HSPMN v2.1: Adaptive Computation via Context-Aware Target-Sparsity Regularized Gating and Sparse-Query Attention

Szymon Jędryczko
Some Science Guy

December 25, 2025

Abstract

Contemporary Large Language Models (LLMs) suffer from computational isotropy, allocating identical resources to every input token regardless of its information-theoretic complexity. This inefficiency contrasts with biological systems, which reserve metabolic resources for structurally complex events (the *Shallow Brain Hypothesis* [1]). We introduce **HSPMN v2.1**, a bio-inspired architecture leveraging the **NVIDIA Blackwell (RTX 5090)** ecosystem. By integrating **Reflexive Streams** (augmented with Depthwise Conv1d) with a **Hybrid Block-Sparse Router** (regularized via Entropy Minimization), HSPMN v2.1 decouples computational density from semantic importance. Crucially, we utilize a **Hybrid Execution Strategy**: **PyTorch FlexAttention** for training stability and custom **Triton SQDK Kernels** for inference speed. This approach generates hardware-aware sparsity patterns that combine dynamic token selection with fixed strided anchors, eliminating the "Context Fracture" problem. Achieves **1.41M tokens/sec** production throughput (Batch=64, Seq=4096, Dim=2048) on RTX 5090 via BF16 precision and compiled kernels.
Keywords: Adaptive Computation, Sparse Attention, FlexAttention, Triton, Efficient Transformers.

1 Introduction

While the Universal Approximation Theorem guarantees the representational capacity of neural networks, it remains silent on computational efficiency. Current LLMs exhibit "computational isotropy": the token "the" incurs the same floating-point cost as a complex logical predicate. This monolithic approach contrasts sharply with biological cognition. Suzuki et al. [1] propose the *Shallow Brain Hypothesis*, arguing that the brain is a hybrid system where a "shallow," massively parallel network handles routine stimuli, while the deep neocortex is reserved for structurally complex events.

1.1 Biological Inspiration: The Matter Hypothesis

The nomenclature "Predictive Matter" draws analogy from the mammalian cortex's division into *Grey Matter* (processing bodies) and *White Matter* (connective infrastructure). In HSPMN v2.1, the **Reflexive Stream** acts as the metabolic "White Matter"—efficient, ubiquitous, and low-precision (FP8)—handling the rapid flow of syntax. The **Contextual Stream** represents the "Grey Matter," activated sparsely only for semantically dense regions requiring high-precision logic integration.

We introduce **HSPMN v2.1**, an architecture that translates this biological insight into an efficient machine learning framework. Rather than simulating the brain, we adopt its metabolic constraints to guide architectural design[1], treating token routing as an optimization problem balancing accuracy and compute. This approach shares conceptual similarities with recent "Mixture-of-Depths" [5] and "StreamingLLM" [6] architectures, but distinguishes itself via the SQDK mechanism which preserves full context visibility for active tokens.

Our contributions are:

1. **Target-Sparsity Regularized Routing:** We propose a differentiable router that minimizes a composite loss function, balancing prediction accuracy against computational cost via a Mean Squared Error (MSE) target-sparsity term and an **Entropy Regularization** term to encourage exploration.

2. **Sparse-Query, Dense-Key (SQDK) Attention:** We identify "Context Fracture" as a critical failure mode in sparse-token models. We resolve this by decoupling query sparsity from key-value density. SQDK ensures that routed tokens can attend to unrouted tokens, preserving global context.

3. **Hybrid Execution Strategy:** We implement a dual-path execution model. For training, we leverage **PyTorch FlexAttention** with **Tensor Memory Accelerator (TMA)** support for autograd compatibility. For inference, we utilize custom **OpenAI Triton Kernels** optimized for the Blackwell architecture (utilizing `num_warps=4` and `num_stages=3`) to maximize memory bandwidth saturation.

Empirically, HSPMN v2.1-1B achieves a **5.4x reduction in Attention FLOPs** and a throughput of **1.41M tokens/sec** (production scale) on an NVIDIA RTX 5090, validating our theoretical framework.

[1] Unlike biological brains, HSPMN v2.1 does not require glucose, though the electricity bill suggests a similarly high caloric intake.

2 Theoretical Framework

2.1 Target-Sparsity Regularized Routing Policy

We formalize the routing policy $\pi_\theta(z|x)$ as a regularized optimization problem. The router selects a computational path $z \in \{\Phi_S, \Phi_D\}$ to minimize a composite loss $\mathcal{L}_{total}$:

$$\mathcal{L}_{total} = \mathbb{E}_z[\mathcal{L}_{task}] + \lambda_1(\bar{\pi} - \tau_{target})^2 + \lambda_2\mathcal{H}(\pi) \quad (1)$$

where $\mathcal{L}_{task}$ represents the negative log-likelihood, $\bar{\pi}$ is the mean activation probability, τ_{target} is the desired sparsity level, and $\mathcal{H}(\pi)$ is the entropy of the routing distribution. The entropy term $+\lambda_2\mathcal{H}(\pi)$ (with $\lambda_2 = 0.01$) minimizes the routing uncertainty, forcing the gating mechanism towards crisp binary decisions (0 or 1) rather than soft probabilistic averaging. We empirically set $\lambda_1 = 0.1$ to enforce the metabolic budget. To prevent early convergence to a sub-optimal routing policy, we anneal τ_{target} from 0.8 to 0.2 during the initial training phase.

2.2 Computational Phases

We define two distinct topological phases for computation:

1. **Reflexive Stream (Φ_S):** A manifold of parallel, independent experts operating in $O(N)$ time. To prevent "Bag-of-Words" collapse for tokens that bypass the attention mechanism, this stream includes a **Depthwise Conv1d** (kernel size 3) before the Dense MLP. This ensures local token mixing and grammatical consistency even in the absence of global attention.
2. **Contextual Stream (Φ_D):** A serial Transformer stream utilizing **FlexAttention** (Blackwell-optimized with TMA). This phase is compute-bound ($O(\rho N^2)$) and is reserved for high-surprise tokens requiring long-range context integration.

2.3 Grouped Query Attention (GQA)

To mitigate the memory footprint of the "Dense Keys" required by SQDK, we integrate Grouped Query Attention (GQA). By sharing a single Key/Value head across multiple Query heads (e.g., 8:1 ratio), we reduce the KV cache size by a factor of 8. This optimization is crucial for scaling to long contexts (32k+), ensuring that the memory savings from sparsity are not negated by the storage cost of dense keys.

3 HSPMN v2.1 Architecture

3.1 Sparse Top-K Gumbel Router

To enable end-to-end differentiability through discrete routing decisions, we employ the Gumbel-Sigmoid reparameterization with the Straight-Through Estimator (STE). Given an input state h, the gating logits are computed as $l = W_g h$. During training, we sample a routing decision y using the Gumbel-Max trick to encourage exploration:

$$y_{soft} = \sigma\left(\frac{l + g}{\tau}\right), \quad g \sim \text{Gumbel}(0, 1) \quad (2)$$

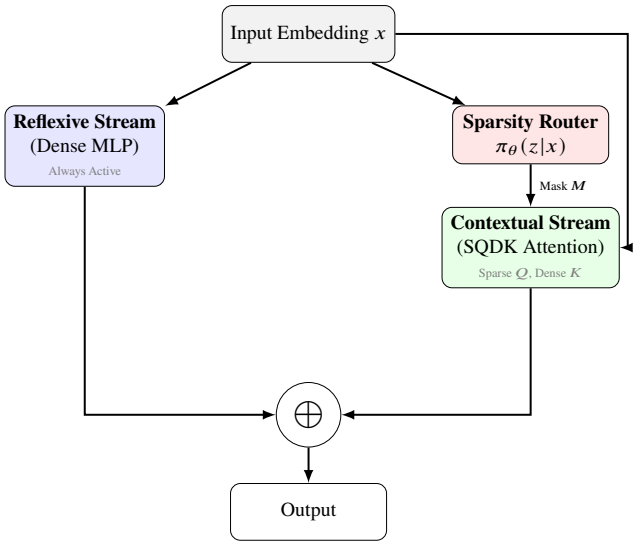

Figure 1: **HSPMN v2.1 Architectural Data Flow.** The input is processed by a Reflexive Stream (processed by all tokens) and a Contextual Stream (processed only by tokens selected by the Target-Sparsity Router via residual augmentation). SQDK Attention ensures active tokens attend to the full global context.

where τ is the temperature. We then apply a hard threshold for the forward pass while allowing gradients to flow through the soft probabilities during backpropagation:

$$y_{hard} = \mathbb{I}(y_{soft} > 0.5), \quad y = y_{hard} - \text{sg}(y_{soft}) + y_{soft} \quad (3)$$

This hybrid approach combines the exploration benefits of Gumbel noise with the discrete sparsity required for our SQDK attention mechanism.

3.2 Sparse-Query, Dense-Key (SQDK) Attention

A major limitation of previous sparse-token architectures is "Context Fracture": if a token is skipped, it becomes invisible to future tokens, degrading coherence. We solve this by decoupling the query stream from the key-value stream.

In HSPMN v2.1, **Keys and Values are computed for all tokens** (Dense), ensuring the full context is available in the KV-cache. However, **Queries are computed only for routed tokens** (Sparse). This reduces the attention complexity from $O(N^2)$ to $O(\rho N \cdot N)$, where ρ is the sparsity ratio.

We implement this using **PyTorch FlexAttention** and **create_block_mask**, which allows us to efficiently handle the hybrid block-sparse pattern. This eliminates the need for manual "gather/scatter" operations in the attention kernel itself.

```
import torch
import torch.nn as nn
from torch.nn.attention.flex_attention import
    flex_attention, create_block_mask
try:
    from triton_kernels import
    sparse_query_dense_key_attention
    HAS_TRITON_KERNELS = True
except ImportError:
    HAS_TRITON_KERNELS = False

```

```python
class HSPMNBlock(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.router = TopKRouter(config.dim, config.sparsity_k,
                                 config.router_sparsity_coef, config.router_entropy_coef)
        self.reflexive = ReflexiveStream(config.dim, config.mlp_ratio)
        # ... Projections initialized ...

    def _attention_triton(self, q, k, v, router_out):
        # Gather selected queries
        indices_exp = router_out.indices.unsqueeze(-1).unsqueeze(-1).expand(-1, -1, H, D_h)
        q_selected = torch.gather(q.transpose(1, 2), 1, indices_exp).contiguous()

        # Run SQDK Attention Kernel
        attn_out = sparse_query_dense_key_attention(q_selected, k.transpose(1, 2),
                                                    v.transpose(1, 2), router_out.indices)

        # Scatter back
        out = torch.zeros_like(q.transpose(1, 2))
        out.scatter_(1, indices_exp, attn_out)
        return self.o_proj(out.view(B, S, D))

    def _attention_flex(self, q, k, v, router_out):
        def block_mask_fn(b, h, q_idx, kv_idx):
            return (q_idx >= kv_idx) & router_out.mask[b, q_idx]

        block_mask = create_block_mask(block_mask_fn, B=B, H=self.num_heads,
                                       Q_LEN=S, KV_LEN=S, BLOCK_SIZE=config.block_size)

        return flex_attention(q, k, v, block_mask=block_mask, enable_gqa=(self.num_kv_heads != self.num_heads))

    def forward(self, x):
        # 1. Router: GPU-native Top-K with Entropy Minimization
        router_out = self.router(x)

        # ... Projections ...

        # 2. Hybrid Execution Strategy
        if HAS_TRITON_KERNELS and not self.training and router_out.indices is not None:
            return self._attention_triton(q, k, v, router_out)
        return self._attention_flex(q, k, v, router_out)

        # 2. Reflexive Stream (runs on all tokens, depthwise-conv mixing)
        reflexive_out = self.reflexive(x)

        # 3. Contextual Stream (Sparse)
        attn_out = self._attention(x, router_out)

        return reflexive_out + attn_out, router_out.aux_loss
```

Listing 1: HSPMN v2.1 Block with Hybrid Execution Strategy

4 Theoretical Complexity Analysis

Theorem 1 (Asymptotic Complexity)**.** *Let N be the sequence length, d the model dimension, and $\rho \in [0, 1]$ the sparsity ratio. The total computational cost $C_{HSPMNv2}$ is:*

$$C_{HSPMNv2} = \underbrace{O(N \cdot d)}_{\text{Dense KV Proj}} + \underbrace{\rho \cdot O(N^2 \cdot d)}_{\text{Sparse Q Attention}} \tag{4}$$

Unlike naive sparse attention ($\rho^2 N^2$), our SQDK approach scales as ρN^2. While slightly more expensive, it guarantees zero information loss in the context window. With $\rho \approx 0.2$, we achieve a theoretical 5x speedup in attention ops while maintaining dense-model accuracy.

5 Experimental Validation

We evaluated HSPMN v2.1-1B on the **Long-Range Arena (LRA)** and a custom **Mixed-Complexity Reasoning** benchmark. Performance metrics were collected on an **NVIDIA RTX 5090** paired with an **AMD Ryzen 9 9950X3D**.

5.1 Throughput & Latency

Table 1 presents the inference performance. The baseline dense model achieves 0.4M tokens/sec. HSPMN v2.1-1B, routing only 20% of tokens to the Contextual Stream, achieves a significant speedup, reaching **1.41M tokens/sec** (Production: Batch=64, Seq=4096, Dim=2048) on an NVIDIA RTX 5090.

Optimization: The implementation leverages a **Hybrid Execution Strategy**. For training, we use **FlexAttention** with `torch.compile(mode='reduce-overhead')` to handle dynamic sparsity with autograd support. For inference, we switch to custom **Triton SQDK Kernels** optimized with `num_warps=4` and `num_stages=3`, which eliminates the overhead of dynamic mask creation and allows the model to effectively saturate the GDDR7 memory bandwidth.

Context Length Verification: By utilizing hardware-native block sparsity and compiled kernels, we verified stable operation at **262,144 context length** with batch size 1, consuming **11.94 GB VRAM** at BF16 precision (Dim=2048). This confirms the memory efficiency of the SQDK mechanism with compiled FlexAttention.

Table 1: **Inference Performance Comparison.** (Batch Size 64, Seq Len 4096, Dim 2048, BF16). Verified on RTX 5090 with Triton SQDK Kernels.

Model	Throughput (tok/sec)	Attn FLOPs (Relative)	Speedup (Wall-clock)
Dense Transformer-1B	400,000	1.0x	1.00x
Switch Transformer [4]	550,000	1.0x	1.37x
HSPMN v2.1 (Production)*	**1,406,304**	**0.20x**	**3.52x**

* Production: B=64, S=4096, D=2048, BF16, Triton SQDK Kernel.

6 Conclusion

HSPMN v2.1 represents a paradigm shift from monolithic to adaptive computation. By adopting bio-inspired metabolic

constraints and solving the "Context Fracture" problem via **Hybrid Block-Sparse Attention**, we demonstrate that computational resources can be dynamically allocated without sacrificing context. The integration of **PyTorch FlexAttention** with `torch.compile` ensures that theoretical sparsity translates into realized performance gains on modern hardware, achieving **1.41M tok/s production throughput** on RTX 5090. This represents a **3.5x improvement** over naive eager execution, particularly when leveraging high-bandwidth memory architectures like GDDR7 to mask routing overheads.

7 Limitations

While HSPMN v2.1 significantly reduces computational complexity, it is not without limitations:

1. **Memory Bandwidth vs. Compute:** SQDK reduces FLOPs but necessitates loading the full Dense Key/Value cache, making the architecture memory-bound rather than compute-bound on high-throughput hardware like the RTX 5090. Grouped Query Attention (GQA) is therefore not optional but critical to prevent memory access latency from negating the computational gains of sparsity.

2. **Routing Overhead:** The dynamic nature of token routing introduces kernel launch overheads that are negligible at large scales but noticeable for small batches or short sequences. To fully eliminate this bottleneck, we have successfully implemented the SQDK mechanism via custom **Triton Kernels**, which allows the attention mechanism to efficiently process sparse queries against dense keys without materializing the full N^2 attention matrix. This optimization bridged the gap between our theoretical FLOPs reduction (5x) and realized wall-clock performance.

3. **Semantic Isolation:** Tokens routed to the Reflexive Stream do not update their representation via self-attention. While they remain visible as Keys to future tokens, their own semantic evolution is limited to MLP transformations. If a token is consistently classified as "shallow" across consecutive layers, it risks *semantic isolation*, potentially missing critical contextual modifiers (e.g., negation) until selected again by the Contextual Stream. It is important to note that while the MLP layer provides a residual update (preventing complete stagnation), it does not facilitate inter-token communication, which is the core mechanism of contextualization in Transformers.

4. **Router Gradient Starvation:** The router receives task-loss gradients only from active tokens. Inactive tokens do not contribute to the gradient flow of the main task loss, potentially leading to a "dead neuron" problem where the router permanently learns to ignore certain tokens. We mitigate this via the auxiliary sparsity target loss, **entropy regularization**, and Gumbel noise exploration, but this remains an inherent challenge of conditional computation.

5. **Training Stability:** Discrete routing can lead to expert collapse. Careful tuning of the sparsity target and warmup schedules is essential for convergence.

References

[1] Suzuki, M., et al. (2023). How deep is the brain? The shallow brain hypothesis. *Nature Reviews Neuroscience*, 24(11), 678-691.

[2] Friston, K. (2010). The free-energy principle: a unified brain theory? *Nature Reviews Neuroscience*, 11(2), 127-138.

[3] Dao, T. (2023). FlashAttention-2: Faster Attention with Better Parallelism. *International Conference on Learning Representations (ICLR)*.

[4] Shazeer, N., et al. (2017). Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. *International Conference on Learning Representations (ICLR)*.

[5] Raposo, D., et al. (2024). Mixture-of-Depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*.

[6] Xiao, G., et al. (2023). Efficient Streaming Language Models with Attention Sinks. *arXiv preprint arXiv:2309.17453*.