAM-Infeasible

Corollary 3.2 (Monolithic FFN is SRAM-infeasible under one-pass tiled execution)**.** *A standard SwiGLU FFN is the single-head case $d_h = d_{\text{model}}$. With bf16 inputs, an fp32 accumulator, $B_q \geq 128$ and $d_{\text{model}} \geq 2048$, its resident tiles need more SRAM than any shipping datacenter GPU provides per SM (about 2 MB would be needed; A100 has 192 KB, H100 and B200 have 228 KB), so Eq.* (3) *is violated.*

Proof. $\mathbf{Q}_{\text{tile}}$ alone is $128 \cdot 2048 \cdot 2 = 512$ KB, already exceeding per-SM SRAM on every GPU listed (Fig. 1). □

3.4 Multi-Head Decomposition Restores SRAM-Feasibility

Corollary 3.3 (Multi-head decomposition restores SRAM-feasibility)**.** *Splitting the model dimension into $H \geq \lceil d_{\text{model}}/d_h^* \rceil$ heads of width $d_h = d_{\text{model}}/H$ and running the loop of Eq.* (2) *per head makes it fit in SRAM for any d_{model}: the per-tile working set becomes $\Theta(B_q \cdot d_h)$ and no longer grows with d_{model}.*

The rest of the paper builds on Corollary 3.3. In this setting, multi-head structure for FFN is not an expressivity convenience but the hardware-geometry transform that makes one-pass I/O-aware tiling of the FFN possible. The architecture and the kernel depend on each other. Without the kernel, multi-head FFN materializes H per-head intermediates in HBM, paying $\Theta(L \cdot H \cdot d_{\text{ff}})$ activation memory (the linear-in-H blow-up an unfused MH-MoE-style design [Wu et al., 2024] would incur); without multi-head, the kernel cannot fit on chip.

4 FlashFFN: A Constraint-Derived FFN

Section 3 established two structural requirements an SRAM-feasible FFN must satisfy on modern GPUs:

- **(R1) Multi-head decomposition.** The model dimension must be split into heads of width $d_h \leq d_h^*$ (Corollary 3.3); on H100 with bf16 inputs and tile size $B_q = 128$, the strict cutoff is $d_h^* \approx 184$ (Proposition 3.1), so $d_h = 128$ runs with headroom while $d_h = 256$ marginally exceeds the strict bound and runs at reduced occupancy.
- **(R2) Pathway-level SwiGLU $8/3$ ratio.** SwiGLU's well-validated optimum sets the FFN intermediate-to-input width ratio to $8/3$ [Touvron et al., 2023, Shazeer, 2020]. Since (R1) fixes the per-head input width to d_h, applying the FFN computation directly per head with intermediate width d_{ff} gives a per-head ratio d_{ff}/d_h that scales as H and explodes at scale: $\{16, 21, 45\}$ across our 128M/370M/1.3B configurations, far above the $8/3$ optimum. Restoring the optimum requires a per-head pathway of width $d_e \approx (8/3)\, d_h$, kept independent of d_{model}.

FlashFFN is the design we instantiate to satisfy both: a multi-head FFN whose per-head computation is itself decomposed into E parallel SwiGLU sub-networks of width $d_e \approx (8/3)\, d_h$, gated per token. Figure 2 sketches the layer-level structure; Section 4.1 writes it out as equations and Figure 3 zooms into a single head. Section 4.2 realizes the design in a fused I/O-aware kernel that physically achieves SRAM residency. The two architectural moves are not independent fixes added one at a time; together they make the two constraints (R1) and (R2) satisfiable at the same time. We empirically validate each in isolation in Section 5: the MH-FFN control (R1 only) tracks the SwiGLU baseline at 370M, and the Dense-MoE control (R2 only) underperforms it; only the composition yields a strict gain.

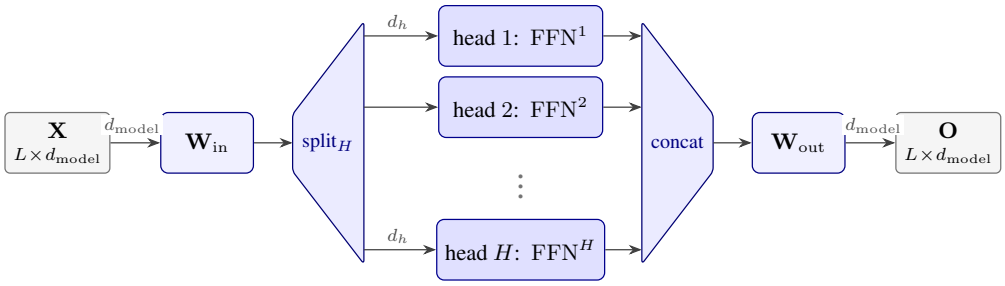


Figure 2: **Multi-head decomposition (layer view).** The block input $\mathbf{X} \in \mathbb{R}^{L \times d_{\text{model}}}$ is projected by $\mathbf{W}_{\text{in}}$ and split along the model dimension into H parallel heads of width $d_h = d_{\text{model}}/H$. Each head computes its own FFN (FFN^h, detailed in Fig. 3); the H outputs are concatenated and projected back to d_{model} by $\mathbf{W}_{\text{out}}$. Keeping $d_h \le d_h^*$ is what makes each head's per-tile working set SRAM-feasible (R1, Prop. 3.1).

4.1 The FlashFFN Architecture

Per-head input. Given $\mathbf{X} \in \mathbb{R}^{L \times d_{\text{model}}}$, we form per-head queries by a single linear projection and an H-way split along the model dimension:

$$\mathbf{Q} \;=\; \text{split}_H\big(\mathbf{X}\,\mathbf{W}_{\text{in}}\big) \;\in\; \mathbb{R}^{L \times H \times d_h}, \quad \mathbf{W}_{\text{in}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}. \tag{5}$$

Each head operates in a width-d_h subspace of the model dimension, satisfying (R1).

Sub-networks at the SwiGLU $8/3$ ratio. Within each head h, we instantiate E parallel SwiGLU pathways, each of intermediate width $d_e \approx (8/3)\, d_h$, giving a per-head intermediate width of $E \cdot d_e$. The number of sub-networks E is chosen so the FFN-block parameter count $3\, H\, E\, d_e\, d_h$ matches the SwiGLU baseline $3\, d_{\text{ff}}\, d_{\text{model}}$ (since $d_h = d_{\text{model}}/H$, this reduces to $E\, d_e = d_{\text{ff}}$, matching the Table 6 entries directly). The parameters of the e-th sub-network in head h are $\mathbf{K}_e^h, \mathbf{U}_e^h, \mathbf{V}_e^h \in \mathbb{R}^{d_e \times d_h}$, and each sub-network is private to its head (no cross-token or cross-head communication, which preserves the per-head fused-kernel structure required by Section 4.2).

Per-token sigmoid-normalized gating. A gating matrix $\mathbf{W}^h \in \mathbb{R}^{d_h \times E}$ maps each token's per-head input to E logits that are sigmoid-normalized into pathway weights:

$$\mathbf{P}^h = \mathbf{Q}_{:,h,:}\,\mathbf{W}^h, \qquad \mathbf{R}^h_{\ell,e} = \frac{\sigma(\mathbf{P}^h_{\ell,e})}{\sum_{e'=1}^{E} \sigma(\mathbf{P}^h_{\ell,e'}) + \varepsilon}, \quad e = 1, \ldots, E. \tag{6}$$

Unlike sparse top-k MoE, this gating is dense (all sub-networks are evaluated every step); its purpose is not capacity decoupling but per-token re-weighting of the E ratio-correct pathways.

Per-head aggregation and head merging. The head output is the gated sum of its E sub-network outputs:

$$\mathbf{S}_{\ell,h,:} \;=\; \sum_{e=1}^{E} \mathbf{R}^h_{\ell,e}\, \widetilde{\text{FFN}}\big(\mathbf{Q}_{\ell,h,:};\, \mathbf{K}^h_e,\, \mathbf{U}^h_e,\, \mathbf{V}^h_e\big), \tag{7}$$

and the heads are merged by an output projection back to d_{model}:

$$\mathbf{O}^{\text{FlashFFN}} \;=\; \text{concat}_H(\mathbf{S})\,\mathbf{W}_{\text{out}}, \qquad \mathbf{W}_{\text{out}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}. \tag{8}$$

4.2 Realizing SRAM-Residency: The I/O-Aware Fused Kernel

This subsection translates the SRAM-feasibility result of Proposition 3.1 into actual on-chip residency, by describing the fused kernel that physically realizes the tiled loop of Eq. (2) for the sub-networked head defined above.

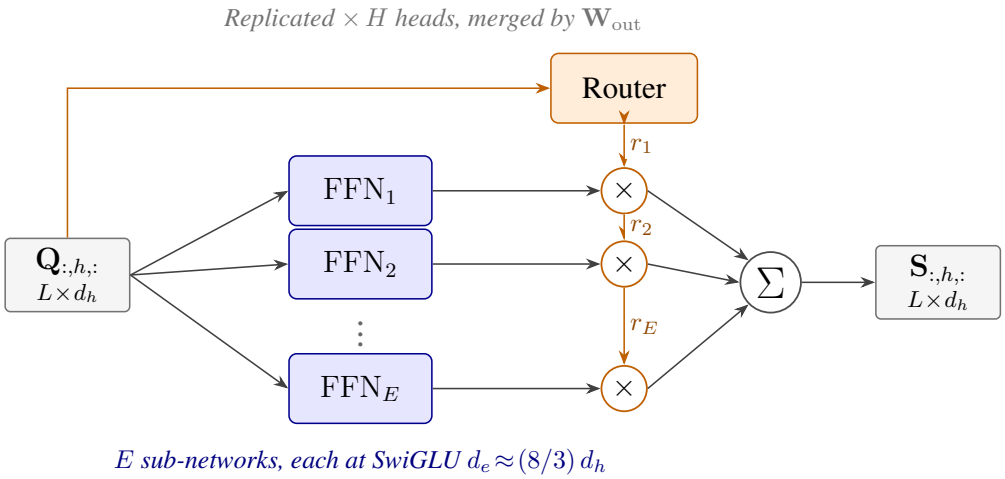


Figure 3: **Per-head FFN structure** (one head; replicated across all H). The per-head input enters E parallel SwiGLU sub-networks at the optimal $d_e \approx (8/3)d_h$ ratio (R2). The Router produces per-token sigmoid-normalized gates $r_1, \ldots, r_E$ that re-weight each sub-network's output before aggregation into the head output $\mathbf{S}_{:,h,:}$.

For each head, the per-sub-network input $\mathbf{Q}_{:,h,:} \in \mathbb{R}^{L \times d_h}$ and parameter matrices $\mathbf{K}_e^h, \mathbf{U}_e^h, \mathbf{V}_e^h \in \mathbb{R}^{d_e \times d_h}$ enter the loop of Eq. (2) with $d_{\text{ff}} = d_e$ per sub-network. The kernel's outer loop iterates over sub-networks $e = 1, \ldots, E$ (applying the gating weight $\mathbf{R}_{:,e}^h$ as a multiplicative scalar per token); its inner loop iterates over d_e/B_k parameter tiles per sub-network. At every step, the accumulator $\mathbf{O}_{\text{tile}} \in \mathbb{R}^{B_q \times d_h}$ stays in SRAM, the parameter tiles $\mathbf{K}_m, \mathbf{U}_m, \mathbf{V}_m$ are streamed in from HBM, and the intermediate $\mathbf{A}_m$ never reaches HBM. The HBM-resident intermediate therefore drops from $\mathcal{O}(L \cdot d_{\text{ff}})$ to an on-chip live working set of $\mathcal{O}(B_q \cdot d_h)$, which lowers per-layer peak FFN memory by up to 2.7× at long sequences (Table 3). The complete kernel is given in Algorithm 1 (Triton, consumer-grade and Ampere) and Algorithm 4 (Hopper/ThunderKittens [Spector et al., 2025] with warp-specialized producer/consumer pipelining); the corresponding backward passes are Algorithms 2–3 and Algorithm 5, respectively.

5 Experiments

We validate the scalability and efficacy of our approach by training models across three scales: 128M, 370M, and 1.3B parameters. All models are trained on THE PILE [Gao et al., 2020] with a context length of 4,096. We train the 128M and 370M models for 60B tokens and the 1.3B model for 100B tokens. Evaluation is performed on the PG19 [Rae et al., 2019] validation split. To ensure fair comparison, we strictly match parameter counts and synchronize optimizer settings across all conditions. Matching the budget fixes total parameters and FLOPs but not depth: FlashFFN uses 10/21/20 layers against the baseline's 12/24/24 at 128M/370M/1.3B (1.3210B vs. 1.3233B parameters at 1.3B; Table 6), following the standard cross-architecture protocol [Dao and Gu, 2024]. All 1.3B comparisons, including model-level efficiency, use these two trained models. Depth alone does not produce the gain: the 20-layer MH-FFN control at 370M gains nothing (Appendix A).

Baselines. We compare FlashFFN against four distinct baselines, each isolating one design axis. (i) The **Standard Baseline** is a Llama-Like [Touvron et al., 2023] model utilizing a standard position-wise SwiGLU FFN and RoPE [Su et al., 2021] embeddings. (ii) **MH-FFN** is the naive multi-head split where every head reads the full intermediate width d_{ff}, giving a per-head expansion ratio $d_{\text{ff}}/d_h = (8/3)\, H$ that drifts off the SwiGLU optimum and grows linearly with H (16× at 128M, 21× at 370M, 45× at 1.3B); this isolates R1 (multi-head decomposition) by leaving R2 (per-pathway $\sim 8/3$ restoration) unsatisfied. (iii) **Dense-MoE** sets $H = 1$, retaining the MoE-like sub-networks but discarding the multi-head split; this isolates R2 from R1. (iv) **Parametric KV (PKV)** replaces the FFN with multi-head attention over learnable keys and values ($d_e = 1$), validating the necessity of the position-wise SwiGLU structure (element-wise SiLU/Hadamard) over an attention-style softmax.

Model	128M	370M	1.3B
Baseline (SwiGLU)	26.74	20.70	17.17
MH-FFN (R1 only)	26.31	20.74	–
Dense-MoE (R2 only)	26.82	21.37	–
PKV (no FFN)	–	28.05	–
FlashFFN (d_h=64)	26.36	21.03	17.27
FlashFFN (d_h=128)	**26.00**	**20.37**	**16.33**
FlashFFN (d_h=256)	27.06	20.68	16.43

Table 1: Evaluation PPL on PG19 across 128M, 370M, and 1.3B. **MH-FFN** ablates R2 (multi-head only); **Dense-MoE** ablates R1 (H=1); **PKV** replaces the FFN with attention over learnable KV (d_e=1). Missing entries (–) are by design: PKV, a validated negative control (28.05 vs. 20.70 PPL at 370M), and the R1/R2 controls are run only where the mechanistic attribution is done, and 1.3B compute is reserved for FlashFFN and the baseline. 370M carries the full mechanistic attribution. FlashFFN-128 (bold) is the best variant at every scale.

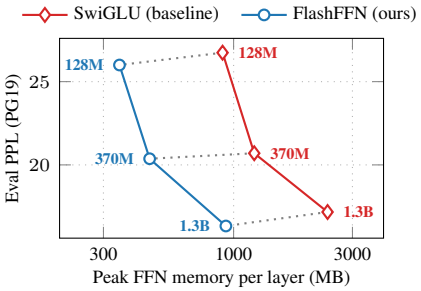


Figure 4: **Pareto improvement at every scale.** Eval PPL vs. measured per-layer peak FFN memory (H100, trained configurations, $bs=8$, $L=8064$, log x; values in Appendix G), a uniform reduction of about $2.6\times$. FlashFFN (blue) lies strictly southwest of SwiGLU (red) at 128M, 370M, and 1.3B.

FlashFFN. We evaluate FlashFFN (Section 4) at three head-dimension settings, $d_h \in \{64, 128, 256\}$, to identify the quality-optimal head dimension at and around the SRAM-feasible cutoff (Prop. 3.1 gives $d_h^* \approx 184$, so $d_h \in \{64, 128\}$ are interior and $d_h = 256$ marginally exceeds the strict bound). The FlashFFN block replaces the SwiGLU FFN at every layer of the baseline; attention, RMSNorm [Zhang and Sennrich, 2019], RoPE, and embedding are kept identical. We tune the FlashFFN configuration so that total parameter count and per-step FLOPs match the SwiGLU baseline at every scale (full configurations in Appendix F, Table 6). Under this strict alignment, FlashFFN-128hdim attains lower evaluation PPL than the SwiGLU baseline at every scale, with the gap *widening at the largest scale* ($\Delta\text{PPL}=0.74,\ 0.33,\ 0.84$ at 128M/370M/1.3B; Section 5.1, Figure 4).

For comprehensive details regarding model architecture, tokenizer specifications, detailed hyperparameters, and sub-network configurations, please refer to Appendix H.

5.1 Language Modeling at Different Scales

Main result and ablation controls. FlashFFN-128hdim achieves the lowest evaluation PPL at every tested scale (Table 1). We use 370M as our mechanistic-attribution setting where compute permits running all three structural controls; 1.3B is reserved for scale-up validation. At 370M, three controls isolate where the gain comes from, mapping one-to-one onto requirements R1 (multi-head decomposition) and R2 (per-pathway $\sim 8/3$ ratio) of Section 4. *MH-FFN* (R1 only, no sub-networks; per-head ratio $(8/3)\,H$) tracks the baseline closely with the predicted scale-dependent decay: at 128M ($H=6$, ratio $16\times$) slightly better than baseline (PPL 26.31 vs. 26.74); at 370M ($H=8$, ratio $21\times$) the lead has vanished (20.74 vs. 20.70). As H grows the per-head ratio drifts further from the SwiGLU optimum, bleeding back the expressivity gain, so R1 alone is a wash at scale. *Dense-MoE* (R2 only, $H=1$) underperforms the baseline at both scales we ran it (26.82 vs. 26.74 at 128M; 21.37 vs. 20.70 at 370M): the sub-network aggregation alone is a small regression. Only the composition yields a strict gain (FlashFFN at PPL 20.37, 370M; 16.33, 1.3B); each piece is roughly free on its own and together they are strictly better, exactly as the R1+R2 derivation predicts. Mechanistically, R1 alone shrinks per-head expressivity (with H=8 at 370M, each head's ratio is $21\times$, far from the SwiGLU $8/3$ optimum) without restoring it; R2 alone restores the $8/3$ ratio but inside a single H=1 FFN, where it offers no expressivity edge over a monolithic SwiGLU and the dense gating adds optimization noise without a structural counterpart to absorb it. The composition is the first variant where each head simultaneously sits in a width-d_h subspace (R1) *and* runs E pathways at the $\sim 8/3$ ratio (R2), giving every head the expressivity floor that SwiGLU was tuned for; this is the architectural completeness, not a tuning happenstance, that produces the strict gain. *PKV* (FFN replaced by multi-head attention over learnable KV, $d_e=1$) collapses to a separate league (PPL 28.05 at 370M), confirming that the position-wise SwiGLU structure (element-wise SiLU/Hadamard rather than attention's softmax) is also essential.

Table 2: Downstream benchmarks for 1.3B and 370M models. **Bold** = best, underline = second best. The grey rows indicate that all the **best** results come from the FlashFFN architecture; FlashFFN-128hdim performs the best across different scales.

Scale	Model	HellaSwag↑	SIQA↑	PIQA↑	OBQA↑	WinoGrande↑	RACE↑	Avg.↑
370M	Baseline (SwiGLU)	33.20	40.94	64.53	27.50	51.70	21.66	39.92
370M	PKV (control)	27.85	39.61	60.50	27.20	51.78	21.67	38.10
370M	FlashFFN-64hdim	32.57	**41.50**	65.34	27.60	**53.35**	21.59	40.32
370M	FlashFFN-128hdim	**33.97**	40.63	**66.27**	**27.80**	52.41	21.80	**40.48**
370M	FlashFFN-256hdim	33.60	39.71	64.85	27.30	52.17	**21.94**	39.92
1.3B	Baseline	39.47	41.76	67.30	27.60	52.64	21.73	41.75
1.3B	FlashFFN-64hdim	39.80	42.17	67.08	27.20	51.30	**22.62**	41.70
1.3B	FlashFFN-128hdim	**42.96**	**44.17**	68.44	**27.80**	**54.46**	22.26	**43.35**
1.3B	FlashFFN-256hdim	41.80	42.32	**68.88**	27.60	53.35	22.01	42.66

Head-dimension sweet spot and scale-up. Sweeping $d_h \in \{64, 128, 256\}$ at both 370M and 1.3B (Table 1) picks out $d_h = 128$ as the empirical quality optimum: 64 underfits (per-head bottleneck), and 256 approaches monolithic-FFN behavior on the model side. At 1.3B and 100B training tokens FlashFFN-128hdim retains its lead over the SwiGLU baseline, with $\Delta\mathrm{PPL} = 0.84$ on PG19, the largest gap of our three scales ($\Delta\mathrm{PPL} = 0.74, 0.33, 0.84$ at 128M/370M/1.3B; Figure 4).

5.2 Downstream Task Performance

Setup. To ascertain whether the improved validation PPL of FlashFFN translates into enhanced downstream capabilities, we evaluated our 370M and 1.3B models on a comprehensive suite of benchmarks. We assess commonsense reasoning using HellaSwag [Zellers et al., 2019], Social IQA (SIQA) [Sap et al., 2019], Physical IQA (PIQA) [Bisk et al., 2019], OpenBookQA (OBQA) [Mihaylov et al., 2018], and WinoGrande [Levesque et al., 2011, Sakaguchi et al., 2019]. Additionally, we evaluate reading comprehension using the RACE dataset [Lai et al., 2017].

Results. As summarized in Table 2, the per-task winner on every individual benchmark across both 370M and 1.3B is a FlashFFN variant (highlighted in grey). Given our modest training scale (60–100B tokens), we treat the average across the benchmark suite as a more robust indicator than any single per-task score. **FlashFFN-128hdim** secures the highest average at both scales, confirming $d_h = 128$ as the effective sweet spot identified by Table 1; off-optimum settings ($d_h \in \{64, 256\}$) sit within the SwiGLU baseline's range on average. The takeaway is therefore the matched one between LM PPL and downstream: *the optimal variant* ($d_h = 128$) improves over baseline at every scale we tested, while off-optimum head-dimensions are roughly baseline-comparable. The 1.3B per-task gain pattern is also informative: the largest improvements concentrate on reasoning-heavy tasks (HellaSwag $+3.49$, SIQA $+2.41$, WinoGrande $+1.82$), with smaller gains on memorization-leaning tasks (PIQA $+1.14$, RACE $+0.53$, OBQA $+0.20$), suggesting that the gain reflects an expressivity improvement rather than a memorization advantage from the matched parameter budget.

5.3 Pareto Efficiency at Fixed Compute

We now ask whether FlashFFN's quality improvements come at a hidden compute or memory cost. They do not: the architecture obtains its quality gains while *simultaneously* reducing peak activation memory and improving model-level throughput, placing it strictly southwest of the SwiGLU baseline in the (peak memory, evaluation PPL) plane at every model scale we tested.

Measurement protocol. All efficiency results are measured on H100. Per-layer numbers time the full FFN replacement block, from $\mathbf{W}_{\mathrm{in}}$ through routing, the fused kernel and concatenation to $\mathbf{W}_{\mathrm{out}}$ (Appendix G). Model-level numbers run the two trained 1.3B models end to end (20-layer FlashFFN vs. 24-layer SwiGLU).

Peak memory against fused SwiGLU. Under one protocol for all implementations (Table 3), FlashFFN has the lowest peak memory at every length: 2.4–$2.7\times$ below eager SwiGLU and Liger, and 1.7–$1.9\times$ below `torch.compile`, the strongest fused baseline. These implementations still

Table 3: Per-layer peak FFN memory (MB) on H100 for FlashFFN and SwiGLU implementations (Liger [Hsu et al., 2024], `torch.compile` [Ansel et al., 2024], xFormers [Lefaudeux et al., 2022]), all measured under the same protocol ($bs=8$, trained 1.3B FFN configuration). The last two columns divide eager and `torch.compile` memory by FlashFFN memory.

L	Eager	Liger	`torch.compile`	xFormers	**FlashFFN**	vs. eager	vs. compile
4032	1240	1240	901	1579	**524**	$2.37\times$	$1.72\times$
8064	2384	2384	1706	3062	**931**	$2.56\times$	$1.83\times$
16128	4666	4666	3311	6020	**1746**	$2.67\times$	$1.90\times$
32832	9398	9398	6640	12156	**3434**	$\mathbf{2.74}\times$	$\mathbf{1.93}\times$

Table 4: Model-level throughput (tokens/s) of the trained 1.3B models on H100: 20-layer FlashFFN vs. 24-layer SwiGLU at matched parameters. *Liger* swaps the Liger fused SwiGLU kernel [Hsu et al., 2024] into the baseline model. For eager-SwiGLU prefill we report the speedup only. The two decode comparisons are separate runs, hence the slightly different FlashFFN values.

Prefill, $bs=8$	$L=256$	1024	4096	8192	16384	32768
FlashFFN	113.9K	131.3K	125.7K	109.5K	89.6K	65.4K
Liger SwiGLU	111.3K	125.0K	115.6K	100.0K	79.0K	56.1K
Speedup vs. Liger	$1.02\times$	$1.05\times$	$1.09\times$	$1.09\times$	$1.13\times$	$\mathbf{1.17}\times$
Speedup vs. eager SwiGLU	$1.00\times$	$1.07\times$	$1.10\times$	$1.12\times$	$1.14\times$	$\mathbf{1.18}\times$
Decode (CUDA graphs)	FlashFFN	Eager	Speedup	FlashFFN	Liger	Speedup
$bs=1$, 4K context	572	531	$1.08\times$	576	548	$1.05\times$
$bs=1$, 32K context	302	266	$1.13\times$	305	272	$1.12\times$
$bs=8$, 4K context	2078	1833	$1.13\times$	2064	1842	$1.12\times$
$bs=8$, 32K context	366	308	$\mathbf{1.19}\times$	366	308	$\mathbf{1.19}\times$

materialize the $[B, L, d_{\text{ff}}]$ intermediate, which FlashFFN never writes to HBM. The reduction is uniform across scales (about $2.6\times$ at $L=8064$; Fig. 4), and without the fused kernel the multi-head FFN runs out of memory at 16K tokens (Appendix G).

Model-level throughput. In prefill (Table 4), FlashFFN is at parity with eager SwiGLU at $L=256$ and faster at every longer prompt, up to $1.18\times$ at 32K; decode throughput, measured under CUDA graphs with a dedicated decode-step kernel, is 1.08–$1.19\times$ higher. Against the Liger-fused baseline, FlashFFN is 1.02–$1.17\times$ faster in prefill and 1.05–$1.19\times$ in decode. Per layer, the updated kernel is also at parity or faster at short lengths (1.00–$1.07\times$ for $L=192$–1920; Table 7).

6 Related Work

I/O-aware kernels and the on-chip-memory frontier. The FlashAttention series [Dao et al., 2022, Dao, 2023, Shah et al., 2024] re-cast attention as a memory-bound operation and replaced the HBM-resident softmax score matrix with on-chip tiled accumulation. FlashAttention's per-tile working set obeys the *same* SRAM inequality as our Eq. (3), with the attention-head dimension in place of d_h: attention already enters that bound per-head with $d \le d_h^*$ on Hopper, whereas FFN must first be *made* per-head by an architectural split before the same kernel template applies. FlashFFN is therefore best read as the FFN counterpart of FlashAttention, with a different architectural prerequisite and the same SRAM accounting. Block-sparse and I/O-aware MLP kernels exist (e.g., Liger Kernel [Hsu et al., 2024]) but fuse element-wise operations only, still materializing the full intermediate in HBM, as Cor. 3.2 predicts; Table 3 measures this against Liger, `torch.compile` [Ansel et al., 2024] and xFormers [Lefaudeux et al., 2022]. Concurrent work [Zhang et al., 2026] also targets HBM traffic: a deeply fused kernel for long-context inference cuts HBM traffic and improves cache reuse (up to 13.2% over SGLang on H100). It optimizes the kernel for the unchanged monolithic block, whereas FlashFFN changes the block so that a one-pass on-chip schedule becomes feasible; the two are complementary. Alternative single-kernel schedules (output-dimension tiling, split-K reduction, lower-precision accumulation) can in principle re-fit the monolithic FFN, but each pays a kernel-level cost (extra HBM re-reads of $\mathbf{Q}$, a reduction barrier with doubled output I/O, or numerical precision loss). FlashFFN instead removes the constraint architecturally and pays none of these costs.

Compatibility with parallelism and orthogonal memory optimizations. FlashFFN is an intra-SM memory-geometry fix and is *orthogonal* to (and stackable with) the standard scaling and memory toolchain. Tensor parallelism [Shoeybi et al., 2019] is directly supported in the same form as multi-head attention: partitioning along the head axis distributes $\mathbf{W}_{\text{in}}$ and $\mathbf{W}_{\text{out}}$ across devices with the same all-reduce pattern, requires no kernel changes, and inherits existing TP infrastructure. Pipeline parallelism [Huang et al., 2018] cuts across layers and is unaffected by per-layer geometry. ZeRO [Rajbhandari et al., 2019] and activation checkpointing [Chen et al., 2016] operate at the training-state and time-vs-memory axes respectively, both compatible with FlashFFN's already-reduced per-layer footprint. These techniques optimize different axes (cross-device communication, depth, training memory); FlashFFN optimizes the on-SM working set, and the gains compose.

Multi-head structures in FFN-like networks. Multi-Head MoE (MH-MoE) [Wu et al., 2024] introduced a multi-head FFN split for expressivity; implemented without a fused kernel, such a design materializes H per-head intermediates in HBM, so the split needs a fused kernel to realize its memory advantage. MoEfication [Zhang et al., 2022] retroactively interprets a trained FFN as a sparse MoE without changing the on-chip-memory profile; sparse top-k MoE [Shazeer et al., 2017, Jiang et al., 2024] decouples capacity from per-step compute and is a complementary axis. FlashFFN's per-head sub-networks admit a natural sparse-routing extension that we leave to future work: with H heads each carrying E sub-networks, top-k routing across the shared HE-expert pool (e.g. top-8 of 64 at $H{=}E{=}8$), in the spirit of shared-expert MoE designs [Dai et al., 2024], would recover MH-MoE's expressivity ambition while preserving FlashFFN's SRAM-feasible kernel structure. MH-MoE is the closest architectural prior (split along the model dimension, parallel FFN-like paths, merge projections); FlashFFN differs in its dense per-head sub-networks and in a fused kernel that never materializes the $[B, L, d_{\text{ff}}]$ intermediate. We do not compare against MH-MoE or its rebalanced formulation [Huang et al., 2024], since a sparse model cannot match a dense one on both total parameters and activated FLOPs (Appendix A); our claim is limited to the dense regime, and the sparse extension (FlashMoE) is follow-up work.

7 Conclusion

FlashFFN derives a dense feed-forward block for SwiGLU-based LLMs from a hardware-geometry constraint: under one-pass tiled execution, monolithic SwiGLU FFN with $d_{\text{model}} \geq 2048$ is SRAM-infeasible on every shipping datacenter GPU (Cor. 3.2), and multi-head decomposition is the simplest architectural fix (Cor. 3.3). Combined with sub-network ratio restoration and a fused kernel, FlashFFN achieves a Pareto improvement at matched compute across 128M–1.3B models on The Pile: up to $2.7\times$ lower per-layer peak FFN memory than eager SwiGLU ($1.9\times$ lower than `torch.compile`), up to $1.18\times$ higher prefill and $1.19\times$ higher decode throughput at the model level, lower evaluation PPL, and higher downstream accuracy at $d_h = 128$. The framing generalizes beyond FFN as a lens for architecture-kernel co-design.

Acknowledgments and Disclosure of Funding

We thank Ant Group for providing the computing resources, and Prof. Kewei Tu's group at ShanghaiTech University for providing NVIDIA GPUs for our exploratory experiments. We would like to thank Jiaqi Leng for insightful discussions and wonderful feedback. FlashFFN also builds on the open-source work of the Hugging Face Transformers, Hazy Research ThunderKittens, and OpenAI Triton teams: our models are implemented in Transformers, and our fused kernels are written in ThunderKittens and Triton. We are grateful to these teams for making their work available to the community. The authors declare no competing interests.

References

Jason Ansel, E. Yang, Horace He, N. Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, D. Berard, E. Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, A. Desmaison, Z. DeVito, Elias Ellison, W. Feng, Jiong Gong, Michael Gschwind, B. Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, M. Lezcano, Yan-Bo Liang, J. Liang, Yinghai Lu, Chi-Keung Luk, Bertrand A. Maher, Yunjie Pan, C. Puhrsch, Matthias Reso, Mark-Albert Saroufim, Marcos Yukio Siraichi, Helen Suk, Shun-Ting Zhang, Michael Suo, P. Tillet,

Xu Zhao, Eikan Wang, Ke-Ren Zhou, Richard Zou, Xiao-Dong Wang, Ajit Mathews, W. Wen, G. Chanan, Peng Wu, and Soumith Chintala. PyTorch 2: Faster machine learning through dynamic Python bytecode transformation and graph compilation. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *AAAI Conference on Artificial Intelligence*, 2019. URL https://api.semanticscholar.org/CorpusID:208290939.

Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Martin Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Benqi Wang, and Samuel Weinbach. Gpt-neox-20b: An open-source autoregressive language model. *ArXiv*, abs/2204.06745, 2022. URL https://api.semanticscholar.org/CorpusID:248177957.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *ArXiv*, abs/1604.06174, 2016. URL https://api.semanticscholar.org/CorpusID:15865278.

Damai Dai, Chengqi Deng, Chenggang Zhao, Runxin Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. In *Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://api.semanticscholar.org/CorpusID:266933338.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *ArXiv*, abs/2307.08691, 2023. URL https://api.semanticscholar.org/CorpusID:259936734.

Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *ArXiv*, abs/2405.21060, 2024. URL https://api.semanticscholar.org/CorpusID:270199762.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *ArXiv*, abs/2205.14135, 2022. URL https://api.semanticscholar.org/CorpusID:249151871.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling. *ArXiv*, abs/2101.00027, 2020. URL https://api.semanticscholar.org/CorpusID:230435736.

Isaac Gerber. Attention is not all you need: The importance of feedforward networks in transformer models. *ArXiv*, abs/2505.06633, 2025. URL https://api.semanticscholar.org/CorpusID:278501830.

Mor Geva, R. Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. *ArXiv*, abs/2012.14913, 2020. URL https://api.semanticscholar.org/CorpusID:229923720.

Pin-Lun Hsu, Yun Dai, Vignesh Kothapalli, Qingquan Song, Shao Tang, Siyu Zhu, Steven Shimizu, Shivam Sahni, Haowen Ning, and Yanning Chen. Liger kernel: Efficient triton kernels for llm training. *ArXiv*, abs/2410.10989, 2024.

Shaohan Huang, Xun Wu, Shuming Ma, and Furu Wei. MH-MoE: Multi-head mixture-of-experts. *ArXiv*, abs/2411.16205, 2024.

Yanping Huang, Yonglong Cheng, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, and Z. Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Neural Information Processing Systems*, 2018. URL https://api.semanticscholar.org/CorpusID:53670168.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts. *ArXiv*, abs/2401.04088, 2024. URL https://api.semanticscholar.org/CorpusID:266844877.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard H. Hovy. Race: Large-scale reading comprehension dataset from examinations. In *Conference on Empirical Methods in Natural Language Processing*, 2017. URL https://api.semanticscholar.org/CorpusID:6826032.

Benjamin Lefaudeux, Francisco Massa, Diana Liskovich, Wenhan Xiong, Vittorio Caggiano, Sean Naren, Min Xu, Jieru Hu, Marta Tintore, Susan Zhang, Patrick Labatut, Daniel Haziza, Luca Wehrstedt, Jeremy Reizenstein, and Grigory Sizov. xFormers: A modular and hackable transformer modelling library. https://github.com/facebookresearch/xformers, 2022.

Hector J. Levesque, Ernest Davis, and L. Morgenstern. The winograd schema challenge. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*, 2011. URL https://api.semanticscholar.org/CorpusID:15710851.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Conference on Empirical Methods in Natural Language Processing*, 2018. URL https://api.semanticscholar.org/CorpusID:52183757.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative LLM inference using phase splitting. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2024.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. *ArXiv*, abs/1911.05507, 2019. URL https://api.semanticscholar.org/CorpusID:207930593.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16, 2019. URL https://api.semanticscholar.org/CorpusID:269617042.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande. *Communications of the ACM*, 64:99 – 106, 2019. URL https://api.semanticscholar.org/CorpusID:198893658.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. Socialiqa: Commonsense reasoning about social interactions, 2019. URL https://arxiv.org/abs/1904.09728.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. *ArXiv*, abs/2407.08608, 2024. URL https://api.semanticscholar.org/CorpusID:271098045.

Noam M. Shazeer. Glu variants improve transformer. *ArXiv*, abs/2002.05202, 2020. URL https://api.semanticscholar.org/CorpusID:211096588.

Noam M. Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *ArXiv*, abs/1701.06538, 2017. URL https://api.semanticscholar.org/CorpusID:12462234.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *ArXiv*, abs/1909.08053, 2019. URL https://api.semanticscholar.org/CorpusID:202660670.

Benjamin Frederick Spector, Simran Arora, Aaryan Singhal, Arjun Parthasarathy, Daniel Y. Fu, and Christopher Ré. Thunderkittens: Simple, fast, and adorable kernels. In *International Conference on Learning Representations*, 2025. URL https://api.semanticscholar.org/CorpusID:278498151.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *ArXiv*, abs/2104.09864, 2021. URL https://api.semanticscholar.org/CorpusID:233307138.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *ArXiv*, abs/2302.13971, 2023. URL https://api.semanticscholar.org/CorpusID:257219404.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, Aug 2017a. arXiv:1706.03762v2 [cs.CL].

Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Neural Information Processing Systems*, 2017b. URL https://api.semanticscholar.org/CorpusID:13756489.

Xun Wu, Shaohan Huang, Wenhui Wang, and Furu Wei. Multi-head mixture-of-experts. *ArXiv*, abs/2404.15045, 2024. URL https://api.semanticscholar.org/CorpusID:269302764.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Annual Meeting of the Association for Computational Linguistics*, 2019. URL https://api.semanticscholar.org/CorpusID:159041722.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *ArXiv*, abs/1910.07467, 2019. URL https://api.semanticscholar.org/CorpusID:113405151.

Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. Moefication: Transformer feed-forward layers are mixtures of experts. In *Findings of the Association for Computational Linguistics: ACL*, 2022. URL https://aclanthology.org/2022.findings-acl.71/.

Zixi Zhang, Zhiwen Mo, Yiren Zhao, and Robert Mullins. Deep kernel fusion for transformers. *ArXiv*, abs/2602.11808, 2026.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024.

A Limitations

The main limitation of this work is that we have not yet evaluated FlashFFN under sparse Mixture-of-Experts (MoE) routing. FlashFFN is fully compatible with MoE: each head's E sub-networks already have the structure of dense experts, and the per-head fused kernel is unchanged by routing them sparsely. More interestingly, the multi-head structure unlocks an expressivity gain that standard per-head MoE cannot reach. In the conventional layout, H heads each own a private pool of E experts and route top-k within their own pool ($H \cdot k$ activated experts per token). In FlashFFN, the $H \cdot E$ sub-networks can be *shared across heads*: every head routes top-k over the same global pool of $H \cdot E$ experts. At $H = E = 8$ and $k = 8$ this widens the routable pool from 8 to 64 at identical activated-compute, increasing expressivity without changing FLOPs. We leave a full-scale empirical study of this shared-expert routing to future work.

Scope. Our experiments cover dense, SwiGLU-based decoder LLMs. FFNs with other activations (ReLU, GELU) and sparse MoE models are not evaluated. The SRAM analysis of Section 3 depends only on the accumulator width of the one-pass schedule, but we make no empirical claim beyond SwiGLU.

Depth under the matched-budget protocol. At 1.3B we compare a 20-layer FlashFFN model with a 24-layer SwiGLU baseline at matched total parameters and FLOPs, the standard protocol for cross-architecture comparisons [Dao and Gu, 2024]. When the block changes, single-axis alignments (iso-depth, iso-width, iso-d_{ff}) cannot all hold together with the total parameter and FLOP budget, so we rebalance depth. Our claim is therefore that, within a fixed budget, the FlashFFN family contains a model that is both better and faster than the SwiGLU baseline. The MH-FFN control (20 layers at 370M, no gain) and the d_h sweep at fixed depth and parameter count (1.3B PPL between 16.33 and 17.27) indicate that the block design drives the gain, but we did not run a 24-vs-24 comparison, which would separate block design from depth rebalancing more cleanly.

No matched MH-MoE baseline. We do not compare against MH-MoE [Wu et al., 2024] or its rebalanced formulation [Huang et al., 2024]. Sparse top-k routing decouples total parameters from activated FLOPs: at matched FLOPs MH-MoE exceeds our parameter budget, and at matched parameters it activates fewer FLOPs per token, so either choice confounds the comparison. The rebalanced formulation addresses expert-width accounting between sparse models, not this dense-versus-sparse mismatch. We therefore claim only that, at a matched budget, FlashFFN improves on the dense SwiGLU baseline, and leave the sparse comparison to follow-up work.

Efficiency measurements. All efficiency results are measured on H100. We report prefill and decode separately and do not profile mixed prefill+generation batches. Production serving increasingly places the two phases on separate instances (P/D disaggregation; Zhong et al., 2024, Patel et al., 2024), a setup FlashFFN is compatible with, since each instance then runs one of the two phases we measure. Quality results are single training runs without error bars.

B Mechanism of I/O-Aware FFN and the Necessity of Multi-Head Design

In this section, we provide a detailed numerical analysis of the I/O-aware fused kernel (the "Flash" mechanism) employed in FlashFFN. Furthermore, we demonstrate that the Multi-Head architecture is not merely an expressivity enhancement but a strict hardware prerequisite to enable this memory-efficient computation on modern GPUs due to SRAM capacity constraints.

B.1 Concrete Calculation Example

To illustrate the memory savings of the I/O-aware kernel, we present a concrete calculation example. We assume a simplified setting with sequence length $L = 2$, model dimension $d_{\text{model}} = 3$, and intermediate dimension $d_{\text{ff}} = 6$. We use a tile size of $b = 2$ for the intermediate dimension.

Let the input Q and parameters K, V be defined as follows (using ReLU for simplicity in this manual calculation):

$$Q = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \quad K^\top = \begin{bmatrix} 1 & 0 & 0 & -2 & 0 & 3 \\ 0 & 1 & 0 & 2 & -3 & 0 \\ 0 & 0 & 2 & 0 & -3 & -3 \end{bmatrix}, \quad V = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \\ 2 & 2 & 0 \\ 0 & 3 & 3 \\ 3 & 0 & 3 \end{bmatrix}$$

The target computation is $O = \text{ReLU}(Q \cdot K^\top) \cdot V$.

B.1.1 Approach 1: Naive Algorithm (HBM Bound)

In the standard approach, the full intermediate activation matrix $A \in \mathbb{R}^{L \times d_{\text{ff}}}$ is computed first and materialized in High Bandwidth Memory (HBM).

$$A = Q \cdot K^\top = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -2 & 0 & 3 \\ 0 & 1 & 0 & 2 & -3 & 0 \\ 0 & 0 & 2 & 0 & -3 & -3 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 0 & 2 & -6 & 3 \\ 0 & 1 & 2 & 2 & -6 & -3 \end{bmatrix}$$

This large matrix A consumes $\mathcal{O}(L \cdot d_{\text{ff}})$ memory. Subsequently, the activation and second multiplication are applied:

$$O = \text{ReLU}(A) \cdot V = \begin{bmatrix} 1 & 2 & 0 & 2 & 0 & 3 \\ 0 & 1 & 2 & 2 & 0 & 0 \end{bmatrix} \cdot V = \begin{bmatrix} 14 & 6 & 9 \\ 4 & 5 & 4 \end{bmatrix}$$

Bottleneck: The materialization of A creates high peak memory usage and requires extensive HBM Read/Write operations.

B.1.2 Approach 2: I/O-Aware Online Algorithm (SRAM Bound)

The "Flash" algorithm computes the output block-by-block without ever materializing the full matrix A in HBM. We iterate over blocks of the intermediate dimension d_{ff}.

Step 1: Compute using Columns 1–2 of $K^\top$ and Rows 1–2 of V. These sub-matrices are small enough to be loaded into on-chip SRAM. We compute the partial contribution to the output accumulator:

$$O_{\text{part1}} = \text{ReLU}\left(\begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \right) \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Step 2: Compute using Columns 3–4 of $K^\top$ and Rows 3–4 of V.

$$O_{\text{part2}} = \text{ReLU}\left(\begin{bmatrix} 0 & 2 \\ 2 & 2 \end{bmatrix} \right) \cdot \begin{bmatrix} 0 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 4 & 4 & 0 \\ 4 & 4 & 4 \end{bmatrix}$$

Step 3: Compute using Columns 5–6 of $K^\top$ and Rows 5–6 of V.

$$O_{\text{part3}} = \text{ReLU}\left(\begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 3 \\ -3 & 0 \\ -3 & -3 \end{bmatrix} \right) \cdot \begin{bmatrix} 0 & 3 & 3 \\ 3 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 9 & 0 & 9 \\ 0 & 0 & 0 \end{bmatrix}$$

Final Accumulation: The final output is the sum of partial results, accumulated entirely within SRAM.

$$O = O_{\text{part1}} + O_{\text{part2}} + O_{\text{part3}} = \begin{bmatrix} 1+4+9 & 2+4+0 & 0+0+9 \\ 0+4+0 & 1+4+0 & 0+4+0 \end{bmatrix} = \begin{bmatrix} 14 & 6 & 9 \\ 4 & 5 & 4 \end{bmatrix}$$

Result: The final output is identical to the naive approach, but the intermediate matrix A was never stored in HBM.

B.2 The Hardware Necessity of Multi-Head Design

A critical question arises: *why can we not apply this I/O-aware algorithm directly to a standard, monolithic FFN?* The answer lies in the hardware limitations of the GPU SRAM.

In the Online Algorithm described above, we must maintain an output accumulator matrix O_{acc} in SRAM. For a batch of tokens with block size L_{tile}, the shape of this accumulator is $\text{Shape}(O_{\text{acc}}) = [L_{\text{tile}}, d_{\text{model}}]$.

Consider a standard large model setting on an NVIDIA H100 GPU with model dimension $d_{\text{model}} = 4096$, minimum efficient sequence tile size $L_{\text{tile}} \approx 128$ (to saturate tensor cores), and Float32 for accumulation stability.

The memory required for the accumulator is $128 \times 4096 \times 4$ bytes $= 2$ MB, while the streaming multiprocessor (SM) shared memory (SRAM) on an H100 is only **228 KB**. The accumulator for a monolithic FFN is nearly **10x larger** than the available on-chip memory. This makes applying the Flash algorithm to a standard FFN physically impossible without causing register spilling, which negates the performance gains.

The Solution: Multi-Head Decomposition. By adopting the Multi-Head architecture proposed in FlashFFN, we decompose d_{model} into H heads of size d_h (e.g., $d_h = 128$). The accumulator shape becomes $\text{Shape}(O_{\text{acc}}^{\text{head}}) = [L_{\text{tile}}, d_h]$. The memory requirement per head drops to $128 \times 128 \times 4$ bytes $= 64$ KB, which fits comfortably within the 228 KB SRAM limit, leaving room for input tiles and other buffers. Therefore, the Multi-Head design is not merely an architectural choice for expressivity, but a **necessary condition** to enable memory-efficient, I/O-aware "Flash" computation on current hardware.

C Triton Pseudo Code for FlashFFN

Triton code can run efficiently on consumer-grade GPUs (like RTX 3090) but not on Hopper cards.

Algorithm 1 FLASHFFN-FORWARD-TRITON

Require: $Q \in \mathbb{R}^{B\times H\times L\times d_h}$, $K, U, V \in \mathbb{R}^{H\times E\times d_e\times d_h}$, $R \in \mathbb{R}^{B\times H\times L\times E}$

Strategy: Parallelize over batch, head, and sequence blocks. Inner loops iterate over sub-networks and intermediate dimension blocks.

Meta: `BLOCK_SEQ`, `BLOCK_INTER`, `HEAD_DIM`$= d_h$, `PFFN_DIM`$= d_e$, `NUM_PFFN`$= E$

Grid: `pid0`$\leftarrow$seq_block, `pid1`$\leftarrow$head h, `pid2`$\leftarrow$batch b

$s_0 \leftarrow$ `pid0`$\cdot$`BLOCK_SEQ`
$Q_{\text{blk}} \leftarrow Q[b,\ h,\ s_0 : s_0{+}$`BLOCK_SEQ`$,\ :]$ ▷ masked at sequence tail
$O_{\text{acc}} \leftarrow \mathbf{0}^{\text{BLOCK_SEQ}\times d_h}$ ▷ SRAM accumulator
for $e = 0$ **to** $E{-}1$ **do**
  $R_{\text{rows}} \leftarrow R[b,\ h,\ s_0 : s_0{+}$`BLOCK_SEQ`$,\ e]$
  for $m = 0$ **to** $d_e{-}1$ **step** `BLOCK_INTER` **do**
    $K_{\text{tile}} \leftarrow K[h,\ e,\ m{:}m{+}$`BLOCK_INTER`$,\ :]^\top$
    $U_{\text{tile}} \leftarrow U[h,\ e,\ m{:}m{+}$`BLOCK_INTER`$,\ :]^\top$
    $V_{\text{tile}} \leftarrow V[h,\ e,\ m{:}m{+}$`BLOCK_INTER`$,\ :]$
    $M \leftarrow Q_{\text{blk}} \cdot K_{\text{tile}}; \quad N \leftarrow Q_{\text{blk}} \cdot U_{\text{tile}}$
    $A \leftarrow \text{SiLU}(M) \odot N; \quad A \leftarrow A \odot R_{\text{rows}}$
    $O_{\text{acc}} \leftarrow O_{\text{acc}} + A \cdot V_{\text{tile}}$
  end for
end for
$O[b,\ h,\ s_0 : s_0{+}$`BLOCK_SEQ`$,\ :] \leftarrow O_{\text{acc}}$

Algorithm 2 FLASHFFN-BACKWARD-TRITON (DQ, DR)

Require: saved Q, K, U, V, R; incoming $dS \in \mathbb{R}^{B\times H\times L\times d_h}$
Grid/Meta as in Alg 1
$s_0 \leftarrow$ `pid0`$\cdot$`BLOCK_SEQ`; $h \leftarrow$ `pid1`; $b \leftarrow$ `pid2`
$Q_{\text{blk}} \leftarrow Q[b,\ h,\ s_0 : s_0{+}$`BLOCK_SEQ`$,\ :]$; $dS_{\text{blk}} \leftarrow dS[b,\ h,\ s_0 : s_0{+}$`BLOCK_SEQ`$,\ :]$
$dQ_{\text{acc}} \leftarrow \mathbf{0}^{\texttt{BLOCK_SEQ}\times d_h}$; $dR_{\text{rows}} \leftarrow \mathbf{0}^{\texttt{BLOCK_SEQ}\times 1}$
for $e = 0$ **to** $E{-}1$ **do**
  $R_{\text{rows}} \leftarrow R[b,\ h,\ s_0 : s_0{+}$`BLOCK_SEQ`$,\ e]$
  for $m = 0$ **to** $d_e{-}1$ **step** `BLOCK_INTER` **do**
    $K_{\text{tile}} \leftarrow K[h, e, m{:}m{+}$`BLOCK_INTER`$, :]$;
    $U_{\text{tile}} \leftarrow U[h, e, m{:}m{+}$`BLOCK_INTER`$, :]$;
    $V_{\text{tile}}^{\top} \leftarrow V[h, e, m{:}m{+}$`BLOCK_INTER`$, :]^{\top}$
    $M \leftarrow Q_{\text{blk}} \cdot K_{\text{tile}}^{\top}$; $N \leftarrow Q_{\text{blk}} \cdot U_{\text{tile}}^{\top}$
    $\text{sig} \leftarrow \text{sigmoid}(M)$; $\text{SiLU}(M) \leftarrow M \odot \text{sig}$
    $dA \leftarrow dS_{\text{blk}} \cdot V_{\text{tile}}^{\top}$
    dR_{rows} += $\sum_{\text{cols}}\big(dA \odot \text{SiLU}(M) \odot N\big)$
    $dM \leftarrow \big(dA \odot (R_{\text{rows}} \odot N)\big) \odot \big(\text{sigmoid}(M) + M \cdot \text{sigmoid}(M) \cdot (1 - \text{sigmoid}(M))\big)$
    $dN \leftarrow \big(dA \odot \text{SiLU}(M)\big) \odot R_{\text{rows}}$
    dQ_{acc} += $dM \cdot K_{\text{tile}}$ + $dN \cdot U_{\text{tile}}$
  end for
  $dR[b,\ h,\ s_0 : s_0{+}$`BLOCK_SEQ`$,\ e] \leftarrow dR_{\text{rows}}$
end for
$dQ[b,\ h,\ s_0 : s_0{+}$`BLOCK_SEQ`$,\ :] \leftarrow dQ_{\text{acc}}$

Algorithm 3 FLASHFFN-BACKWARD-TRITON (DK, DU, DV)

Require: saved Q, K, U, V, R; dS
Grid: `pid0`$\leftarrow e \cdot (d_e/$`BLOCK_INTER`$) + \lfloor m/$`BLOCK_INTER`$\rfloor$, `pid1`$\leftarrow h$
decode (e, m) from `pid0`
$K_{\text{tile}} \leftarrow K[h, e, m{:}m{+}$`BLOCK_INTER`$, :]$; $U_{\text{tile}} \leftarrow U[h, e, m{:}m{+}$`BLOCK_INTER`$, :]$; $V_{\text{tile}}^{\top} \leftarrow V[h, e, m{:}m{+}$`BLOCK_INTER`$, :]^{\top}$
$dK_{\text{acc}}, dU_{\text{acc}}, dV_{\text{acc}} \leftarrow \mathbf{0}^{\texttt{BLOCK_INTER}\times d_h}$
for $b = 0$ **to** $B{-}1$ **do**
  for $s_0 = 0$ **to** $L{-}1$ **step** `BLOCK_SEQ` **do**
    $Q_{\text{blk}} \leftarrow Q[b,\ h,\ s_0 : s_0{+}$`BLOCK_SEQ`$,\ :]$;
    $R_{\text{rows}} \leftarrow R[b,\ h,\ s_0 : s_0{+}$`BLOCK_SEQ`$,\ e]$;
    $dS_{\text{blk}} \leftarrow dS[b,\ h,\ s_0 : s_0{+}$`BLOCK_SEQ`$,\ :]$
    $M \leftarrow Q_{\text{blk}} \cdot K_{\text{tile}}^{\top}$; $N \leftarrow Q_{\text{blk}} \cdot U_{\text{tile}}^{\top}$
    $\text{sig} \leftarrow \text{sigmoid}(M)$; $\text{SiLU}(M) \leftarrow M \odot \text{sig}$; $\widetilde{N} \leftarrow R_{\text{rows}} \odot N$
    $A^{\top} \leftarrow (\text{SiLU}(M) \odot \widetilde{N})^{\top}$; $dA \leftarrow dS_{\text{blk}} \cdot V_{\text{tile}}^{\top}$
    $dM \leftarrow (dA \odot \widetilde{N}) \odot \big(\text{sigmoid}(M) + M \cdot \text{sigmoid}(M) \cdot (1 - \text{sigmoid}(M))\big)$;
    $dN \leftarrow (dA \odot \text{SiLU}(M)) \odot R_{\text{rows}}$
    dV_{acc} += $A^{\top} \cdot dS_{\text{blk}}$
    dK_{acc} += $dM^{\top} \cdot Q_{\text{blk}}$; dU_{acc} += $dN^{\top} \cdot Q_{\text{blk}}$
  end for
end for
$dK[h, e, m{:}m{+}$`BLOCK_INTER`$, :] \leftarrow dK_{\text{acc}}$; $dU[h, e, m{:}m{+}$`BLOCK_INTER`$, :] \leftarrow dU_{\text{acc}}$; $dV[h, e, m{:}m{+}$`BLOCK_INTER`$, :] \leftarrow dV_{\text{acc}}$

D Hopper/ThunderKittens [Spector et al., 2025] Pseudocode for FlashFFN

Algorithm 4 FLASHFFN-FORWARD-TK (HOPPER)

Require: $Q \in \mathbb{R}^{B\times H\times L\times d_h}$, $K, U, V \in \mathbb{R}^{H\times E\times d_e\times d_h}$, $R \in \mathbb{R}^{B\times H\times E\times L}$
Meta: `BLOCK_SEQ`, `BLOCK_INTER`, `NUM_STAGES`, `CON_WARPGRPS` $\geq$ 2, `PROD_WARPGRPS` $=$ 1, d_h=128, $d_e \bmod$ `BLOCK_INTER` $= 0$
Grid: $x = \lceil L/(\texttt{BLOCK_SEQ} \cdot \texttt{CON_WARPGRPS})\rceil$, $y = H$, $z = B$
Allocate stage/ring buffers in SRAM for Q (per consumer), R (per consumer), and K, U, V (per stage).
Warmup (producer): prefetch the Q tiles for all consumers in this x-block; prefetch R for subnet e=0; prefetch the first `NUM_STAGES` (K, U, V) inter-tiles.

Producer loop (over inter-tiles):
for inter_tile $=$ `NUM_STAGES`, $\ldots, E \cdot (d_e/\texttt{BLOCK_INTER}) - 1$ **do**
    wait for consumers to finish the target stage; then prefetch the next (K, U, V) tiles into that stage.
    if inter_tile is the first tile of a new subnet e **then**
        prefetch router R rows for all consumers (current x-block).
    end if
end for

Each consumer warpgroup $c \in \{0, \ldots, \texttt{CON_WARPGRPS} - 1\}$ (independent, identical):
load its Q tile; set $O_{\text{acc}} \leftarrow 0$.
for inter_tile $= 0, \ldots, E \cdot (d_e/\texttt{BLOCK_INTER}) - 1$ **do**
    wait until producer has filled the current stage with (K, U, V); if this tile starts a new subnet, wait for R.
    $M \leftarrow Q_{\text{blk}} K_{\text{tile}}^{\top}$; $N \leftarrow Q_{\text{blk}} U_{\text{tile}}^{\top}$
    $S \leftarrow \text{SiLU}(M) \odot N$; $S \leftarrow S \odot r$ ▷ apply router row r
    $O_{\text{acc}} \leftarrow O_{\text{acc}} + S\, V_{\text{tile}}$
    signal producer that this stage can be reused.
end for
store O_{acc} to global output.

E Training Hyperparameters

All models trained at a particular size use the optimizer hyperparameters set in Table 5.

Table 5: Training hyperparameters by model scale.

	128M	**370M**	**1.3B**
Learning rate	3e-3	1.5e-3	1e-3
Learning scheduler	cos_with_min_lr	cos_with_min_lr	cos_with_min_lr
Min LR	1e-5	1e-5	1e-5
Warmup ratio	0.015	0.015	0.015
Adam β_1	0.9	0.9	0.9
Adam β_2	0.95	0.95	0.95
Weight decay	1e-1	1e-1	1e-1
Total batch size	64	64	64
Segment length (tokens)	4096	4096	4096
Training steps	245K	245K	409K

F Training Configurations

Table 6: Model and training configurations across scales and variants. **Alignment protocol.** We strictly match total parameter count and total training FLOPs between each FlashFFN model and its SwiGLU baseline at the same scale (128M/370M/1.3B columns are paired); the matched parameter and step counts are highlighted in the corresponding rows. Because FlashFFN introduces a head-fusion/splitting projection at every block (the per-head pathway $d_h \to E \cdot d_e \to d_h$ is realized as additional $\mathbf{K}_e^h, \mathbf{U}_e^h, \mathbf{V}_e^h$ projections that are absent in the SwiGLU FFN), some structural hyper-parameters *cannot* be held fully identical to the baseline: the FlashFFN n_{layers}, $n_{\text{ffn-heads}}\, H$, sub-network count E, and the rounded "Intermediate size" are chosen so that the resulting parameter count and per-step FLOPs land at the SwiGLU target. We surface this trade-off explicitly: it is an unavoidable consequence of the architectural change, not an apples-to-oranges comparison; every other knob (optimizer, schedule, tokens, d_{model}, attention head count/d_{head}, regularization, RoPE) is held fixed.

	Baseline			**MH-FFN**		**PKV**	**FlashFFN**		
	128M	**370M**	**1.3B**	**128M**	**370M**	**370M**	**128M**	**370M**	**1.3B**
Params (M)	117.96 M	388.05 M	1.323 B	115.70 M	379.79 M	386.55 M	115.77 M	390.08 M	1.321 B
n_{layers}	12	24	24	10	20	24	10	21	20
d_{model}	768	1024	2048	768	1024	1024	768	1024	2048
$n_{\text{att-heads}}/d_{\text{head}}$	12/64	16/64	32/64	12/64	16/64	16/64	12/64	16/64	32/64
Intermediate size	2048	2752	5504	2048	2752	3072	2048	2688	5760
$n_{\text{ffn-heads}}(H)/d_h$	-/-	-/-	-/-	6/128	8/128	8/128	6/128	8/128	16/128
Sub-network Num E	-	-	-	-	-	-	8	7	15
Training steps	245K	245K	409K	245K	245K	245K	245K	245K	409K
Learning rate	3e-3	1.5e-3	1e-3	3e-3	1.5e-3	1.5e-3	3e-3	1.5e-3	1e-3
Training tokens	60B	60B	100B	60B	60B	60B	60B	60B	100B
Hidden act	`silu`	`silu`	`silu`	`silu`	`silu`	`silu`	`silu`	`silu`	`silu`
`initializer_range`	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
`max_position_embeddings`	4096	4096	4096	4096	4096	4096	4096	4096	4096
`pretraining_tp`	1	1	1	1	1	1	1	1	1
`rms_norm_eps`	`1e-05`	`1e-05`	`1e-05`	`1e-05`	`1e-05`	`1e-05`	`1e-05`	`1e-05`	`1e-05`
`rope_scaling`	`null`	`null`	`null`	`null`	`null`	`null`	`null`	`null`	`null`
`tie_word_embeddings`	`true`	`false`	`false`	`true`	`false`	`false`	`true`	`false`	`false`
`torch_dtype`	`bfloat16`	`bfloat16`	`bfloat16`	`bfloat16`	`bfloat16`	`bfloat16`	`bfloat16`	`bfloat16`	`bfloat16`
`vocab_size`	50432	50432	50432	50432	50432	50432	50432	50432	50432

Algorithm 5 FLASHFFN-BACKWARD-TK (HOPPER)

Require: saved Q, K, U, V, R; upstream dO
Meta: `BLOCK_SEQ`=`BLOCK_INTER`, `NUM_STAGES`$= 2$, two consumer warpgroups, one producer
Grid: $x = \lceil (E \cdot d_e)/(2 \cdot \texttt{BLOCK_INTER}) \rceil,\ y = H,\ z = B$

Assign each consumer a distinct inter-tile (A/B). Allocate per-stage SRAM for Q, dO, R; per-consumer SRAM for its (K, U, V) tile; small scratch for partial dQ/dR exchange.
Warmup (producer): prefetch (K, U, V) for both consumers' inter-tiles; prefetch first-stage Q, dO, and subnet-e router R.

Producer loop (over sequence tiles):
for each sequence tile t **do**
    wait for consumers to release stage $t \bmod 2$; prefetch $Q[b,h,t]$, $dO[b,h,t]$, and $R[b,h,e,t]$ for that stage.
end for

Consumer warpgroup #0 (inter-tile A):
init $dK_{\mathrm{acc}}, dU_{\mathrm{acc}}, dV_{\mathrm{acc}} \leftarrow 0$.
for each sequence tile t **do**
    wait for producer to provide Q, dO, R for stage $t \bmod 2$; use preloaded (K, U, V) for A.
    $M \leftarrow Q_{\mathrm{blk}} K_{\mathrm{A}}^{\top}$; $N \leftarrow Q_{\mathrm{blk}} U_{\mathrm{A}}^{\top}$
    $A \leftarrow \mathrm{SiLU}(M)$; $dA \leftarrow dO_{\mathrm{blk}} V_{\mathrm{A}}^{\top}$
    $A' \leftarrow A + \mathrm{sigmoid}(M) \cdot (1 - A)$
    $dR_{\mathrm{rows}} \mathrel{+}= \mathrm{row_sum}(dA \odot A \odot N)$
    $dN \leftarrow (dA \odot A) \odot r$; $dM \leftarrow (dA \odot (N \odot r)) \odot A'$
    $dQ^{(A)} \leftarrow dN\, U_{\mathrm{A}} + dM\, K_{\mathrm{A}}$
    $dK_{\mathrm{acc}} \mathrel{+}= dM^{\top} Q_{\mathrm{blk}}$; $dU_{\mathrm{acc}} \mathrel{+}= dN^{\top} Q_{\mathrm{blk}}$
    $A_{\mathrm{gated}} \leftarrow A \odot (N \odot r)$; $dV_{\mathrm{acc}} \mathrel{+}= A_{\mathrm{gated}}^{\top} dO_{\mathrm{blk}}$
    write $dQ^{(A)}$ and dR_{rows} to a shared slot; notify peer.
end for
store-add $dK_{\mathrm{acc}}, dU_{\mathrm{acc}}, dV_{\mathrm{acc}}$ to global.

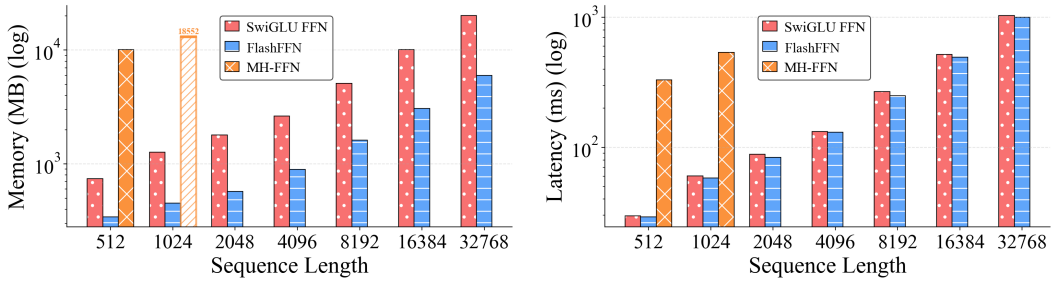
Consumer warpgroup #1 (inter-tile B):
same loop with (K, U, V) for B, producing $dQ^{(B)}$, $dR^{(B)}_{\mathrm{rows}}$, and its own $dK_{\mathrm{acc}}, dU_{\mathrm{acc}}, dV_{\mathrm{acc}}$.
at each t: wait for peer's $dQ^{(A)}$/$dR^{(A)}$, then
    $dQ[b,h,t] \mathrel{+}= dQ^{(A)} + dQ^{(B)}$, $dR[b,h,e,t] \mathrel{+}= dR^{(A)} + dR^{(B)}$,
then free the shared slot so producer can reuse the stage.
store-add its $dK_{\mathrm{acc}}, dU_{\mathrm{acc}}, dV_{\mathrm{acc}}$ to global.

G Efficiency Benchmark Configuration Details on Hopper

Timing scope. Per-layer latency and memory cover the full FFN replacement block: $\mathbf{W}_{\mathrm{in}}$, routing, the fused per-head kernel, head concatenation and $\mathbf{W}_{\mathrm{out}}$. The SwiGLU baseline uses $d_{\mathrm{ff}} = 5504$, the trained 1.3B configuration of Table 6. Table 9 is the full single-layer sweep with the originally released kernel, which was slower than SwiGLU for $L \leq 1536$. The released kernel has since been updated to run faster at short sequence lengths, and decoding now uses a dedicated decode-step kernel. Table 7 gives the updated per-layer speedups at short lengths, and Table 8 gives per-layer results at the 128M and 370M configurations. The final memory comparison against fused SwiGLU implementations is Table 3; where its values differ from Table 9, all memory claims in the paper use Table 3. Model-level throughput is in Table 4. Fig. 4 uses per-layer peak memory measured at the trained configurations ($bs = 8$, $L = 8064$): FlashFFN vs. SwiGLU is 348 vs. 903, 460 vs. 1210, and 931 vs. 2384 MB at 128M, 370M and 1.3B.

Table 7: Per-layer speedup of FlashFFN over SwiGLU at short sequence lengths with the updated kernel (H100, $bs = 8$, same FFN configuration as Table 9).

L	192	384	768	1536	1920
Speedup	1.00×	1.03×	1.04×	1.05×	1.07×



(a) Memory vs. Sequence Length

(b) Latency vs. Sequence Length

Figure 5: Single-layer memory and latency vs. sequence length on H100 (log-scale), from an earlier measurement with the originally released kernel. FlashFFN uses less peak memory than SwiGLU at every length. The final memory comparison, under one protocol for all implementations, is Table 3; the updated kernel's short-sequence latency is in Table 7. The multi-head FFN *without* the fused kernel (included as a sanity reference for what a pure-PyTorch multi-head implementation produces) exhibits the linear-in-H memory blow-up that Corollary 3.3 would otherwise hide, OOMing at 16K tokens.

Table 8: Per-layer latency and peak memory at the trained 128M and 370M FFN configurations on H100.

Scale	FlashFFN (ms)	SwiGLU (ms)	FlashFFN (MB)	SwiGLU (MB)
128M	0.60	0.68	71	106
370M	1.03	1.11	95	143

Table 9: FlashFFN vs. SwiGLU vs. MH-FFN across sequence lengths on **Hopper**, with the originally released kernel.

Arch	**bs**	L	d_e	H	E	d_h	**Latency (ms)**			**Memory (MB)**		
							FlashFFN	**SwiGLU**	**MH-FFN**	**FlashFFN**	**SwiGLU**	**MH-FFN**
Hopper	8	192	384	16	15	128	1.05	0.66	5.93	82.5	156.1	1185.5
Hopper	8	384	384	16	15	128	1.54	1.32	8.97	129.1	213.5	2271.5
Hopper	8	768	384	16	15	128	2.50	2.48	14.73	158.8	327.5	4443.5
Hopper	8	1536	384	16	15	128	5.05	4.82	26.93	218.0	555.5	8787.5
Hopper	8	1920	384	16	15	128	6.00	6.35	32.97	247.6	665.8	10959.5
Hopper	8	2880	384	16	15	128	8.75	9.40	47.58	321.7	951.5	16389.5
Hopper	8	4032	384	16	15	128	11.74	13.06	57.96	410.6	1288.6	22905.5
Hopper	8	8064	384	16	15	128	19.36	25.49	126.17	721.6	2477.8	45711.5
Hopper	8	16128	384	16	15	128	47.05	50.00	OOM	1343.8	4857.5	OOM
Hopper	8	32832	384	16	15	128	95.73	102.90	OOM	2632.4	9783.5	OOM

H Detailed Experimental Setup

In this section, we provide the granular details of our experimental configuration to facilitate reproducibility.

H.1 Training Configuration

All models are pre-trained with a global batch size of 64 and a sequence length of 4,096. We use the AdamW optimizer. The learning rate schedules, peak learning rates, and weight decay settings are synchronized across all models within the same parameter scale to ensure fair comparison. Detailed optimizer hyperparameters are listed in Table 5.

H.2 Model Architecture Details

H.2.1 General Settings.

Our backbone architecture follows a Llama-like design [Touvron et al., 2023].

Tokenizer: We use the GPT-NeoX [Black et al., 2022] tokenizer with a vocabulary size of $50{,}432$.

Positional Embeddings: We employ Rotary Position Embeddings (RoPE) [Su et al., 2021] with base θ=10,000.

Regularization: We disable both attention dropout and FFN bias throughout all layers.

Normalization: RMSNorm is applied before each attention and FFN layer. We generally follow the architectural settings and hyperparameters provided in the appendix of Dao and Gu [2024].

H.2.2 FlashFFN Configuration.

For our FlashFFN models, we determine the configuration of MoE-like sub-networks to maintain the standard SwiGLU expansion ratio. Specifically, for a given head dimension d_h, we set the dimension per sub-network $d_e \approx \frac{8}{3} d_h$, rounded up to the nearest multiple of 64 for hardware efficiency. The number of sub-networks E is then chosen so that the FFN-block parameter count $3\,H\,E\,d_e\,d_h$ matches the SwiGLU baseline $3\,d_{\text{ff}}\,d_{\text{model}}$. Equivalently, since $d_h = d_{\text{model}}/H$, this gives $E\,d_e = d_{\text{ff}}$, which is the value tabulated in the "Intermediate size" row of Table 6 for the FlashFFN columns.

H.2.3 Baseline Specifics.

Parametric KV: This baseline approximates an extreme case of our architecture. It replaces the FFN with a Multi-Head Attention block where Q comes from the input stream, but K and V are learnable parameters independent of the input. This is architecturally equivalent to setting the sub-network dimension $d_e = 1$.

MH-FFN: This model uses the naive multi-head split without the MoE-like sub-network aggregation. We adjust the layer depth and intermediate width d_{ff} to match the FLOPs and parameter count of the SwiGLU baseline.

Detailed model configurations, including layer counts, hidden dimensions, and head structures for all scales (128M, 370M, 1.3B), are tabulated in Table 6.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Section 1 state three contributions: (i) a formal SRAM-feasibility framework for FFN computation under one-pass tiled execution (Section 3, Proposition 3.1); (ii) FlashFFN as the simplest architectural transform satisfying this constraint (Section 4); and (iii) a Pareto improvement over a SwiGLU baseline under strictly matched parameter counts and FLOPs (Section 5). Scope qualifications (dense SwiGLU-based LLMs, the one-pass tiled execution model, the strict parameter and FLOPs alignment of every comparison, and the 1.3B-parameter / 100B-token compute cap) are stated explicitly in Section 1, Section 3, Section 5, and Appendix A.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: A dedicated Limitations section is provided in Appendix A. It identifies sparse Mixture-of-Experts routing as the main unevaluated direction: FlashFFN's per-head sub-networks are structurally compatible with MoE, and the multi-head structure further enables a shared-expert routing scheme in which all heads route over a global $H \cdot E$-expert pool (e.g., from 8 to 64 experts at identical activated-compute), strictly increasing expressivity. A full empirical study of this shared-expert routing is left to future work. The section also states the scope of the experiments (dense SwiGLU-based LLMs), the matched-budget depth protocol (20 vs. 24 layers at 1.3B) and the missing iso-depth control, the absence of a matched MH-MoE comparison, and that efficiency is measured on H100 only.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.

- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: Proposition 3.1 states its assumptions explicitly (one-pass tiled accumulation along Eq. (2), fp32 output accumulator, tile sizes B_q, B_k) and is followed by a complete proof that enumerates all SRAM-resident tiles, sums byte counts, and bounds by the per-SM SRAM budget. Corollaries 3.2 and 3.3 are proven by direct substitution into the proposition; both proofs are given in-line in Section 3.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Architecture, optimizer, learning-rate schedule, batch size, sequence length, token budget, tokenizer, and full per-scale model configurations (parameter count, layer count, d_{model}, d_h, H, E, d_e) for every variant (Baseline, MH-FFN, Dense-MoE, PKV, FlashFFN) are reported in Appendix H and Tables 5, 6. The fused kernel is given as Algorithms 1–5 (Triton + Hopper/ThunderKittens). Datasets used (THE PILE, PG19, and the six downstream benchmarks) are publicly available with standard splits.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.

- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The reference implementation of FlashFFN (PyTorch model + Triton and Hopper/ThunderKittens kernels) is publicly available at `https://github.com/alexzms/FlashMHF` under the Apache License 2.0. Datasets used (THE PILE, PG19) and the six downstream benchmarks are publicly available; tokenizer (GPT-NeoX) is publicly licensed. Pseudocode for forward and backward kernels is also included in the appendix.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Optimizer (AdamW), peak learning rate, learning-rate schedule, weight decay, batch size, sequence length (4096), training-token budget (60B for 128M/370M, 100B for 1.3B), tokenizer (GPT-NeoX, vocab 50432), positional encoding (RoPE, $\theta = 10000$), normalization (RMSNorm), regularization choices, and per-variant model configurations

are all reported in Appendix H (Tables 5, 6). Evaluation splits and benchmarks are listed in Section 5.2.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: Pretraining at 128M, 370M, and 1.3B parameters with 60–100B tokens per run is too computationally expensive for multi-seed reruns within our compute budget. We therefore report single-run results. The analysis emphasizes (a) consistent direction across three independent model scales (128M, 370M, 1.3B; Section 5.1), (b) consistent ranking of the optimal $d_h = 128$ variant on both PG19 evaluation loss and a six-benchmark downstream average (Tables 1, 2), and (c) mechanistic-attribution controls (MH-FFN, Dense-MoE, PKV) at 370M whose direction matches the predictions of Section 4.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: Training token budgets (60B for 128M/370M, 100B for 1.3B) and training-step counts (245K, 245K, 409K respectively) are reported in Appendix H and Table 6. Inference benchmarks were measured on a single NVIDIA H100 SXM (80 GB HBM); the full per-sequence-length latency and memory tabulation is given in Appendix G, Table 9; model-level throughput and the fused-baseline memory comparison are in Tables 4 and 3.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.

- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work is a foundational architecture-and-kernel contribution. We do not collect new data, do not involve human subjects, do not use personally identifying information, and use only publicly licensed datasets and tokenizers (THE PILE, PG19, GPT-NeoX, plus standard public NLP-eval suites). Authors have reviewed the NeurIPS Code of Ethics and the work conforms to it.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Positive: hardware-aware FFN designs reduce activation memory at fixed parameter budget, lowering the GPU-memory threshold for training and serving large language models and broadening research access. Negative: the same efficiency gains can in principle accelerate training of frontier-scale models, with the dual-use considerations standard for foundational language-modeling work; this paper introduces no novel data, no new released checkpoints, and no capabilities beyond a more efficient FFN block, so it does not change the existing risk profile of LM research. We add no novel ethics-sensitive surface.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.

- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: No new pretrained model checkpoints or scraped datasets are released with this paper. All trained models are research-scale ($\leq$ 1.3B parameters) and are not made publicly available. The released artifact is the architecture and kernel implementation, which carries no per-se misuse risk beyond the standard FFN block it replaces.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All datasets and models used are publicly available and cited in the bibliography: THE PILE for pretraining, PG19 for evaluation loss, HellaSwag, SIQA, PIQA, OpenBookQA, WinoGrande, RACE for downstream evaluation, GPT-NeoX-20B tokenizer, and the prior architecture/kernel work (FlashAttention series, MH-MoE, MoEfication, Liger Kernel) are credited in the Related Work section. We use each asset within the terms of its public license; no asset has been re-licensed or redistributed by us.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The only new asset is the reference implementation of FlashFFN (model + fused kernels), released at `https://github.com/alexzms/FlashMHF` under the Apache License 2.0 with standard ML-codebase documentation (README, training/evaluation scripts, environment specification). No new datasets or model checkpoints are released.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: No crowdsourcing experiments or research with human subjects were performed. The paper studies a model-architecture and GPU-kernel co-design and uses only publicly licensed datasets.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve human-subjects research; IRB approval is not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: LLMs are not used as a component of the core methodology. The proposed architecture, kernels, theoretical analysis, and empirical evaluation were developed and conducted without LLM-driven research steps. Any LLM use in this work was confined to writing, editing, or formatting, which the NeurIPS LLM policy does not require to be declared.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.