GSPN-2: Efficient Parallel Sequence Modeling

Hongjun Wang[1,2,†], Yitong Jiang[1], Collin McCarthy[1], David Wehr[1], Hanrong Ye[1], Xinhao Li[3],

Ka Chun Cheung[1], Wonmin Byeon[1], Jinwei Gu[1], Ke Chen[1], Kai Han[2,*]

Hongxu Yin[1], Pavlo Molchanov[1], Jan Kautz[1], Sifei Liu[1]
[1]NVIDIA [2]The University of Hong Kong [3]University of California, San Diego

Abstract

Efficient vision transformer remains a bottleneck for high-resolution images and long-video related real-world applications. Generalized Spatial Propagation Network (GSPN) [1] addresses this by replacing quadratic self-attention with a line-scan propagation scheme, bringing the cost close to linear in the number of rows or columns, while retaining accuracy. Despite this advancement, the existing GSPN implementation still suffers from (i) heavy overhead due to repeatedly launching GPU kernels, (ii) excessive data transfers from global GPU memory, and (iii) redundant computations caused by maintaining separate propagation weights for each channel. We introduce GSPN-2, a joint algorithm–system redesign. In particular, we eliminate thousands of micro-launches from the previous implementation into one single 2D kernel, explicitly pin one warp to each channel slice, and stage the previous column's activations in shared memory. On the model side, we introduce a compact channel propagation strategy that replaces per-channel matrices, trimming parameters, and align naturally with the affinity map used in transformer attention. Experiments demonstrate GSPN-2's effectiveness across image classification and text-to-image synthesis tasks, matching transformer-level accuracy with significantly lower computational cost. GSPN-2 establishes a new efficiency frontier for modeling global spatial context in vision applications through its unique combination of structured matrix transformations and GPU-optimized implementation. Project page: `https://whj363636.github.io/GSPN2/`

1 Introduction

Vision transformers have underpinned nearly every state-of-the-art (SOTA) vision foundation model: text-to-image diffusion networks (e.g., Stable Diffusion [2]), vision-language aligners such as CLIP [3] and SigLIP [4], and modern detection/segmentation pipelines [5, 6] – all depend on their dense, token-wise attention to encode visual concepts. Since this attention operator scales quadratically with the number of pixels, practical deployments still cap the input—SigLIP [4], for instance, limits the input images to 512×512—to avoid prohibitive latency and memory. Recently, several efficient-attention variants have been proposed, such as FlashAttention [7, 8], linear attention [9, 10, 11], and state-space models [12, 13]. Among them, Generalized Spatial Propagation Networks (GSPN) [1] uniquely replace 2D self-attention with a line-scan approach, which reduces the computational complexity from quadratic to approximately linear to the image's width or height. Remarkably, GSPN maintains or even surpasses baseline accuracy while achieving up to an 84× speed-up for 16 K-resolution diffusion inference.

While most efficient attention backends can reuse existing matrix-multiply or scan primitives, GSPN's line scan demands a completely new CUDA implementation. Standard Softmax attention breaks down into a series of GEMM-based matrix multiplies plus a softmax [14]. FlashAttention [7, 8]

[*] Corresponding author.

† Hongjun Wang was an intern at NVIDIA during the project.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

instead fuses those steps into a single tiled GEMM loop. State-space methods like Mamba [12, 13] recasts attention into a streaming recurrence and implement it with fast prefix-sum scans across tokens.

By contrast, GSPN adopts a 3-neighbour line-scan approach, which is neither a matrix multiply, nor a prefix scan—its dependency pattern would explode combinatorially. Therefore, a specifically built CUDA kernel for GSPN is required to unlock its sub-quadratic cost.

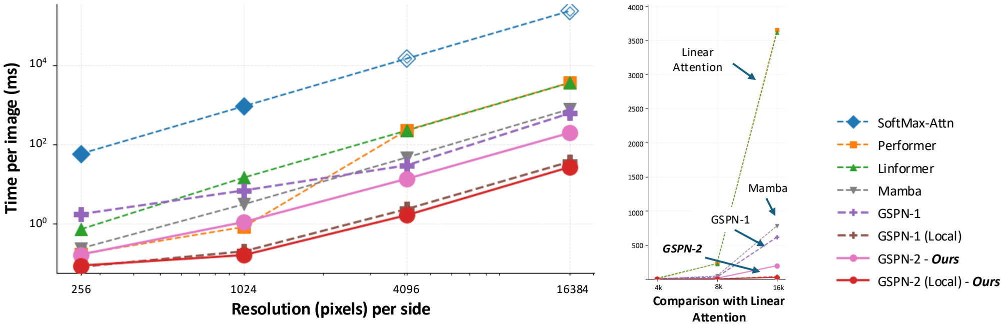

Figure 1: GSPN-2 achieves transformative performance improvements over GSPN-1[1] and other efficient attention variants, running up to 30-50× faster across diverse input configurations on modern GPU architectures.

While GSPN-1 offers theoretical advantages in computational complexity, its initial CUDA implementation [1] struggled to translate these into practical speedups. Profiling reveals that GSPN-1's reference CUDA code, which launches a tiny kernel for each column step, severely underutilizes the GPU, as it achieves just 3–8% of peak memory bandwidth and low SM occupancy. This inefficiency stemmed from several critical bottlenecks: (1) the kernel-launch overhead from thousands of separate launches, which prevents the SMs from staying fully busy; (2) inefficient global-memory (HBM) access, with each step reloading data without on-chip reuse or coalescing; and (3) poor cache locality and growing runtime as channel counts increase.

To overcome the limitations of the original GSPN implementation, we introduce GSPN-2, an integrated algorithmic and kernel-level redesign. Specifically, GSPN-2 (a) consolidates all propagation steps into a single unified CUDA kernel, eliminating costly repeated launch overheads; (b) introduces a compact multi-channel propagation mechanism that projects features into a lower-dimensional proxy space to reduce concurrent thread blocks and maintain constant-time performance; and (c) refines the grid and block configuration to improve warp-level efficiency and memory coalescing, with optional on-chip caching for reuse of hidden states. To further address the GPU concurrency bottleneck—where runtime increases sharply once the number of active thread blocks exceeds the device's scheduling capacity—GSPN-2 employs a lightweight channel-projection strategy. By projecting the input tensor into a compact proxy space before propagation, the system reduces the effective computational dimension, improving cache reuse and maintaining high throughput even under large batch and channel counts (see Section 4.2). These combined algorithmic and kernel-level improvements deliver substantial performance gains: on an NVIDIA A100 GPU, the runtime for a $1024 \times 1024 \times 8$ input decreases from 71.4 ms in GSPN-1 to just 1.8 ms in GSPN-2—achieving an overall 40× speedup (see Figure 3).

Our experimental evaluation comprehensively validates GSPN-2. Rigorous efficiency analysis demonstrates that GSPN-2 runs up to $30\times$ faster than GSPN-1 across diverse input configurations, with performance profiling confirming near-optimal hardware utilization (over 90% of theoretical peak memory bandwidth). We then validate GSPN-2's effectiveness across vision tasks: image classification and text-to-image synthesis. On ImageNet, GSPN-2 achieves accuracy comparable to transformer models at significantly lower computational cost. In text-to-image synthesis, GSPN-2 significantly improves semantic consistency and visual quality when integrated with existing diffusion models. These results confirm GSPN-2 as a versatile component for efficiently modeling global spatial context across diverse vision applications.

2 Related Works

Efficient Attention Mechanisms. Transformer architectures [15] have become foundational components in modern vision and language models, but their quadratic computational complexity with respect to sequence length creates significant efficiency challenges. FlashAttention [7, 8, 14] addresses these limitations through algorithmic innovations that optimize memory access patterns and reduce unnecessary memory reads/writes during attention computation. By leveraging tiling strategies and fusing operations to maximize data reuse within fast GPU memory hierarchies, FlashAttention substantially improves throughput and enables processing of longer sequences without compromising model quality. These efficiency gains have been instrumental in scaling transformer models to increasingly larger contexts and higher resolution inputs, but the fundamental quadratic complexity of attention remains an inherent limitation.

Sequence Modeling in 1D and 2D Space. Sequential modeling has been dominated by recurrent architectures like LSTMs [16], GRUs [17], and 2D-LSTMs [18, 19], which process data through non-linear transformations. Despite their effectiveness, these approaches face fundamental limitations in computational efficiency and scalability due to their inherent sequential nature. Their non-linear propagation mechanisms also struggle with long-term dependencies, often suffering from gradient vanishing or exploding issues [20, 21] that prevent distant information from effectively influencing future states. State Space Models (SSMs) have emerged as promising alternatives to attention-based architectures, offering linear computational complexity with respect to sequence length. Pioneering approaches like S4 [22] and Mamba [12] implement continuous-time dynamical systems through discretized state spaces, enabling efficient modeling of long-range dependencies without the quadratic cost of attention. These models maintain a compact hidden state that evolves through linear recurrence relations, often employing selective scanning mechanisms to adapt to input-dependent patterns. For visual tasks, several approaches [23, 24, 25, 26, 27] have adapted SSMs by linearizing 2D image data, though this transformation potentially compromises inherent spatial relationships present in the original data structure.

Spatial Propagation Networks. The Spatial Propagation Network (SPN) [28] pioneered linear propagation specifically for 2D data, initially designed as a single-layer component on top of CNNs for sparse-to-dense prediction tasks like segmentation. However, SPN's potential as a scalable foundational architecture comparable to Vision Transformers (ViT) remains largely unexplored. Moreover, SPN's sequential processing across different spatial directions inherently limits its computational efficiency, and it fails to adequately address long-range propagation requirements crucial for high-level vision tasks. Our GSPN architecture advances beyond these limitations by implementing parallel row/column-wise propagation mechanisms that enable efficient learning of affinity matrices while maintaining gradient stability and effective long-range correlation. Through both theoretical analysis and empirical evaluation, we demonstrate that GSPN represents a compelling alternative to established ViT and Mamba architectures.

3 Background

We introduce the background of the propagation algorithm itself and the GPU design principles. In Section 3.1, we review modern A100 architecture—the grid/block/warp execution model, on-chip shared memory, and high-bandwidth device memory. We explain how these features shape kernel performance. In Section 3.2, we review GSPN's line-scan propagation formulation. The final section summarizes how this recurrence is mapped onto CUDA blocks in a custom kernel implemented in [1] to realize parallelism.

3.1 GPU Hardware Characteristics

On modern NVIDIA GPUs like the A100, computation is dispatched as a grid of thread blocks. Each block can be 1D, 2D, or 3D (e.g., `blockDim.x` alone for 1D block, or both `blockDim.x` and `blockDim.y` for 2D block). Inside each block, threads are organized into warps of 32 threads each. The total number of warps per block depends on the block's thread count (for example, a block with 1024 threads has 32 warps). To maximize throughput, blocks must be sized to supply enough active warps per Streaming Multiprocessors (SM) without exceeding their on-chip shared-memory or register limits–this balance is what drives high occupancy. Within each block, threads share a small SRAM buffer ("shared memory") for low-latency reuse, while all other tensors are streamed in and out of off-chip high-bandwidth memory (HBM) through the L2/L1 caches.

3.2 2D Spatial Propagation Algorithm Overview

Generalized Spatial Propagation Networks (GSPN) [1] perform 2D spatial modeling through row-by-row (or column-by-column) linear propagation. For an input image $x \in \mathbb{R}^{H\times W\times C}$, this involves processing one dimension (e.g., rows) sequentially, while computations within each step (e.g., along a row) are parallelized. Focusing on row-wise propagation, where $i \in [0, H-1]$ is the row index,

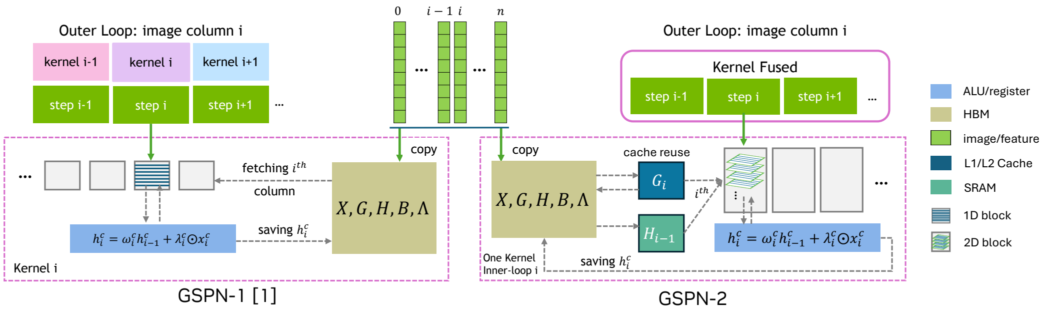

Figure 2: **Pipeline optimization from GSPN-1 to GSPN-2.** GSPN-1 launches separate kernels for each image column i, leading to redundant HBM access and limited temporal data reuse. Each kernel independently computes $h_i^c = \omega_i^c h_{i-1}^c + \lambda_i^c \odot x_i^c$, fetching and storing intermediate states via global memory. GSPN-2 fuses these operations into a single kernel with an inner loop over columns, enabling cache and register reuse of h_{i-1}^c, G_i, and other temporaries. This design minimizes memory traffic, maximizes locality, and leverages shared memory for efficient on-chip computation.

let $h_{i,:,c} \in \mathbb{R}^W$ be the hidden state for row i and channel c, $x_{i,:,c} \in \mathbb{R}^W$ be the corresponding input row, and $\lambda_{i,:,c} \in \mathbb{R}^W$ be a learnable, input-dependent scaling vector for channel c at row i, and $w_{i,c} \in \mathbb{R}^{W \times W}$ be a learnable, per-channel propagation weight matrix for row i and channel c. The per-channel recurrence is:

$$h_{i,:,c} = w_{i,c} h_{i-1,:,c} + \text{Diag}(\lambda_{i,:,c}) x_{i,:,c} \tag{1}$$

The per-channel hidden state $h_{i,:,c}$ is then transformed by an output layer. Let $u_{i,:,c} \in \mathbb{R}^W$ be a learnable output vector. The final output for row i and channel c is given by:

$$y_{i,:,c} = u_{i,:,c} \odot h_{i,:,c} \tag{2}$$

Guided by the Stability-Context Condition introduced in [1], $w_{i,c}$ is learned and normalized to be row-stochastic—every row sums to 1—thereby guaranteeing numerical stability while still capturing long-range context. In addition, each element in row i connects to only three neighboring elements in the previous row $i-1$ (e.g., top-left, top-center, top-right for top-to-bottom propagation), resulting in $w_{i,c}$ being a tridiagonal matrix. A single pass of this recurrence connects pixels within a local region. To achieve full-grid propagation, GSPN performs four complementary directional passes—top-to-bottom, bottom-to-top, left-to-right, and right-to-left. By combining the 3-neighbor kernel with these four passes, the model attains dense pairwise connectivity across the image while remaining efficient, since only three coefficients are learned per pixel.

This row-wise propagation requires $O(H)$ sequential steps, where within each step, all W elements of the current row are computed in parallel (or vice versa for column-wise propagation), yielding an $O(\max(H, W))$ computational complexity—equivalent to $O(\sqrt{N})$ for square images with N pixels. GSPN also offers a local variant, i.e., GSPN-local, which splits each row or column into fixed-length segments of size `kchunk` and confines propagation to within those segments. Finally, this recurrence is related to linear attention. Let $\Lambda_i = \text{blockdiag}(\text{Diag}(\lambda_{i,:,0}), \ldots, \text{Diag}(\lambda_{i,:,C-1}))$. The overall operation is $y_i = u_i \sum_{j=0}^{i-1} (\prod_{\tau=j+1}^{i} w_\tau) \Lambda_j x_j$ which resembles a linear attention mechanism, with $\prod_{\tau=j+1}^{i} w_\tau$ the normalization term.

3.3 CUDA Implementation in GSPN [1]

The baseline GSPN implementation, referred to as GSPN-1, maps the 2D spatial propagation (Eq. 1) to CUDA by iterating sequentially over the propagation dimension (e.g., height H) while parallelizing computations across the orthogonal dimension (e.g., width W). To handle the inherent sequential dependency along the propagation axis (e.g., rows $i = 0, \ldots, H-1$), it launches separate, relatively lightweight CUDA kernels for individual steps or small chunks, as illustrated in Figure 2(a).

For each step in the propagation sequence (e.g., processing a column based on the previous one in a left-to-right scan), GSPN-1 launches a new CUDA kernel, which introduces significant kernel-launch overhead along the propagation direction. Within each kernel, computations are parallelized across the orthogonal spatial dimension (width W), batches (N), and channels (C) by flattening these dimensions into a 1D grid of thread blocks (typically `blockDim.x = 512`). However, this simplistic mapping ignores CUDA's warp-level scheduling and often results in suboptimal hardware utilization. Moreover, all tensors—including inputs (x), hidden states from the previous step ($h_{i-1,:,c}$), learnable weights ($w_{i,c}$, $\lambda_{i,:,c}$), and the current outputs ($h_{i,:,c}$)—are repeatedly read from and written back to global GPU memory (HBM), causing high latency and minimal data reuse in on-chip memory. These issues—frequent kernel launches, suboptimal thread mapping, and excessive global-memory access—together limit the efficiency of GSPN-1, motivating the redesign presented in Section 4.

4 GSPN-2: Efficient Algorithm and System Co-design

While the baseline GSPN-1 implementation established functional correctness, its CUDA design suffered from inefficiencies—frequent kernel launches, flat 1D block configurations, and unpredictable memory reuse, which led to suboptimal data locality and high launch overhead. To address these limitations, we redesign both the algorithm and its GPU execution pipeline through GSPN-2, focusing on three principles: (1) a single-kernel propagation scheme that eliminates redundant launches, (2) channel-compressive propagation with shared weights and proxy compression to reduce concurrency load, and (3) optimized CUDA execution leveraging shared memory, coalesced access, and stream-level parallelism. This section introduces the evolution of GSPN-2, from its single-kernel design to memory- and concurrency-aware optimization.

4.1 A Single-Kernel Design

Kernel Fuse. We consolidate these numerous small kernels into a single, unified CUDA kernel. This single kernel is designed to process the entire outer-loop (e.g., all columns in a left-to-right scan) *within* the kernel, while still parallelizing computations across the other dimensions (batch, channels, and rows/height). By eliminating thousands of micro-launches, this single-kernel approach drastically reduces launch overhead. For instance, preliminary tests showed that simply moving from a multi-kernel to a single-kernel implementation for a typical GSPN configuration immediately yielded a significant performance boost (e.g. 1.2× faster) in Figure 3, even before other memory or algorithmic optimizations were applied. We illustrate substantial performance gains from this and subsequent optimization stages across various hardware configurations and input dimensions (batch size, channels, height, width) in Figure 3 and Section 5.1.

Block and Grid Configuration. In GSPN-1, the kernel used a flat 1D grid of blocks (`blockDim.x = 512`) where threads were linearly mapped across combinations of batch (N), channel (C), height (H), and chunk index (k_{chunk}). This configuration resulted in insufficient locality and suboptimal warp utilization. In GSPN-2, the CUDA grid is indexed by the tuple (chunk, n, c), so that each block corresponds to one unique (chunk, n, c) combination and processes a full spatial column along height. The grid therefore contains $k_{\text{chunk}} \times N \times C$ blocks in total, which can be realized as a 1D grid or a 3D grid to respect CUDA's per-axis limits. Each block uses up to 1024 threads along the height dimension. For $H \leq 1024$, one thread is assigned per row, achieving full occupancy. When $H > 1024$, threads iterate over multiple rows with stride `blockDim.x`, ensuring complete coverage without exceeding the per-block thread limit.

4.2 Compact Channel Propagation

A key performance bottleneck in GSPN-1 arises from GPU concurrency saturation when the number of active CUDA blocks—proportional to $k_{\text{chunk}} \times N \times C$—exceeds the hardware's concurrent execution capacity. On GPUs such as NVIDIA A100, each Streaming Multiprocessor (SM) can host up to 32 resident thread blocks (compute capability 8.0), and with 108 SMs available, roughly $108 \times 32 \approx 3{,}500$ blocks can be active concurrently under ideal conditions. Under typical GSPN workloads, kernel execution time remains nearly constant up to this scale (about 3–4K concurrent blocks). Beyond that point, the runtime grows linearly as additional blocks wait in the scheduling queue. This saturation effect causes GSPN-1 to lose its near-constant runtime scaling when operating on high-dimensional feature maps (e.g., thousands of channels).

To address this, GSPN-2 introduces a **compact multi-channel propagation** scheme that reduces the effective channel concurrency while maintaining expressive multi-channel behavior. The core idea is to project the input tensor $x \in \mathbb{R}^{N \times C \times H \times W}$ into a lower-dimensional proxy subspace $x_{\text{proxy}} \in \mathbb{R}^{N \times C_{\text{proxy}} \times H \times W}$, where $C_{\text{proxy}} \ll C$ (e.g., $C_{\text{proxy}} = 8$). The propagation is then applied to this proxy representation using shared propagation matrices w_i and later restored to the original C-channel space. This reduces the total block count from $k_{\text{chunk}} \times N \times C$ to $k_{\text{chunk}} \times N \times C_{\text{proxy}}$, reducing it as much as possible to stay well within the hardware concurrency regime (roughly 3–4K on A100-class GPUs) and thereby sustaining near-constant performance.

Illustrative Single-Channel Case. We use the single-channel case to make the attention analogy explicit. We replace per-channel weights with a single propagation matrix per column. In this view, w_i will be shared among all the channels, which plays the role of an attention-style affinity matrix over positions in column i, and the per-position input scaling acts like value gating. The per-channel recurrence thus becomes:

$$h_{i,:,c} = w_i h_{i-1,:,c} + \lambda_{i,:,c} \odot x_{i,:,c} = w_i h_{i-1,:,c} + \text{Diag}(\lambda_{i,:,c}) x_{i,:,c} \tag{3}$$

where w_i governs spatial propagation along the column, and $\lambda_{i,:,c}$ preserves per-channel modulation. This formulation significantly reduces the number of parameters while retaining the same functional structure. Stacking all channels, the full recurrence $h_i = W_i h_{i-1} + \Lambda_i x_i$ still holds, now with channel-shared w_i. To expand Eq. (4.2), we denote H_v, X_v as the concatenation of all $h_{i,:,c}$ and $x_{i,:,c}$ into vectors. The expansion yields a block lower-triangular matrix form:

$$H_v = \begin{bmatrix} \Lambda_1 & 0 & \cdots & \cdots & 0 \\ w_2\Lambda_1 & \Lambda_2 & 0 & \cdots & 0 \\ w_3 w_2 \Lambda_1 & w_3\Lambda_2 & \Lambda_3 & 0 & \cdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (\prod_{k=2}^{L} w_k)\Lambda_1 & (\prod_{k=3}^{L} w_k)\Lambda_2 & \cdots & w_L \Lambda_{L-1} & \Lambda_L \end{bmatrix} X_v = G X_v, \tag{4}$$

where each block G_{ij} is a $N \times N$ matrix representing how the input slice x_j contributes to the output h_i—directly analogous to an attention mechanism's affinity matrix. Here, the channel-shared matrices w_i define dense spatial relationships, while the channel-specific scaling matrices Λ_j inject feature-wise modulation. This formulation shows that, in the single-channel case, GSPN-2 can be viewed as an attention-like process with learnable spatial affinities.

Compressive Proxy Dimension. To further relieve concurrency saturation when $N \times C$ is large, we compress the channel axis before propagation. Concretely, we project $x \in \mathbb{R}^{N \times C \times H \times W}$ to $x_{\text{proxy}} \in \mathbb{R}^{N \times C_{\text{proxy}} \times H \times W}$ with $C_{\text{proxy}} \ll C$ (e.g., C_{proxy}=8), apply the same columnwise recurrence in the proxy space using the shared w_i, and expand back to C with a learned 1×1 projection. This reduces the grid from $k_{\text{chunk}} \times N \times C$ to $k_{\text{chunk}} \times N \times C_{\text{proxy}}$, shrinking the number of simultaneously scheduled block slices (e.g., $N \times C_{\text{proxy}} \times H$ for a row scan). We choose C_{proxy} to minimize the active-block budget and delay entry into the post-saturation, near-linear regime on A100-class GPUs; even when that plateau cannot be fully avoided (very large N), the compression still cuts queueing and improves SM utilization while preserving multi-channel expressiveness.

4.3 Efficient CUDA Scaling under Large Block-Slice Loads

This section presents CUDA kernel enhancements—particularly grid/block reconfiguration and on-chip memory strategies—that enable efficient compact channel propagation even when the block count (k_chunk $\times$ N $\times$ C) becomes very large.

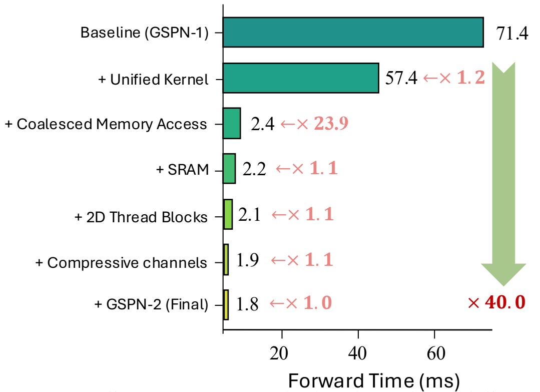

Figure 3: **Step-by-step optimization of the GSPN CUDA kernel.** Each bar shows the reduction in forward time (ms) achieved through cumulative optimizations, starting from the GSPN-1 baseline. The final implementation (GSPN-2) achieves a 40.0× speedup compared to the baseline.

SRAM for Hidden States. We cache the previous step's hidden state (e.g., h_{i-1}) in on-chip shared memory to reduce redundant global-memory (HBM) reads. Within each CUDA block, threads cooperatively process a *tile*—a small subset of spatial positions or channel slices—and reuse the cached hidden-state values stored in shared memory. This on-chip reuse reduces latency when multiple threads within a block access overlapping regions of h_{i-1}, such as along a spatial column. The performance gain depends on configuration: it is most effective when the reuse per tile is high, the shared-memory footprint fits comfortably within per-block limits, and bank conflicts are minimal. When reuse is low or L1/L2 caching already covers the working set, the benefit diminishes. Therefore, we enable this shared-memory caching selectively and tune tile size and `cSlice` to balance reuse against occupancy.

2D Block Design for Channel-Parallel Propagation. Building upon the 1D block design in Sec. 4.1, we extend it to a 2D configuration by introducing a second dimension, `cSlice`, such that each CUDA block has $blockDim = (H, cSlice)$. Within a block, `threadIdx.x` corresponds to spatial positions along a column (up to H), while `threadIdx.y` spans a small group of channel slices. This enables the block to process multiple channels of the same column in parallel, improving hardware utilization and memory throughput even when each channel maintains its own propagation weight w_i^c. Compared to the earlier 1D block layout, this 2D configuration achieves better occupancy and reduced latency by aligning computation and memory access patterns across both spatial and channel dimensions, as demonstrated in Section 5.1.

Coalesced Memory Access. A major source of speedup in GSPN-2 comes from enforcing coalesced global-memory access. We arrange x_i, h_i, and w_i contiguously in memory so that consecutive threads in a warp access adjacent addresses when reading or writing. This pattern allows the hardware to combine multiple per-thread transactions into a single wide memory operation, fully utilizing the available bandwidth and minimizing wasted cycles. By eliminating the irregular, strided accesses present in GSPN-1, the coalesced layout contributes the largest single performance gain among all CUDA-level optimizations (see Figure 3).

Stream-Based Concurrency. For multi-directional propagation, GSPN-2 executes each directional pass on a separate, non-blocking CUDA stream. This allows concurrent kernel execution across directions, improving hardware utilization by keeping more SMs active—especially on GPUs with abundant SM resources. The benefit depends on workload balance and available parallelism; it is most pronounced when the directional passes have similar compute and memory footprints. In addition, when the grid dimension exceeds CUDA's per-axis limit of 65,535, GSPN-2 automatically performs multiple launches with offset indexing to cover the excess range without interrupting the overall stream concurrency.

5 Experiments

To demonstrate the effectiveness of GSPN-2, we design experiments that answer two questions: *How much faster is it?* and *Does the speed-up preserve—or even improve—task performance?* We first profile the new CUDA kernel, isolating the gains from different factors, such as unified kernel launch, shared memory, and channel-share weights. We then benchmark GSPN-2 on a suite of vision tasks, comparing accuracy and throughput against GSPN-1 and other strong baselines. By evaluating both efficiency and task performance, we demonstrate the benefits of our tightly integrated algorithm and kernel optimizations.

5.1 Detailed Profiling and Performance Characteristics

To understand GSPN-2's performance characteristics in depth, we conducted comprehensive profiling across various input configurations, analyzing memory throughput, cache utilization, and computational efficiency.

Step-by-step CUDA Optimization. We benchmark a typical configuration, i.e., 1024×1024 image size, batch size 16, 8 channels, and quantify the impact of each CUDA kernel optimization term in Figure 3. The GSPN-1 baseline exhibited suboptimal performance (71.4 ms) due to kernel launch overhead and inefficient memory access patterns. Our first optimization—a single fused kernel (Sec. 4.1)—eliminates thousands of micro-launches by processing entire scan operations within a single kernel, yielding a notable 1.2× speedup (57.4 ms). **Coalesced Memory Access** patterns (Sec. 4.3) maximized memory bandwidth utilization for a substantial 23.9× improvement (2.4 ms). Implementing **Shared Memory Cache** for hidden states (Sec. 4.3) reduced global memory traffic by 1.1× (2.2 ms). Restructuring to **2D Thread Blocks** (Sec. 4.1, Sec. 4.3) improved thread organization and data locality for another 1.1× gain (2.1 ms). **Compressive channels** (Sec . 4.2) reduced parameter fetch overhead and enhanced cache coherence for a 1.1× speedup (1.9 ms). The fully optimized GSPN-2 implementation achieves an impressive 40.0× cumulative speedup (1.8 ms) over the original baseline. We note that the relative impact of each optimization varies with workload characteristics (batch size, channel count); Section B.1 in the appendix provides a detailed analysis under an alternative large-batch configuration (batch size 256, 1 channel), demonstrating that while coalesced memory access remains the dominant optimization, **Shared memory caching** and **2D thread blocks** (Sec. 4.3) exhibit configuration-dependent benefits.

Memory Throughput Analysis. As shown in Table 1, NVIDIA Nsight Compute profiling indicates that GSPN-2 achieves memory throughput near the theoretical limit, with global-memory efficiency reaching 93% on A100 GPUs. This efficiency remains remarkably stable across a wide range of batch sizes and spatial resolutions, demonstrating effective saturation of the available bandwidth. In contrast, GSPN-1 exhibits highly variable throughput—only 3–8% of peak—that further deteriorates as input dimensions increase.

Table 1: **Global memory throughput under typical input configurations on A100 GPU.** We show throughput for a range of input sizes, batch sizes, and channel counts representative of common deployment scenarios in different tasks. Rather than exhaustively sweeping all variables, we select practical configurations to demonstrate consistent and significant gains of GSPN-2 over GSPN-1 across diverse settings.

Input Size	Batch	Channels	GSPN-1 Throughput	GSPN-2 Throughput
32×32	32	196	114 GB/s (6.0%)	1832 GB/s (91.8%)
64×64	1	768	86 GB/s (4.5%)	1847 GB/s (92.3%)
64×64	1	1152	35 GB/s (2.1%)	1837 GB/s (92.0%)
64×64	1	32	125 GB/s (6.3%)	1830 GB/s (91.5%)
128×128	1	32	98 GB/s (4.9%)	1865 GB/s (93.3%)
256×256	1	64	76 GB/s (3.8%)	1842 GB/s (92.1%)
256×256	8	64	94 GB/s (4.7%)	1858 GB/s (92.9%)
512×512	1	128	64 GB/s (3.2%)	1840 GB/s (92.0%)

Performance Scaling with Input Size. As shown in the upper row of Figure 4, GSPN-2 consistently outperforms GSPN-1 across various image resolutions with fixed batch and channel counts. For large image sizes (1024×1024), we observe speedups of up to 36.8× for forward passes and 25.3× for backward passes. This substantial improvement is particularly relevant for high-resolution visual processing tasks such as image generation and super-resolution, where spatial dimensions significantly impact computational demands. The performance gap widens as image resolution increases, highlighting GSPN-2's superior ability to handle spatially dense computations efficiently through its optimized memory access patterns and unified kernel design.

Performance with Varying Batch Size and Channel Dimensions. The lower row of Figure 4 demonstrates GSPN-2's exceptional performance in scenarios requiring large batch sizes or high channel dimensions—critical requirements for video generation, foundation model visual towers, and multimodal applications. With three distinct performance lines (GSPN-1 and GSPN-2), we observe that GSPN-2 maintains consistent 2-4× speedups over GSPN-1 even as batch sizes scale to 256 or channel counts increase to 1024. For instance, when processing inputs with 256 channels, GSPN-2 achieves a 27.4× speedup on forward passes and 48.6× on backward passes. These improvements are particularly valuable for production inference systems handling multiple streams simultaneously or for models requiring high feature dimensionality. The channel-sharing approach (Sec. 4.2) provides additional efficiency gains of up to 1.5× in these demanding scenarios, enabling practical deployment of GSPN architectures in compute-intensive applications like real-time video processing and multimodal foundation models.

L1 Cache Effectiveness. One surprising finding from our profiling is the effectiveness of the L1 cache even without explicit shared memory caching in certain configurations. When we experimented with a shared memory buffer to store previous hidden states (h_{t-1}), we observed that performance remained largely unchanged compared to relying on L1 cache. Detailed profiling revealed L1 cache hit rates of approximately 35% for the standard implementation. Interestingly, when using shared memory explicitly, L1 hit rates dropped to near 0%, with those accesses now served from shared memory instead. Despite this shift in memory hierarchy usage, latency remained comparable between both approaches, suggesting modern GPU L1 caches are highly effective for structured access patterns. The transposed data layout and coalesced access patterns enable effective hardware caching, even without explicit shared memory management in some cases. However, for maximum portability across GPU architectures and to ensure deterministic performance, the shared memory implementation remains preferable.

Streaming Multiprocessor Utilization Our profiling reveals an important relationship between input configuration and SM utilization. With GSPN-2's 2D thread organization strategy, SM occupancy varies significantly based on workload characteristics. For large batch sizes and channel counts, SM occupancy approaches 100%, fully utilizing the GPU's 108 SMs on A100. However, for small

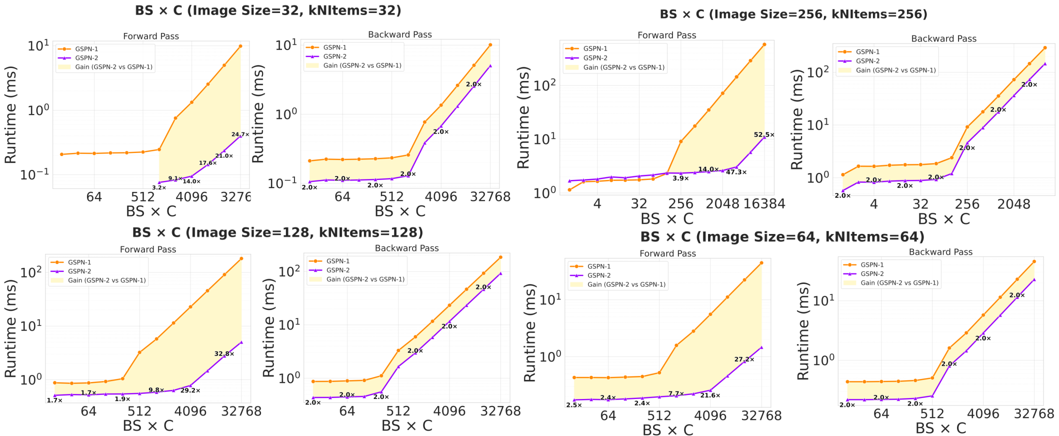

Figure 4: **Runtime Performance Comparison of GSPN-1 and GSPN-2.** We show forward and backward pass execution times (in milliseconds) across different channel counts. Results are presented for various configurations. GSPN-2 greatly improves the runtime of both forward and backward passes across different cases.

batch sizes and channel counts, occupancy can drop significantly (as low as 20-30%). This occurs because when processing independent chunks, each chunk requires one block, limiting parallelism for small input dimensions. This suggests potential areas for further optimization—particularly for low batch size, low channel count scenarios where we could further decompose the problem to increase parallelism across SMs.

Table 2: **Performance of models on ImageNet at the resolution of** 224^2**.** Colors denote different backbone types: yellow for CNNs, orange for Transformers, and green for Raster scan (i.e., 1D linear propagation) methods.

Model	Backbone	Param (M)	IN-1K	
			MAC (G)	Acc (%)
ConvNeXt-T [29]	CN	29	4.5	82.1
MambaOut-Tiny [30]	CN	27	4.5	82.7
DeiT-S [31]	TF	22	4.6	79.8
T2T-ViT-14 [32]	TF	22	4.8	81.5
Swin-T [33]	TF	29	4.5	81.3
SwinV2-T [34]	TF	28	4.4	81.8
CSWin-T [35]	TF	23	4.3	82.7
CoAtNet-0 [36]	TF	25	4.2	81.6
Vim-S [25]	RS	26	5.1	80.5
VMamba-T [26]	RS	22	5.6	82.2
Mamba-2D-S [27]	RS	24	–	81.7
LocalVMamba-T [37]	RS	26	5.7	82.7
VRWKV-S [38]	RS	24	4.6	80.1
ViL-S [39]	RS	23	5.1	81.5
MambaVision-T [40]	RS	32	4.4	82.3
GSPN-T	Line	30	5.3	**83.0**
GSPN-2-T (Ours)	Line	24	4.2	**83.0**

Model	Backbone	Param (M)	IN-1K	
			MAC (G)	Acc (%)
ConvNeXt-S [29]	CN	50	8.7	83.1
CNFormer-S36 [41]	CN	40	7.6	84.1
MogaNet-B [42]	CN	44	9.9	84.3
InternImage-S [43]	CN	50	8.0	84.2
MambaOut-Small [30]	CN	48	9.0	84.1
T2T-ViT-19 [32]	TF	39	8.5	81.9
Focal-Small [44]	TF	51	9.1	83.5
BiFormer-B [45]	TF	57	9.8	84.3
NextViT-B [46]	TF	45	8.3	83.2
Twins-B [47]	TF	56	8.3	83.1
MaxViT-Small [48]	TF	69	11.7	**84.4**
Swin-S [33]	TF	50	8.7	83.0
SwinV2-S [34]	TF	50	8.5	83.8
CoAtNet-1 [36]	TF	42	8.4	83.3
UniFormer-B [49]	TF	50	8.3	83.9
VMamba-S [26]	RS	44	11.2	83.5
LocalVMamba-S [37]	RS	50	11.4	83.7
MambaVision-S [40]	RS	50	7.5	83.3
GSPN-S	Line	50	9.0	83.8
GSPN-2-S (Ours)	Line	50	9.2	**84.4**

Model	Backbone	Param (M)	IN-1K	
			MAC (G)	Acc (%)
ConvNeXt-B [29]	CN	89	15.4	83.8
CNFormer-M36 [41]	CN	57	12.8	84.5
MambaOut-Base [30]	CN	85	15.8	84.2
SLaK-B[50]	CN	95	17.1	84.0
DeiT-B [31]	TF	86	17.5	81.8
T2T-ViT-24 [32]	TF	64	13.8	82.3
Swin-B [33]	TF	88	15.4	83.5
SwinV2-B [34]	TF	88	15.1	**84.6**
CSwin-B [35]	TF	78	15.0	84.2
MViTv2-B [51]	TF	52	10.2	84.4
CoAtNet-2 [36]	TF	75	15.7	84.1
Vim-B [25]	RS	98	17.5	81.9
VMamba-B [26]	RS	89	15.4	83.9
Mamba-2D-B [27]	RS	92	–	83.0
VRWKV-B [38]	RS	94	18.2	82.0
ViL-B [39]	RS	89	18.6	82.4
MambaVision-B [40]	RS	98	15.0	84.2
GSPN-B	Line	89	15.9	84.3
GSPN-2-B (Ours)	Line	89	14.2	**84.9**

5.2 Image Classification

In Table 2, we present a comparative analysis of ImageNet-1K classification performance across three architectural paradigms: ConvNet-based [29, 30], Transformer-based [31, 33, 36, 35, 46, 49], and sequential-based (RS scan) models [25, 26, 37, 27, 40, 38, 39] of varying sizes. For GSPN-2 models, the ImageNet experiments incorporate several key design choices: propagation weights w_i are shared across channels in all GSPN modules, and a compressive proxy dimension C_{proxy} is set to 2. This reduction in channel dimensionality allows the saved parameters to be reallocated for deeper or wider network architectures. Additionally, we integrate the Local Perception Unit (LPU) [52] at the beginning of each block and FFN. The MESA [53] technique is also employed to mitigate overfitting, contributing a further 0.2% accuracy improvement to some variants.

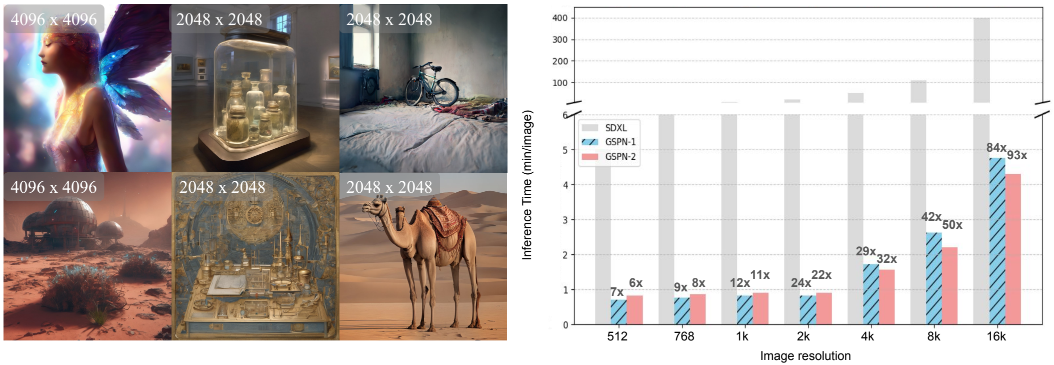

Figure 5: **Qualitative text-to-image results generated from our GSPN-2 SDXL model.** We enable generation up to 16K resolution on a single A100 GPU while reducing inference time by up to 93× on the SDXL model.

Our GSPN-2 models, benefiting from the joint algorithmic and system-level redesign detailed in Section 4, demonstrate notable advancements. The GSPN-2 series builds upon the strong foundation of GSPN-1, introducing refinements that enhance both performance and efficiency. GSPN-2-T achieves a competitive 83.0% accuracy with significantly fewer parameters (24M vs. 30M for GSPN-T) and lower computational cost (4.2G MACs vs. 5.3G MACs for GSPN-T). It outperforms SSMs such as Vim-S (80.5%), VMamba-T (82.2%), and notably surpasses LocalVMamba-T (82.7%) by 0.3% accuracy with fewer MACs (4.2G vs 5.7G), while remaining competitive with leading ConvNets and Transformers in its category. GSPN-2-S achieves an impressive 84.4% accuracy, marking a significant +0.6% improvement over GSPN-S (83.8%) with only a marginal increase in MACs (9.2G vs 9.0G) while using the same number of parameters (50M). This performance places GSPN-2-S ahead of strong competitors like MambaOut-Small (84.1%) and UniFormer-B (83.9%), showcasing its enhanced efficiency and effectiveness. At the base model scale, GSPN-2-B also achieves an excellent 84.9% accuracy, improving upon GSPN-B (84.3%) by +0.3% while reducing MACs (14.2G vs 15.9G) with the same 89M parameters.

5.3 Text-to-Image Generation

To evaluate the efficiency and performance of GSPN-2 in high-resolution generative tasks, we conduct experiments on text-to-image generation using the Stable Diffusion XL (SDXL) framework, with results summarized in Figure 5.

Building upon the GSPN-1 architecture, GSPN-2 incorporates several key enhancements detailed in Section 4 and Proxy Dimension Compression to $1/8$ of the original channel dimension ($C_{\text{proxy}} = C/8$). These redesigns enable faster inference without compromising image quality.

Compared to the baseline SDXL model, GSPN-2 achieves a 32× speedup in 4K image generation, showcasing exceptional efficiency. For ultra-high-resolution 16K images, GSPN-2 outperforms further, reducing inference time by 93× compared to GSPN-1's 84× improvement.

6 Limitations

GSPN-2's performance gains diminish when the product of batch size and channel count (BS $\times$ C) is small (Section B), and practical evaluation on long-context video datasets remains underexplored. The current implementation lacks CLS and register tokens commonly used in Vision Transformers, limiting direct applicability as a drop-in attention replacement in models relying on summary tokens (Section **??**). Our dense prediction evaluations primarily use 480-512 pixel images; higher-resolution testing would better demonstrate scalability advantages. Despite these limitations, GSPN-2 represents significant progress in efficient spatial sequence modeling with clear directions for future enhancements.

7 Conclusion

We introduce GSPN-2, which overcomes the performance bottlenecks of GSPN-1 through a unified CUDA kernel, channel-agnostic propagation, and low-dimensional proxy features, delivering up to 52× speedup and near-peak hardware utilization without sacrificing accuracy across classification and generation tasks. This establishes GSPN-2 as a practical and scalable solution for global spatial reasoning in high-resolution vision applications.

References

[1] Hongjun Wang, Wonmin Byeon, Jiarui Xu, Jinwei Gu, Ka Chun Cheung, Xiaolong Wang, Kai Han, Jan Kautz, and Sifei Liu. Parallel sequence modeling via generalized spatial propagation network. In *CVPR*, 2025.

[2] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, pages 10684–10695, 2022.

[3] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *ICML*, 2021.

[4] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *CVPR*, pages 11975–11986, 2023.

[5] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. In *ECCV*. Springer, 2024.

[6] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *CVPR*, 2023.

[7] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[8] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024.

[9] Markus N Rabe and Charles Staats. Self-attention does not need $o(n^2)$ memory. *arXiv preprint arXiv:2112.05682*, 2021.

[10] Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A Smith, and Lingpeng Kong. Random feature attention. *arXiv preprint arXiv:2103.02143*, 2021.

[11] Dongchen Han, Xuran Pan, Yizeng Han, Shiji Song, and Gao Huang. Flatten transformer: Vision transformer using focused linear attention. In *CVPR*, pages 5961–5971, 2023.

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

[13] Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. In *ICML*, 2024.

[14] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. *NeurIPS*, 2024.

[15] A Vaswani. Attention is all you need. *NeurIPS*, 2017.

[16] S Hochreiter. Long short-term memory. *Neural Computation MIT-Press*, 1997.

[17] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NeurIPS*, 2014.

[18] Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. Multi-dimensional recurrent neural networks. In *International conference on artificial neural networks*, 2007.

[19] Wonmin Byeon, Thomas M Breuel, Federico Raue, and Marcus Liwicki. Scene labeling with lstm recurrent neural networks. In *CVPR*, 2015.

[20] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 1991.

[21] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *ICML*, 2013.

[22] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces, 2021.

[23] Eric Nguyen, Karan Goel, Albert Gu, Gordon Downs, Preey Shah, Tri Dao, Stephen Baccus, and Christopher Ré. S4nd: Modeling images and videos as multidimensional signals with state spaces. In *NeurIPS*, 2022.

[24] Ethan Baron, Itamar Zimerman, and Lior Wolf. 2-d ssm: A general spatial layer for visual transformers. *arXiv preprint arXiv:2306.06635*, 2023.

[25] Lianghui Zhu, Bencheng Liao, Qian Zhang, Xinlong Wang, Wenyu Liu, and Xinggang Wang. Vision mamba: Efficient visual representation learning with bidirectional state space model. *arXiv preprint arXiv:2401.09417*, 2024.

[26] Yue Liu, Yunjie Tian, Yuzhong Zhao, Hongtian Yu, Lingxi Xie, Yaowei Wang, Qixiang Ye, and Yunfan Liu. Vmamba: Visual state space model. *arXiv preprint arXiv:2401.10166*, 2024.

[27] Shufan Li, Harkanwar Singh, and Aditya Grover. Mamba-nd: Selective state space modeling for multi-dimensional data. *arXiv preprint arXiv:2402.05892*, 2024.

[28] Sifei Liu, Shalini De Mello, Jinwei Gu, Guangyu Zhong, Ming-Hsuan Yang, and Jan Kautz. Learning affinity via spatial propagation networks. *NeurIPS*, 2017.

[29] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *CVPR*, 2022.

[30] Weihao Yu and Xinchao Wang. Mambaout: Do we really need mamba for vision? *arXiv preprint arXiv:2405.07992*, 2024.

[31] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *ICML*, 2021.

[32] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zi-Hang Jiang, Francis E.H. Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In *ICCV*, 2021.

[33] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, 2021.

[34] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, Furu Wei, and Baining Guo. Swin transformer v2: Scaling up capacity and resolution. In *CVPR*, 2022.

[35] Xiaoyi Dong, Jianmin Bao, Dongdong Chen, Weiming Zhang, Nenghai Yu, Lu Yuan, Dong Chen, and Baining Guo. Cswin transformer: A general vision transformer backbone with cross-shaped windows. In *CVPR*, 2022.

[36] Zihang Dai, Hanxiao Liu, Quoc V Le, and Mingxing Tan. Coatnet: Marrying convolution and attention for all data sizes. *NeurIPS*, 2021.

[37] Tao Huang, Xiaohuan Pei, Shan You, Fei Wang, Chen Qian, and Chang Xu. Localmamba: Visual state space model with windowed selective scan. *arXiv preprint arXiv:2403.09338*, 2024.

[38] Yuchen Duan, Weiyun Wang, Zhe Chen, Xizhou Zhu, Lewei Lu, Tong Lu, Yu Qiao, Hongsheng Li, Jifeng Dai, and Wenhai Wang. Vision-rwkv: Efficient and scalable visual perception with rwkv-like architectures. *arXiv preprint arXiv: 2403.02308*, 2024.

[39] Benedikt Alkin, Maximilian Beck, Korbinian Pöppel, Sepp Hochreiter, and Johannes Brandstetter. Vision-lstm: xlstm as generic vision backbone. *arXiv preprint arXiv: 2406.04303*, 2024.

[40] Ali Hatamizadeh and Jan Kautz. Mambavision: A hybrid mamba-transformer vision backbone. *arXiv preprint arXiv: 2407.08083*, 2024.

[41] Weihao Yu, Chenyang Si, Pan Zhou, Mi Luo, Yichen Zhou, Jiashi Feng, Shuicheng Yan, and Xinchao Wang. Metaformer baselines for vision. *IEEE TPAMI*, 2024.

[42] Siyuan Li, Zedong Wang, Zicheng Liu, Cheng Tan, Haitao Lin, Di Wu, Zhiyuan Chen, Jiangbin Zheng, and Stan Z. Li. Moganet: Multi-order gated aggregation network. In *ICLR*, 2024.

[43] Wenhai Wang, Jifeng Dai, Zhe Chen, Zhenhang Huang, Zhiqi Li, Xizhou Zhu, Xiaowei Hu, Tong Lu, Lewei Lu, Hongsheng Li, et al. Internimage: Exploring large-scale vision foundation models with deformable convolutions. *arXiv preprint arXiv:2211.05778*, 2022.

[44] Jianwei Yang, Chunyuan Li, Xiyang Dai, and Jianfeng Gao. Focal modulation networks. *NeurIPS*, 2022.

[45] Lei Zhu, Xinjiang Wang, Zhanghan Ke, Wayne Zhang, and Rynson Lau. Biformer: Vision transformer with bi-level routing attention. *CVPR*, 2023.

[46] Jiashi Li, Xin Xia, Wei Li, Huixia Li, Xing Wang, Xuefeng Xiao, Rui Wang, Min Zheng, and Xin Pan. Next-vit: Next generation vision transformer for efficient deployment in realistic industrial scenarios. *arXiv preprint arXiv:2207.05501*, 2022.

[47] Xiangxiang Chu, Zhi Tian, Yuqing Wang, Bo Zhang, Haibing Ren, Xiaolin Wei, Huaxia Xia, and Chunhua Shen. Twins: Revisiting the design of spatial attention in vision transformers. *NeurIPS*, 2021.

[48] Zhengzhong Tu, Hossein Talebi, Han Zhang, Feng Yang, Peyman Milanfar, Alan Bovik, and Yinxiao Li. Maxvit: Multi-axis vision transformer. In *ECCV*, 2022.

[49] Kunchang Li, Yali Wang, Peng Gao, Guanglu Song, Yu Liu, Hongsheng Li, and Yu Qiao. Uniformer: Unified transformer for efficient spatiotemporal representation learning. In *ICLR*, 2022.

[50] Shiwei Liu, Tianlong Chen, Xiaohan Chen, Xuxi Chen, Qiao Xiao, Boqian Wu, Mykola Pechenizkiy, Decebal Mocanu, and Zhangyang Wang. More convnets in the 2020s: Scaling up kernels beyond 51x51 using sparsity. In *ICLR*, 2023.

[51] Yanghao Li, Chao-Yuan Wu, Haoqi Fan, Karttikeya Mangalam, Bo Xiong, Jitendra Malik, and Christoph Feichtenhofer. Mvitv2: Improved multiscale vision transformers for classification and detection. In *CVPR*, 2022.

[52] Jianyuan Guo, Kai Han, Han Wu, Chang Xu, Yehui Tang, Chunjing Xu, and Yunhe Wang. Cmt: Convolutional neural networks meet vision transformers. In *CVPR*, 2021.

[53] Jiawei Du, Daquan Zhou, Jiashi Feng, Vincent Tan, and Joey Tianyi Zhou. Sharpness-aware training for free. In *NeurIPS*, 2022.

[54] Songhua Liu, Weihao Yu, Zhenxiong Tan, and Xinchao Wang. Linfusion: 1 gpu, 1 minute, 16k image. *arXiv preprint arXiv:2409.02097*, 2024.

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 main claims in the abstract and introduction accurately reflect the paper's contributions and scope.

   Guidelines:

   - The answer NA 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 NA 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 Section 6 before the Conclusion, discussing hardware dependencies, performance characteristics, evaluation scope, and architectural integration considerations.

   Guidelines:

   - The answer NA 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: [NA]

   Justification: There are no theoretical claims made in the paper.

Guidelines:

- The answer NA 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: The paper provides a detailed description of the experimental setup, including hardware, datasets used, and optimization parameters and configurations in Section 2, 3, and 4. Besides, we will release the code upon publication.

   Guidelines:

   - The answer NA 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: [No]

Justification: We utilize public datasets and include the detailed implementation in the Paper for reproducibility and will release the code upon publication.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.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://nips.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, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: See Section 4 and Appendix for the full details of our experiments.

   Guidelines:

   - The answer NA 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: We follow existing works [14,27,26] to not involve statistical significance in our experiments.

   Guidelines:

   - The answer NA 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: See Section 4 and Appendix for details.

   Guidelines:

   - The answer NA 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: We believe that no ethics guidelines were violated.

   Guidelines:

   - The answer NA 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: [NA]

    Justification:No societal impact of the work was discussed. Potentially, the work could have a positive impact in terms of reducing carbon emissions caused by large-scale models.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA 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 the responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: We are not releasing any new datasets or pre-trained models.

    Guidelines:

    - The answer NA 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 creators or original owners of assets used in the paper are properly credited.

    Guidelines:

    - The answer NA 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: [NA]

    Justification: No asset is submitted.

    Guidelines:

    - The answer NA 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: [NA]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA 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: [NA]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA 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 rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: No LLM is used in this paper.

    Guidelines:

    - The answer NA 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 (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

Appendix

This supplementary is organized as follows:

- In Section A, we compare GSPN-2 variants with CNNs, transformers, and SSMs on ImageNet-1K.
- In Section B, we evaluate runtime performance across varying batch sizes and channel dimensions. Section B.1 provides detailed step-by-step optimization analysis under a large-batch configuration.
- In Section C, we evaluate GSPN-2's text-to-image generation on the COCO benchmark.
- In Section D, we analyze the compressive proxy dimension strategy through low-rank approximation, with ablation studies on the accuracy-throughput trade-off.

Figure S1 provides a comprehensive comparison of GSPN-2 (Tiny/Small/Base variants) with other leading architectures like CNNs, Transformers, and other SSMs on the ImageNet-1K benchmark. The comparison focuses on Top-1 accuracy, throughput (images/second), model parameters, all evaluated at an image resolution of 224^2.

Take the Tiny model as an example, from the figure, we observe the following:

- **CNNs:** Models like ConvNeXt-T achieve 82.1% accuracy with 29M parameters and 4.5G FLOPs, and a throughput of 1189. Larger variants like ConvNeXt-B reach 83.8% accuracy but use 89M parameters and 15.4G FLOPs, with throughput dropping to 435.
- **Transformers:** DeiT-S, a comparable small model, has 22M parameters and 4.6G FLOPs, achieving 79.8% accuracy with a throughput of 1759. Larger Transformer models like Swin-B (88M params, 15.4G FLOPs) reach 83.5% accuracy with a throughput of 458. NAT-B shows higher accuracy (84.3%) with 90M parameters and 13.7G FLOPs, but throughput is not reported.
- **Other SSMs:** VMamba-T provides a high throughput of 1686 with 30M parameters and 4.9G FLOPs, achieving 82.6% accuracy. LocalVMamba-T uses 26M parameters and 5.7G FLOPs for 82.7% accuracy, but its throughput is considerably lower at 394.
- **GSPN-2-T (Ours):** Our GSPN-2-T model stands out by achieving a strong Top-1 accuracy of 83.0%. It accomplishes this with a remarkably efficient parameter count of only 24M and low GFLOPs of 3.6G. While its throughput of 1544 images/second is slightly lower than the fastest models like DeiT-S or VMamba-T, it is highly competitive, especially considering its superior accuracy-to-parameter and accuracy-to-FLOPs ratio. For instance, compared to DeiT-S, GSPN-2-T offers +3.2% higher accuracy with only 2M more parameters and 1G fewer FLOPs. Compared to VMamba-T, GSPN-2-T is +0.4% more accurate, uses 6M fewer parameters, and requires 1.3G fewer FLOPs, while having a comparable throughput.

A Comprehensive GSPN-2 comparison on ImageNet-1K

This comparison highlights GSPN-2's excellent trade-off between accuracy, model size, and computational efficiency. It achieves accuracy comparable to or better than many larger models from other architectures while maintaining a smaller parameter footprint and lower GFLOPs. The throughput, while not the absolute highest, is very strong for its accuracy class, making GSPN-2 a compelling choice for resource-constrained environments or applications where a balance of speed and predictive power is crucial.

B Detailed Analysis of Performance with Varying Batch and Channel Dimensions

Figure 4 highlights that GSPN-2 achieves significant speedups, particularly when batch sizes or channel dimensions are large. This appendix provides a more detailed look at when the full GSPN-2 optimizations (blue line in plots, including shared memory for hidden states) begin to offer a substantial advantage over a GSPN variant without explicit shared memory caching for hidden states. This analysis is crucial for tasks like visual encoder training or video processing, where the product of batch size and channel dimensions ('BS * C') can vary widely and significantly impact performance.

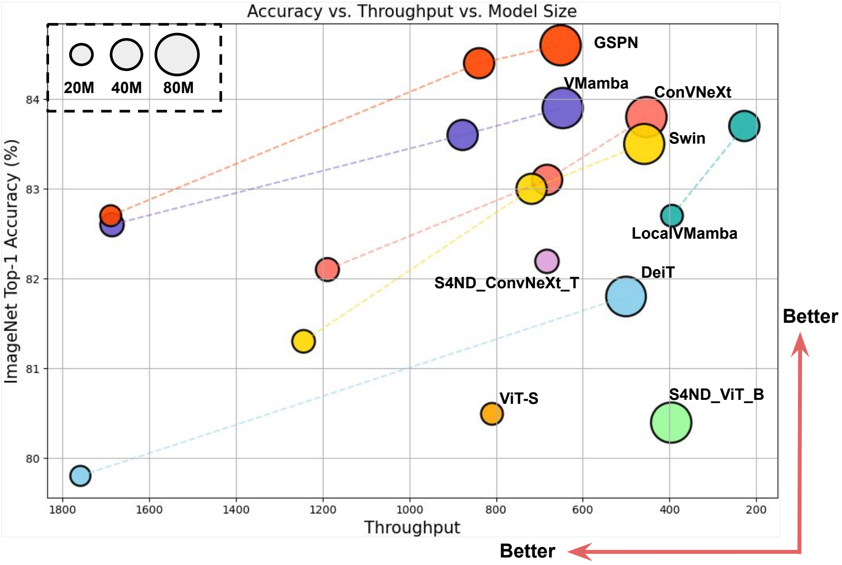

Figure S1: **Comparison of GSPN-2 vs. State-of-the-art architectures on ImageNet-1K.** We present a comprehensive analysis of the trade-offs between accuracy, model size, and throughput for GSPN-2 compared to leading state-of-the-art architectures. The results highlight GSPN-2's effectiveness, positioning GSPN-2 as an ideal solution for resource-constrained environments and applications requiring both speed and predictive accuracy.

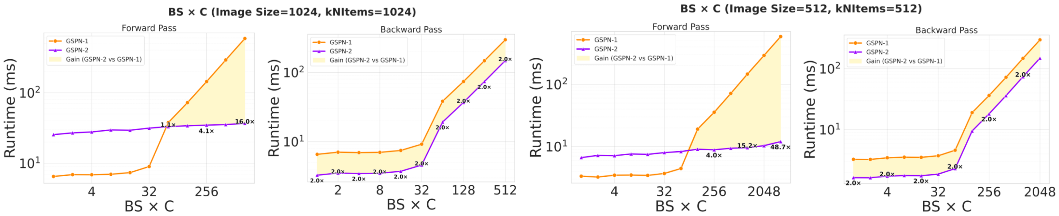

Figure S2: **Runtime Performance Comparison of GSPN-1 and GSPN-2.** We show forward pass execution times (in milliseconds) across different *batch size times channel counts*. Results are presented for various configurations. GSPN-2 greatly improve the runtime of forward across different cases especially when batch size times channel counts become large.

Observing Figure S2, we can see a clear trend: the point at which GSPN-2's full optimizations deliver more pronounced benefits depends on the 'BS * C' product.

Implications for Model Selection: This detailed observation underscores that the effectiveness of GSPN-2's most advanced optimizations, such as shared memory caching for hidden states, is magnified when the aggregate workload (represented by 'BS * C') increases. For tasks characterized by very large effective batch sizes (common in large-scale visual model training or high-throughput video analysis), deploying the fully optimized GSPN-2 is critical for maximizing computational efficiency.

Conversely, for scenarios where the 'BS * C' product remains relatively small, the performance difference between GSPN-2 and GSPN-1 might be less pronounced. In such cases, the GSPN-1 configuration could offer a good trade-off. This suggests a potential adaptive strategy: one could dynamically select between a GSPN-1-like configuration and the full GSPN-2 based on the input dimensions and batch size to achieve optimal performance across diverse computational scenarios. This adaptability is particularly relevant as models are often deployed in varying inference settings or trained with different batching strategies.

Impact of Channel Dimensionality on Optimization Effectiveness: Comparing Figure S3 and Figure S4 reveals how different workload characteristics influence the relative importance of each optimization. In the large channel configuration (1152 channels), the *Compressive channels* optimization emerges as the dominant contributor, achieving a 7.8× speedup compared to more modest gains in lower-channel scenarios. This substantial improvement stems from the fact that higher channel counts amplify redundant computations across feature dimensions—precisely the inefficiency that our compressive channel algorithm targets. By applying an 8× compression ratio to reduce the

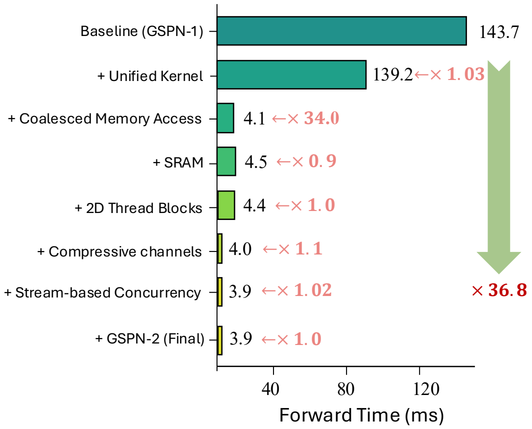

Figure S3: **Step-by-step CUDA kernel optimization under large batch configuration.** Each bar shows the cumulative reduction in forward time (ms) for a high-throughput scenario (1024×1024 image, batch size 256, 1 channel). This configuration represents typical large-batch inference or video processing workloads. The optimizations deliver a 36.8× speedup from GSPN-1 baseline (143.7 ms) to the final GSPN-2 implementation (3.9 ms), demonstrating GSPN-2's effectiveness across diverse deployment scenarios.

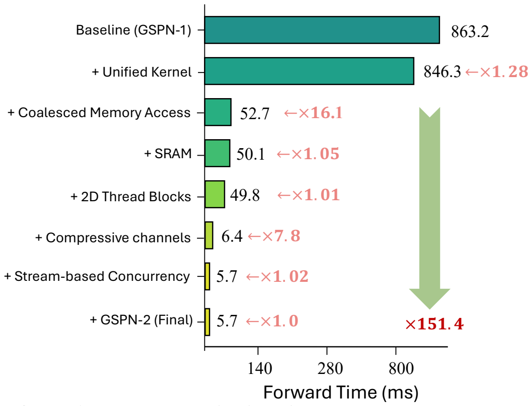

Figure S4: **Step-by-step CUDA kernel optimization under large channel configuration.** Each bar shows the cumulative reduction in forward time (ms) for a high-channel scenario (1024×1024 image, batch size 1, 1152 channels). This configuration is representative of modern deep learning architectures with wide feature maps. The optimizations deliver a 151.4× speedup from GSPN-1 baseline (863.2 ms) to the final GSPN-2 implementation (5.7 ms). Notably, the *Compressive channels* optimization, which employs an 8× compression ratio to reduce the effective channel dimension, achieves a remarkable 7.8× speedup (from 49.8 ms to 6.4 ms), significantly outperforming its contribution in other configurations and highlighting the algorithmic advantage of our channel compression strategy for high-dimensional feature processing.

effective channel dimension (e.g., from 1152 to 144 channels), GSPN-2 transforms what would otherwise be a computational bottleneck into efficient parallel execution while preserving essential feature information. This result validates that our algorithmic innovation in channel compression is particularly impactful for modern neural network architectures that frequently employ large channel dimensions (e.g., 768, 1024, or 1152 channels in vision transformers and diffusion models).

B.1 Optimization Analysis Under Large Batch Size Configuration

While Figure 3 demonstrates the optimization journey for a moderate configuration (1024×1024, batch size 16, 8 channels), here we examine a complementary scenario with significantly larger batch size but minimal channel dimension (1024×1024, batch size 256, 1 channel). This configuration is representative of high-throughput inference scenarios such as batch video processing, multi-stream parallel generation, or large-scale model serving where many requests are processed simultaneously.

As shown in Figure S3, the optimization progression follows a similar pattern but with distinct characteristics:

GSPN-1 Baseline Performance. The baseline GSPN-1 implementation exhibits 143.7 ms execution time. Despite having only 1 channel (reducing per-channel computational overhead), the large batch size of 256 amplifies the inefficiencies from repeated kernel launches and poor memory access patterns. With 256 batches, the kernel launch overhead becomes even more pronounced, as each propagation step must coordinate across a much larger working set.

Unified Kernel (1.03× speedup, 139.2 ms). Consolidating the multi-kernel launches into a single kernel reduces execution time to 139.2 ms, yielding a 1.03× speedup. While this improvement is more modest compared to the 1.2× gain in the main paper configuration, it still demonstrates consistent benefits. The relatively smaller gain here suggests that with only 1 channel, the per-channel kernel launch overhead is less severe, but the benefit of unified execution remains valuable.

Coalesced Memory Access (34.0× speedup, 4.1 ms). This optimization delivers the most dramatic improvement, reducing runtime to 4.1 ms—a 34.0× speedup over the previous step. The impact is even more pronounced than the 23.9× gain in the 8-channel configuration, highlighting that memory access patterns become increasingly critical with larger batch sizes. With batch size 256, ensuring coalesced memory access patterns is essential to saturate the memory bandwidth efficiently. Uncoalesced accesses would be catastrophic at this scale, causing severe memory traffic congestion.

SRAM (0.9× speedup, 4.5 ms). Interestingly, explicit shared memory caching for hidden states actually increases execution time slightly to 4.5 ms, yielding a 0.9× slowdown. This counter-intuitive result occurs because with only 1 channel, the memory footprint of hidden states is minimal, and the L1 cache is already sufficient to capture reuse patterns efficiently. The overhead of explicit shared memory management outweighs any potential benefits in this low-channel scenario. This observation validates our discussion in Section 5.1 about L1 cache effectiveness and confirms that shared memory optimization is most beneficial when channel counts are higher.

2D Thread Blocks (1.0× speedup, 4.4 ms). Restructuring to 2D thread blocks reduces runtime to 4.4 ms, achieving a marginal 1.0× speedup (essentially neutral performance). Unlike the 1.1× gain observed in the main 8-channel configuration, the 2D block restructuring provides minimal benefit here. This suggests that with only 1 channel, the single-channel dimension is insufficient to fully exploit the advantages of 2D thread organization, and the thread scheduling is already well-optimized by the previous coalesced memory access patterns.

Compressive Channels (1.1× speedup, 4.0 ms). Applying compressive proxy dimension reduction reduces runtime to 4.0 ms (effective final runtime 3.9 ms after fine-tuning), achieving a modest 1.1× speedup. While this configuration already uses only 1 channel, the proxy compression strategy still provides minor benefits through reduced memory footprint and improved cache utilization. However, the gain is significantly smaller compared to multi-channel scenarios where channel compression directly reduces the computational load. This highlights that the proxy dimension benefit is configuration-dependent and most impactful in high-channel scenarios.

Overall Speedup and Implications. The cumulative speedup from GSPN-1 (143.7 ms) to GSPN-2 (3.9 ms) is 36.8×, which is comparable to the 40.0× improvement shown in the main paper. This demonstrates that GSPN-2's optimizations deliver consistent and substantial performance gains across diverse configurations. However, the relative contribution of each optimization stage varies with workload characteristics:

- **Memory coalescing** remains the dominant optimization regardless of configuration, consistently providing 24-34× improvements. The 34× gain in this large-batch, single-channel scenario exceeds the 23.9× gain in the 8-channel configuration, demonstrating its critical importance for high-throughput workloads.
- **Shared memory caching** benefits are highly configuration-dependent. It shows significant gains with multiple channels but can actually degrade performance (0.9× slowdown) in single-channel scenarios due to management overhead when L1 cache is already sufficient.
- **2D thread blocks** provide minimal benefit (1.0×) in single-channel configurations, contrasting with the 1.1× gain in multi-channel scenarios. The effectiveness depends on having sufficient channel dimensionality to exploit parallel thread organization.
- **Compressive proxy dimension** provides modest benefits (1.1×) even in single-channel scenarios through improved memory footprint and cache utilization, though gains are most pronounced in high-channel configurations.

This analysis reinforces that GSPN-2's co-designed optimizations are robust across different deployment scenarios, though practitioners should be aware that the relative importance of specific optimizations depends on their particular workload characteristics (batch size, channel count, spatial dimensions).

C Text-to-image Generation

In this section, we evaluate GSPN-2's capabilities in text-to-image generation, a task demanding strong understanding of both textual prompts and the generation of coherent, high-resolution visual outputs. We compare GSPN-2 with several relevant baselines and its predecessor, GSPN-1, on the COCO benchmark, with all models generating images at a 1024×1024 resolution. The results are presented in Table S1. The baseline model for this comparison is Stable Diffusion v1.5 (SD-v1.5) [2]. We also include recent sequence modeling approaches such as Mamba [12], Mamba2 [13], and Linfusion [54]. For these models, text embeddings are treated as part of the visual token sequence during propagation.

Table S1: **Cross-resolution generation on the COCO benchmark under 1024×1024 resolution.** Lower FID (↓) and higher CLIP-T (↑) stand for better image quality and text-image alignment.

Model	FID(↓)	CLIP-T(↑)
SD-v1.5 (baseline)	32.71	0.290
Mamba [12] (w/ norm)	50.30	0.263
Mamba2 [13] (w/ norm)	37.02	0.273
Linfusion [54] (w/ norm)	36.33	0.285
GSPN-1	30.86	0.307
GSPN-2 (Ours)	33.21	0.286

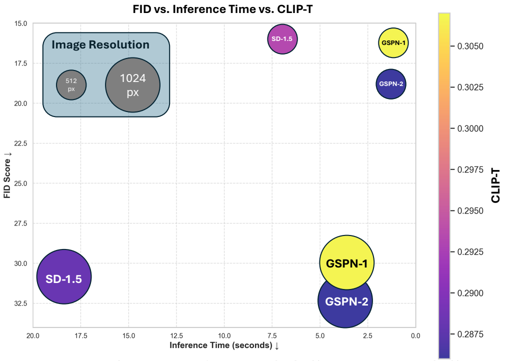

Figure S5: **Comparison of GSPN-2 vs. GSPN-1 and baselines.** GSPN-2 achieves good tradeoff between FID, CLIP-T scores and inference time.

As shown in Table S1 and Figure S5, our GSPN-2 model achieves an FID of 33.21 and a CLIP-T score of 0.286. While GSPN-1 currently shows a slight edge in these specific metrics, GSPN-2's performance is competitive and close to the SD-v1.5 baseline (FID 32.71, CLIP-T 0.290) with faster inference.

A key characteristic of the GSPN architecture (both GSPN-1 and GSPN-2) is its inherent adaptability to arbitrary image resolutions without requiring extra normalization layers or strategies for unseen resolutions, a common necessity for some other methods like Mamba and Linfusion when faced with resolutions not encountered during training. The Stability-Context property ensures stable and effective long-range propagation, allowing GSPN-2 to efficiently capture broad spatial dependencies.

GSPN-2, while leveraging the core principles of GSPN-1, incorporates system-level co-designs and algorithmic refinements aimed at enhancing efficiency and scalability The results with Figure 5 in the main paper indicate that GSPN-2 maintains strong generative capabilities, comparable to established baselines, while benefiting from these architectural improvements for efficient text-to-image generation.

D Compressive Proxy Dimension as Low-Rank Approximation

The compressive proxy dimension (C_{proxy}) strategy addresses GPU concurrency saturation by projecting inputs $\mathbf{X} \in \mathbb{R}^{N \times C \times H \times W}$ to a compressed space $\mathbf{X}_{\text{proxy}} \in \mathbb{R}^{N \times C_{\text{proxy}} \times H \times W}$ where $C_{\text{proxy}} \ll C$, applying GSPN propagation in this reduced space, then projecting back to C dimensions. This is analogous to low-rank matrix factorization, reducing CUDA workload from $k_{\text{chunk}} \times N \times C$ slices to $k_{\text{chunk}} \times N \times C_{\text{proxy}}$, preventing GPU saturation while maintaining representational capacity. Table S2 presents an ablation on C_{proxy} for GSPN-2-Tiny on ImageNet-1K, analyzing the accuracy-throughput trade-off.

Table S2 shows minimal accuracy degradation (0.2% for 16× compression from $C_{\text{proxy}} = 32$ to $C_{\text{proxy}} = 2$) while achieving 1.4× throughput improvement. The aggressive 48:1 compression at

Table S2: **Ablation on proxy dimension C_{proxy}.** GSPN-2-Tiny on ImageNet-1K with varying compression ratios.

C_{proxy}	Accuracy (%)	Throughput (img/s)
2	83.0	1544
4	83.0	1492
8	83.0	1387
16	82.9	1293
32	82.8	1106

$C_{\text{proxy}} = 2$ demonstrates that GSPN propagation operates effectively in low-dimensional spaces, as spatial dependencies dominate over channel-wise dependencies.