CHUNKFLOW: COMMUNICATION-AWARE CHUNKED PREFETCHING FOR LAYERWISE OFFLOADING IN DISTRIBUTED DIFFUSION TRANSFORMER INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Layerwise offloading reduces the GPU memory footprint of large diffusion transformer (DiT) inference by prefetching upcoming layers from host memory, but its effectiveness hinges on hiding prefetch latency behind per-layer computation. This assumption breaks down when the per-GPU compute workload is small. Moreover, on PCIe-only nodes, prefetch and inter-GPU collective communications such as all-gather and all-to-all contend on the shared PCIe path, exposing prefetch latency even when compute would otherwise hide it. We revisit layerwise offloading as a co-scheduling problem between prefetch and communication, guided by a first-order analytical model that predicts when prefetch can be hidden by computation. Building on this model, we design ChunkFlow, a communication-aware, chunk-granular offloading runtime that adaptively yields to collective communication and smoothly trades GPU memory for prefetch volume. On three representative diffusion transformers running on commodity L40S and H100 GPUs over PCIe with Ulysses sequence parallelism, ChunkFlow delivers up to $1.32\times$ step-time speedup over SGLang's existing layerwise offloading, reduces peak GPU memory by up to 49% over the no-offload baseline at near-identical step time once the workload is large enough, and exposes a tunable memory–latency tradeoff that recovers near-zero step-time overhead in the small-workload regime.

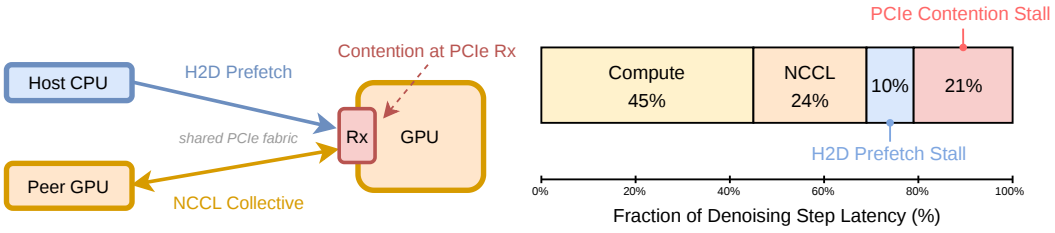
1 INTRODUCTION

Diffusion models (Ho et al., 2020; Song et al., 2021b) have become one of the dominant paradigms for generative modeling, powering state-of-the-art systems in image (Rombach et al., 2022; Podell et al., 2024) and video (Brooks et al., 2024; Kong et al., 2024; Team Wan et al., 2025) generation. Modern systems increasingly adopt Transformer-based denoisers (Vaswani et al., 2017; Dosovitskiy et al., 2021), commonly referred to as Diffusion Transformers (DiTs) (Peebles & Xie, 2023; Chen et al., 2024; Esser et al., 2024), which scale to tens of billions of parameters and unify architectures across modalities. A key bottleneck in serving large DiTs is GPU memory capacity. Unlike LLMs, where sequence lengths are typically on the order of 10^3–10^4 tokens, DiTs operate on significantly longer sequences due to spatial and temporal tokenization, often reaching 10^5 to 10^6 tokens for high-resolution images and videos. For example, generating a single $1280\times720\times129$-frame video with the 13B-parameter HunyuanVideo (Kong et al., 2024) requires distributing the model across 8 H100 GPUs via sequence parallelism using the official xDiT (Fang et al., 2024b) configuration. This scaling is primarily driven by the memory footprint of long-sequence attention—activations and KV buffers on top of the model weights—rather than intrinsic compute demand.

A standard remedy to the above problem is *layerwise offloading*, which stores weights in host memory and asynchronously prefetches them to the GPU memory one layer ahead. By keeping only two layers on the GPU memory, this solution substantially reduces peak memory, and hence is adopted in production-quality inference systems such as SGLang (Zheng et al., 2024) and vLLM (Kwon et al., 2023).

The efficiency of layerwise offloading depends on whether the per-layer compute time is long enough to hide the host-to-device (H2D) prefetch time. Whether this overlap is achievable is determined by per-GPU compute throughput and host-to-device bandwidth. When the overlap is achievable,

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



(a) Data-path sharing.

(b) Latency breakdown.

Figure 1: PCIe contention on PCIe-based nodes during distributed DiT inference. (a) H2D prefetch and inter-GPU collectives converge at each GPU's PCIe receive port (Rx) and contend for bandwidth. (b) Denoising step latency breakdown: the contention stall (21%) accounts for a large portion.

offloading reduces memory without inflating latency; otherwise, the unfinished prefetch is exposed to the critical path and slows down inference.

In commodity multi-GPU deployments—e.g., $8\times$L40 or $8\times$A6000 PCIe nodes widely used in cloud and on-premise environments (Jiang et al., 2025; Deng et al., 2025; Unat et al., 2024)—intra-node GPU-to-GPU traffic traverses PCIe rather than a dedicated high-bandwidth fabric (NVLink). On such PCIe-only nodes, distributed inference introduces an additional issue beyond the baseline overlap analysis above: H2D prefetch and inter-GPU collective communications—the all-to-all and all-gather of sequence parallelism, the standard way DiT inference engines distribute a model across GPUs (Jacobs et al., 2023; Fang et al., 2024b)—traverse the same PCIe fabric and converge at each GPU's PCIe receive port (Rx), as illustrated in Figure 1a. This *data-path sharing* can significantly slow down the prefetching. For example, with Ulysses sequence parallelism over PCIe, enabling existing layerwise offloading at the official default resolutions inflates denoising step time by up to $2.3\times$, $2.3\times$, and $1.9\times$ on WanVideo (5B) (Team Wan et al., 2025), HunyuanVideo (13B) (Kong et al., 2024), and Flux (12B) (Black Forest Labs, 2024) on two commodity L40S GPUs, and by up to $2.0\times$, $1.9\times$, and $1.8\times$ on two H100 GPUs.

In this work, we build a first-order analytical model to decide when, absent any contention, layer prefetch can be hidden behind per-block computation. The model defines a single quantity—the *critical compute workload* $F^\star$—that characterizes the regime boundary: when the block-level FLOPs F_{block} exceed $F^\star$, full-layer prefetching is expected to incur little overhead; otherwise, part of the prefetch is exposed. Guided by this model, we design ChunkFlow, a chunk-granular offloading runtime that makes layer prefetch pauseable at chunk boundaries, so that collectives are no longer blocked behind a full-layer transfer, and that keeps a tunable fraction of chunks resident for workloads below $F^\star$ (Section 3).

On representative DiTs spanning text-to-image and text-to-video generation, on commodity L40S and H100 GPUs over PCIe, ChunkFlow consistently reduces denoising step time over state-of-the-art layerwise offloading for DiT by up to $1.32\times$, matches the no-offload baseline (assuming the GPU memory is sufficient) once the workload is large enough while retaining up to 49% peak-memory savings.

Contributions. (1) We develop a first-order analytical model that characterizes when layer prefetch can be fully hidden behind per-layer compute, applicable to layerwise offloading in general. (2) We identify a previously overlooked failure mode during layerwise offloading on PCIe-based distributed execution—contention between layer prefetch and collective communication on the shared PCIe path—and address this problem with a communication-aware chunked prefetching mechanism that yields PCIe to collectives at chunk boundaries. (3) For workloads whose per-layer compute is intrinsically too small to hide prefetch, we introduce a chunk-granular partial parameter residency mechanism that provides a tunable, fine-grained memory–latency trade-off.

2 BACKGROUND & MOTIVATION

2.1 DIT INFERENCE AND DISTRIBUTED EXECUTION

DiTs (Peebles & Xie, 2023) couple an iterative sampler with a Transformer-based denoiser invoked once per denoising step on a sequence of S tokens obtained after VAE downsampling and patchification.

To scale beyond a single GPU, distributed DiT inference engines parallelize with sequence parallelism (SP) (Fang et al., 2024b;a; Kong et al., 2024; Team Wan et al., 2025): every rank keeps a full replica of the weights, the tokens are partitioned across ranks, and ranks exchange them inside attention through all-to-all communication; Ulysses SP (Jacobs et al., 2023) is the common backbone and the setup we use throughout. SP is the natural choice because DiT inference is dominated by the sequence rather than by the weights: the weights typically fit on one GPU while the activations of a long sequence do not, so sharding the weights with tensor parallelism (Shoeybi et al., 2019) saves little memory yet adds a per-block all-reduce whose volume does not shrink with the parallel degree, whereas partitioning the sequence moves only a $1/N$ share per rank (Jacobs et al., 2023). Long-context LLM prefill follows the same rule, using tensor parallelism only as far as needed to fit the weights and context parallelism beyond that (Yang et al., 2024). The replicated per-rank weights are exactly what layerwise offloading removes. These collective communications are interleaved with computation within each block and can lie directly on the critical path of execution.

2.2 LAYERWISE OFFLOADING

Layerwise offloading reduces GPU memory footprint by keeping only a small working set of layers resident on the GPU, while the remaining weights are staged in host memory and transferred back on demand. Offloading is performed at layer granularity: during initialization all layers are placed in pinned host memory, and each denoising step then iterates over Transformer blocks. The processing of each layer l proceeds as follows.

- The compute stream waits for the asynchronous prefetch of the layer l to complete, ensuring its parameters are resident on the GPU.
- The compute stream then executes the forward pass of the layer l.
- During this execution, a dedicated CUDA copy stream asynchronously prefetches the next layer $(l{+}1)$.
- After the layer l completes, the runtime releases the parameters in l to reclaim GPU memory.

This reduces the resident working set to only two layers (one active, and one to prefetch). The standard implementation hinges on the assumption that the prefetch of layer $l{+}1$ finishes before layer l's compute does; otherwise, the computation stream must explicitly wait for the prefetch stream in the first step above, introducing stalls on the critical path.

2.3 PCIE CONTENTION UNDER DISTRIBUTED OFFLOADING

On PCIe-only systems—commonplace in commodity multi-GPU servers (e.g., 8×L40 PCIe nodes) that underpin budget-conscious DiT deployments— inter-GPU collectives and H2D prefetch traverse the same PCIe fabric and converge at each GPU's PCIe receive port, whereas on NVLink-equipped systems the collectives use a dedicated fabric and do not interfere with H2D transfers.

Performance characterization. We evaluate this effect using a 5B-parameter WanVideo diffusion transformer on a single node with two H100 GPUs connected via PCIe Gen5 $\times 16$ (peak $63\,\text{GB/s}$ per-GPU Rx shared between CPU-to-GPU H2D transfers and inter-GPU collective traffic), generating 81 frames with Ulysses sequence parallelism (Jacobs et al., 2023). Enabling layerwise offloading increases the average denoising step latency from $1.307\,\text{s}$ to $1.888\,\text{s}$—a $1.44\times$ slowdown. Figure 1b breaks down the inflated step latency: the NCCL all-to-all collective itself accounts for 24%, while the PCIe-contention stall—the all-to-all delayed behind in-flight prefetch traffic at the GPU's Rx—accounts for 21%, an overhead comparable to the collective's own time and confirming that the contention, rather than raw bandwidth or communication cost, dominates the gap.

3 DESIGN

3.1 ANALYTICAL MODEL FOR PREFETCH–COMPUTE OVERLAP

We first ask whether layerwise offloading adds latency at all on a PCIe-only node, in the idealized case in which the prefetch proceeds undisturbed by any contention. The answer turns on two dominant terms in one block execution window: the computation time of the current block and the host-to-device prefetch time of the next block.

Compute model. We model the per-block compute cost of one Transformer block on a single GPU. Our evaluation covers two block designs: the standard *DiT* block (e.g., in WanVideo), which serially interleaves self-attention, cross-attention to the text context, and an MLP; and the *MM-DiT* block (Esser et al., 2024) (e.g., in Flux, HunyuanVideo), which jointly processes image and text streams in a single attention call. We focus on the high-FLOP operators (attention projections, attention, and the MLP); lower-order components such as normalization, softmax, and elementwise operations are absorbed into an empirical calibration factor introduced below. The explicit per-term FLOP derivations for both block types are deferred to Appendix B.

Let B denote the batch size, S the global sequence length after patchification, d the hidden dimension, f the FFN expansion dimension, and L_{ctx} the text context length. We write the resulting per-block FLOP count as $F_{\text{block}}(B, S)$. Under fixed (d, f, L_{ctx}), F_{block} scales linearly in B and is quadratic in S only through self-attention while all other terms are linear in S.

Given the peak BF16 throughput P_{peak} of the GPU, we model the block compute time as

$$T_{\text{comp}}(B, S) = \frac{F_{\text{block}}(B, S)}{\eta_{\text{comp}} P_{\text{peak}}}, \tag{1}$$

where $\eta_{\text{comp}} \in (0, 1]$ is a compute calibration factor capturing the realized fraction of peak BF16 throughput on the target hardware (absorbing imperfect tensor-core utilization, memory-bound kernels, and non-GEMM operators); it is calibrated empirically per platform.

Prefetch model. Let B_{pref} denote the parameter bytes transferred before the next layer can execute (the entire layer for standard layerwise offloading) and BW_{h2d} the host-to-device bandwidth. We model the prefetch time as

$$T_{\text{pref}} = \frac{B_{\text{pref}}}{\eta_{\text{pref}} BW_{\text{h2d}}}, \tag{2}$$

where $\eta_{\text{pref}} \in (0, 1]$ is a prefetch calibration factor absorbing DMA startup cost, transfer granularity effects, and software runtime overheads; it is similarly calibrated empirically.

Critical compute workload. The first-order condition for fully hiding prefetch behind computation is defined as follows.

$$T_{\text{comp}} \geq T_{\text{pref}}. \tag{3}$$

This inequality states that the computation window of the current block must be at least as long as the service time of the next-layer prefetch. On a PCIe-only node computation is the entire hiding window, since prefetch and collectives share the PCIe receive port and no prefetch progresses while a collective is in flight; with NVLink the collectives leave the PCIe path and the window widens to $T_{\text{comp}} + T_{\text{comm}}$.

Substituting the compute and prefetch models above, the overlap condition can be written directly in terms of the block-level compute workload F_{block}.

$$F_{\text{block}}(B, S) \geq F^{\star}, \qquad F^{\star} := \eta_{\text{comp}} P_{\text{peak}} \cdot T_{\text{pref}} = \frac{\eta_{\text{comp}} P_{\text{peak}} B_{\text{pref}}}{\eta_{\text{pref}} BW_{\text{h2d}}}. \tag{4}$$

We call $F^{\star}$ the *critical compute workload*: when $F_{\text{block}} \geq F^{\star}$, the compute window is long enough to hide a full prefetch; when $F_{\text{block}} < F^{\star}$, a part of the transfer is exposed to the critical path. Equivalently, $I^{\star} := F^{\star}/B_{\text{pref}}$ is the turning point between the calibrated GPU compute roof and the host-to-device bandwidth roof in a roofline plot, connecting our analysis to the Hierarchical Roofline Model (Cao et al., 2025); see Appendix A for the graphical view, and Appendix C for the numerical instantiation of $F^{\star}$ on our evaluation platforms.

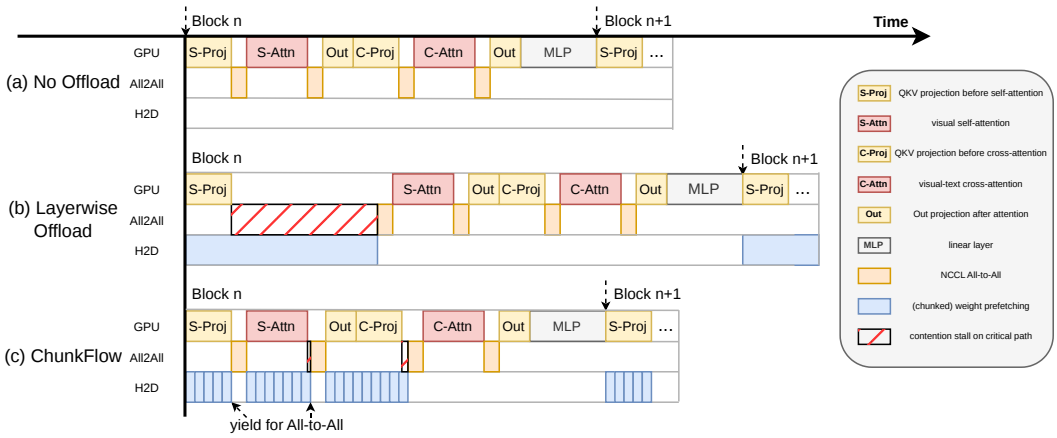


Figure 2: Block-level timing under (a) no offload, (b) whole-layer layerwise offload, and (c) our communication-aware chunked prefetch, illustrated on a DiT block with Ulysses sequence parallelism.

Since B_{pref} is approximately fixed for a given layer while F_{block} grows monotonically in both B and S, increasing either B or S pushes the workload across $F^\star$ and is generally favorable; conversely, partitioning computation across more GPUs (e.g., a higher sequence-parallelism degree) shrinks the per-GPU F_{block} and can pull the local workload back below $F^\star$.

3.2 Communication-Aware Chunked Prefetching

The condition of Section 3.1 assumes that the prefetch proceeds undisturbed. Even when it holds, the existing whole-layer implementation introduces a second cost on PCIe-only nodes. In the default whole-layer prefetch design, the prefetch of layer $l{+}1$ is issued near the beginning of layer l on a dedicated copy stream. However, the computation window before the first collective communication in a transformer block is typically short—for instance, in a self-attention layer under Ulysses sequence parallelism the collective is needed right after the QKV projection. As a result, next-layer prefetch and collective communication frequently overlap in time and contend for the same PCIe path on PCIe-only nodes. Once a large H2D transfer is in flight, the later-arriving collective is effectively delayed until the ongoing transfer finishes, so a part of the prefetch latency that is intended to be hidden by computation is instead exposed to the critical path, as illustrated in Figure 2(b).

To address this issue in PCIe-based environments, we design a communication-aware chunked prefetching mechanism. The key idea is to make layer prefetch *pauseable* and *resumable*, so that prefetch traffic can temporarily yield to the collective communications. Rather than issuing the entire next layer as one monolithic transfer, we partition the parameters of each layer into fixed-size chunks and prefetch them asynchronously chunk by chunk on the copy stream.

Since PCIe provides no mechanism to prioritize collective traffic over in-flight prefetch (Section 2.3) and CUDA streams provide no built-in preemption either, we orchestrate pause/resume in software via a lightweight control flag checked at chunk boundaries. When the compute stream issues an inter-GPU collective, the runtime sets a per-layer pause flag and records a CUDA event signaling the collective's completion. Before launching each chunk's H2D copy, the prefetch worker checks the flag; if set, it enqueues a `cudaStreamWaitEvent` on the recorded event so that the copy stream stalls on the GPU until the collective finishes, after which subsequent chunks resume automatically. This design preserves correctness—no in-flight DMA is ever aborted—and lets the collective interleave with chunk prefetching instead of serialized behind a full-layer transfer, as shown in Figure 2(c). Because the copy stream can only yield at chunk boundaries, a small residual stall—bounded by the service time of one in-flight chunk—remains, but much shorter than the full-layer blocking of whole-layer prefetch.

Chunking does not change the total I/O volume—it only turns a monolithic transfer into a sequence of shorter ones, leaving the chunk size C as a tunable parameter. Concretely, the residual stall above equals one chunk's service time $T_{\text{chunk}} = C/(\eta_{\text{pref}} BW_{\text{h2d}})$, and C governs a tradeoff between transfer efficiency and pause responsiveness: larger C amortizes DMA startup and runtime overhead and

improves H2D efficiency at the cost of a longer residual stall, while smaller C enables finer-grained interruption but underutilizes PCIe bandwidth.

3.3 Reducing Prefetch Time via Partial Parameter Residency

When $F_{\text{block}} < F^{\star}$, full offloading unavoidably exposes part of the prefetch latency on the critical path, creating an inherent tradeoff between GPU memory footprint and exposed prefetch volume.

To navigate this tradeoff, we introduce *partial parameter residency*, which, unlike the chunked prefetching mechanism in Section 3.2, is not specific to PCIe-based nodes and applies whenever the per-layer compute window is too short to hide a full prefetch. Instead of offloading all parameter chunks of a layer, we keep a subset of chunks resident in GPU memory and prefetch only the remaining ones. This mechanism is a natural extension to the chunked layout introduced in Section 3.2: since ChunkFlow already partitions each layer into fixed-size chunks, the residency can be controlled at chunk granularity rather than at the coarser module or whole-layer granularity, giving finer control of the memory–latency tradeoff.

This continuously trades prefetch volume for memory: at 0% residency the mechanism reduces to full offloading, at 100% residency to no offloading, and intermediate levels recover overlap when $F_{\text{block}} < F^{\star}$ at the cost of additional resident memory.

4 Evaluation

4.1 Experimental Setup

We evaluate three open-source diffusion transformers spanning two modalities: Wan2.2-TI2V-5B (Team Wan et al., 2025) (5B, text-to-video), HunyuanVideo (Kong et al., 2024) (13B, text-to-video), and FLUX.1-dev (Black Forest Labs, 2024) (12B, text-to-image). For each model we use the official default resolution, 10 denoising steps, and guidance configuration, and use a single fixed prompt across all runs.

Experiments run on commodity L40S and on H100 GPUs, both over PCIe: a node with two NVIDIA L40S GPUs (46 GiB, PCIe Gen4 $\times$16, no NVLink) and a node with two NVIDIA H100 PCIe GPUs (80 GB, PCIe Gen5 $\times$16) sharing the host PCIe root complex. Both use Ulysses sequence parallelism (Jacobs et al., 2023) with degree 2, FlashAttention (Dao et al., 2022; Dao, 2024; Shah et al., 2024) as the attention backend, BF16 weights, and a chunk size of 16 MB (Section 4.4). ChunkFlow is built on SGLang (Zheng et al., 2024) multimodal-generation runtime. We compare three configurations: (1) No Offload (all weights resident, assuming GPU memory is sufficient, which represents the shortest inference latency), (2) layerwise offload (SGLang's whole-layer prefetching, named Layerwise in the later discussion), and (3) ChunkFlow. Sections 4.3–4.5 are conducted on H100. Step time is the steady-state mean over denoising steps; peak memory is the peak GPU memory reserved by the allocator.

The overlap model and both mechanisms are not tied to this GPU count: varying it simply re-parameterizes the per-GPU F_{block} and B_{pref} in the overlap model (Section 3.1). The mechanisms are also architecturally portable beyond DiT, but the underlying overlap regime is less favorable for LLM inference (especially decode coupled with KV-cache offloading); we discuss this asymmetry in Appendix E.

4.2 Scaling over Frame Size and Batch Size

This evaluation has two goals: (i) evaluating how ChunkFlow performs across a range of per-GPU compute workloads, and (ii) validating the overlap model developed in Section 3.1. For the video models WanVideo and HunyuanVideo, we sweep the number of output frames, which changes the post-patchification sequence length S. For the image model Flux, we sweep the batch size B instead, which changes the block-level compute workload F_{block} through the linear-in-B dependence in the FLOP model. Table 1 reports step time and peak memory for every configuration on both GPUs.

Peak memory. Although ChunkFlow operates at chunk rather than whole-layer granularity, its peak memory is essentially the same as Layerwise and remains substantially below No Offload on both GPUs. At the third configuration point of each model—at the predicted $F^{\star}$—ChunkFlow reduces

Table 1: Scaling over frame size and batch size on both platforms. For each configuration: the average denoising step time in seconds of No Offload (No), Layerwise (LW) and ChunkFlow (CF), with ChunkFlow's speedup over Layerwise in parentheses; and peak GPU memory No Offload→ChunkFlow in GB. OOM: No Offload does not fit in the 46 GiB card.

2×H100 PCIe, Ulysses SP= 2					2×L40S PCIe, Ulysses SP= 2				
Cfg	No	LW	CF (vs LW)	Peak (GB)	Cfg	No	LW	CF (vs LW)	Peak (GB)
WanVideo (config = frames)									
41	0.636	1.274	1.022 (1.25×)	27.0→18.4	21	0.529	1.243	1.213 (1.02×)	29.2→16.0
81	1.307	1.888	1.494 (1.26×)	29.2→20.1	41	1.111	1.772	1.347 (1.32×)	29.2→17.3
121	2.205	2.710	2.379 (1.14×)	29.2→20.0	61	1.537	2.165	1.673 (1.29×)	29.2→18.7
161	3.292	3.728	3.333 (1.12×)	31.0→20.0	81	2.333	2.876	2.388 (1.20×)	29.2→17.3
Flux (config = batch size)									
4	0.816	1.447	1.278 (1.13×)	40.2→19.2	2	0.751	1.456	1.255 (1.16×)	36.8→14.5
8	1.623	2.080	1.863 (1.12×)	60.2→37.2	4	1.511	2.046	1.742 (1.17×)	39.8→29.6
12	2.462	2.778	2.576 (1.08×)	68.9→48.1	6	OOM	2.710	2.449 (1.11×)	—→38.8
16	3.258	3.532	3.345 (1.06×)	70.1→53.3	8	OOM	3.389	3.103 (1.09×)	—→42.3
HunyuanVideo (config = frames)									
9	0.792	1.514	1.184 (1.28×)	39.8→17.5	5	0.614	1.419	1.315 (1.08×)	37.8→17.5
21	1.342	1.980	1.549 (1.28×)	44.4→22.9	13	1.376	2.088	1.704 (1.23×)	39.3→18.2
33	2.868	3.335	2.958 (1.13×)	45.1→22.8	21	OOM	3.498	3.072 (1.14×)	—→18.6
45	3.871	4.278	3.865 (1.11×)	45.0→31.3	29	OOM	4.537	4.015 (1.13×)	—→19.1

peak memory by 31.4%, 30.1%, and 49.3% over No Offload on WanVideo, Flux, and HunyuanVideo on H100 while keeping denoising step time within a small margin of No Offload; on L40S, where the same weights come off a smaller card, the reductions range from 26% to 61%. The finer granularity therefore introduces no memory overhead. In a few cases, such as Flux at batch size 8 on H100, ChunkFlow in fact consumes less memory than Layerwise: allocating chunk buffers at a fixed size stabilizes the allocator's request pattern and reduces memory fragmentation compared to allocating whole-layer-sized blocks.

Step time. ChunkFlow achieves lower denoising step time than Layerwise on both GPUs across all three models and configurations, with speedups of up to $1.32\times$, $1.17\times$, and $1.23\times$ on WanVideo, Flux, and HunyuanVideo on L40S and up to $1.26\times$, $1.13\times$, and $1.28\times$ on H100. The gain does not shrink on the cheaper GPU: what Layerwise loses is a collective blocked behind a monolithic layer transfer, a cost set by the layer's parameter bytes and the H2D bandwidth rather than by compute throughput. As the frame size or batch size grows, ChunkFlow's step time steadily approaches No Offload on both GPUs, matching the prediction of Section 3.1: a larger F_{block} extends the compute window available to hide the prefetch overhead. On the 46 GiB L40S, HunyuanVideo at 21 and 29 frames and Flux at batch 6 and 8 do not fit without offloading; ChunkFlow runs them 1.09–$1.14\times$ faster than Layerwise, the only alternative.

Validating $F^\star$. We validate $F^\star$ derived in Section 3.1. The hardware constants P_{peak} and BW_{h2d} are read from the platform specification, and η_{comp} and η_{pref} are obtained from offline profiling on the same platform; the exact values are listed in Table 2 of Appendix C. Plugging each platform's constants into $F^\star$ yields thresholds that fall at the third configuration point of each model; on H100, for example, approximately 121 frames on WanVideo, batch size 12 on Flux, and 33 frames on HunyuanVideo. Starting from this point our step time is nearly indistinguishable from No Offload, which is consistent with the model. The measured gap continues to shrink slightly even beyond $F^\star$, indicating that the first-order model is accurate to within a small margin. The residual discrepancy is expected: the calibration factors η_{comp} and η_{pref} are picked empirically and not fitted per configuration, and ChunkFlow itself introduces a small amount of synchronization overhead (quantified in Section 4.3) that is not explicitly modeled. Raising the sequence-parallel degree moves the crossover the other way, as Section 3.1 anticipates: with four L40S GPUs at degree 4 every trend above is unchanged, but each rank computes a quarter of the sequence against the same B_{pref}, so the crossover shifts to larger frame counts (Appendix D).

4.3 Step Time Breakdown

To attribute ChunkFlow's speedup over Layerwise to specific cost components, we decompose ChunkFlow's per-step latency at each configuration swept on the H100 node into four parts: compute,

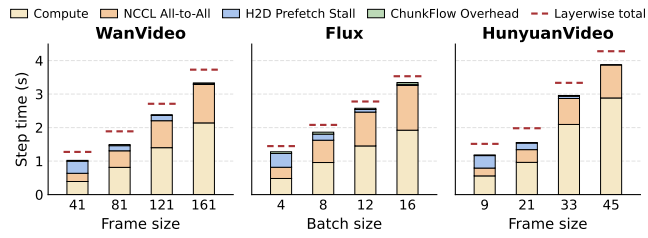


Figure 3: Per-step latency decomposition across frame and batch sizes with ChunkFlow.

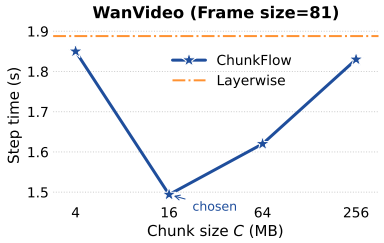


Figure 4: Step time vs. chunk size C on WanVideo at 81 frames (2×H100).

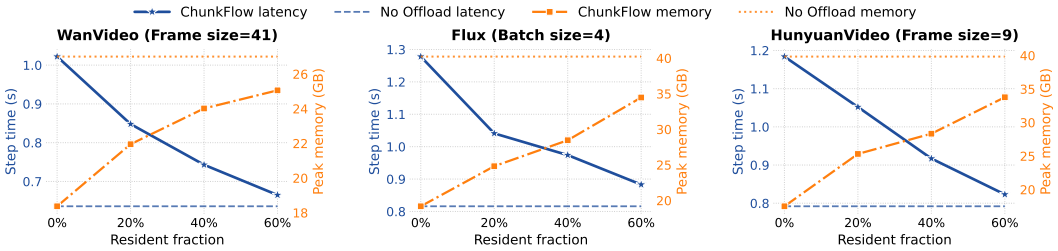


Figure 5: Denoising step time and GPU peak memory at the smallest H100 configuration of each model.

NCCL all-to-all, H2D prefetch stall, and ChunkFlow runtime overhead (Figure 3). Each bar's height equals ChunkFlow's step time. A dashed red marker above each bar shows Layerwise's step time with the same configuration; the gap between the bar top and marker is the PCIe contention stall introduced in Section 2.3 that Layerwise pays and that ChunkFlow eliminates.

ChunkFlow's runtime overhead is small. ChunkFlow's own overhead in Figure 3 comes from the pause/resume signalling at each collective and from the in-flight chunk that the copy stream finishes before yielding (Section 3.2). In practice both contributions remain small: across all three models and all swept configurations, ChunkFlow's overhead stays within about 2% of the total step time, far smaller than the PCIe contention stall it eliminates.

4.4 Chunk Size Selection

ChunkFlow's chunk size C controls the granularity at which the prefetch worker yields to NCCL collectives, and the right value balances two opposing effects. Figure 4 sweeps $C \in \{4, 16, 64, 256\}$ MB on WanVideo at 81 frames on the H100 node. At $C = 4$ MB, the per-chunk DMA startup cost is amortized over too few bytes, effective PCIe bandwidth drops, and step time degrades close to Layerwise. As C grows to 64 MB and then 256 MB, the chunks become progressively coarser and expose fewer yield points to collectives, so the prefetch behaves more and more like a whole-layer transfer and the schedule gradually falls back toward Layerwise. The chunk-tail stall incurred at each yield—bounded by $C/(\eta_{\text{pref}} BW_{\text{h2d}})$ as discussed in Section 3.2—also grows linearly with C, amplifying the fixed yielding cost on top of the lost overlap. $C = 16$ MB sits at the sweet spot between these two regimes and is the operating point we use in all other experiments.

4.5 Memory–Latency Tradeoff via Partial Residency

We evaluate the partial parameter residency mechanism introduced in Section 3.3. For each model we deliberately pick the smallest H100 configuration from the previous experiment—41 frames on WanVideo, batch size 4 on Flux, and 9 frames on HunyuanVideo—because at this scale the compute window is short enough such that a substantial portion of each layer's prefetch is exposed to the critical path under full offloading (i.e., no residency), leaving meaningful headroom for residency to recover. Starting from the fully-offloaded setting (ChunkFlow in Table 1, labelled as 0% residency here), we progressively keep a 20%, 40%, and 60% fraction of each layer's parameter chunks resident in GPU memory and prefetch only the remaining chunks.

Figure 5 illustrates the memory–latency tradeoff. As the resident fraction grows, the prefetch volume per layer shrinks, and denoising step time monotonically decreases toward No Offload. Correspondingly, peak memory rises monotonically from the fully-offloaded level toward No Offload, since more parameters must sit in GPU memory simultaneously. The two trends are strictly opposed, giving a continuous spectrum between minimum memory (full offload) and minimum latency (full residency). At only 60% residency, the step time of all three models comes within a very small margin of No Offload, indicating that the prefetch of the remaining 40% of parameters can be almost fully hidden by the compute window. At this point ChunkFlow still reduces peak memory by 7.2%, 14.3%, and 15.2% over No Offload on WanVideo, Flux, and HunyuanVideo, respectively. In other words, even in the low-compute regime where full offloading leaves visible overhead, ChunkFlow can recover near-zero step-time overhead while still delivering a meaningful reduction in peak memory.

5 Related Work

Memory-efficient inference via offloading. Existing efforts save GPU memory by offloading weights or activations to host or NVMe storage (Ren et al., 2021a;b; Lan et al., 2025; Ren et al., 2024; Xu et al., 2024; Kim et al., 2025). ZeRO-Infinity (Rajbhandari et al., 2021) pioneered hierarchical CPU/NVMe offloading. FlexGen (Sheng et al., 2023) formulates LLM inference as a tensor placement problem and searches for high-throughput offloading schedules under GPU memory budgets. vLLM (Kwon et al., 2023) introduces paged KV-cache management built on tensor parallelism (Shoeybi et al., 2019), and supports group-wise weight offloading at layer granularity. MoE-Lightning (Cao et al., 2025) targets MoE inference and develops a CPU–GPU–I/O pipelining schedule with paged weights to overlap expert transfer with computation. Production-quality inference systems such as SGLang (Zheng et al., 2024) and vLLM (Kwon et al., 2023) rely on naive whole-layer prefetching. They do not model the contention when distributed collectives and H2D transfers share a PCIe fabric.

Efficient diffusion inference. Diffusion model inference is dominated by repeated evaluations of a large denoiser, and a long line of work attacks this bottleneck along two complementary axes. The first axis reduces per-step compute or the number of steps. Faster ODE/SDE samplers (Song et al., 2021a; Lu et al., 2022; Karras et al., 2022) cut the number of denoising steps from hundreds to tens, and step-distillation methods such as consistency models (Song et al., 2023) push generation toward a few or even a single step. Architectural efficiency is improved through better DiT designs (Chen et al., 2024; Esser et al., 2024) and post-training optimizations including weight quantization (Li et al., 2023) and feature caching that reuses high-level activations across adjacent denoising steps (Ma et al., 2024b;a). The second axis is distributing computation across multiple GPUs. DistriFusion (Li et al., 2024) partitions the latent into spatial patches and reuses stale feature maps to amortize cross-patch communication; PipeFusion (Fang et al., 2024a) extends this with a patch-level pipeline parallel schedule; xDiT (Fang et al., 2024b) integrates sequence parallelism, patch-level pipeline parallelism, and CFG parallelism into a unified inference engine; these two lines of work reduce or distribute computation, while we study how to combine layerwise weight offloading with distributed execution.

6 Conclusions

We revisited layerwise offloading for DiT inference as a prefetch–communication co-scheduling problem. Guided by a first-order overlap model that defines a critical compute workload $F^\star$, we designed a chunk-granular offloading runtime with two mechanisms: communication-aware chunked prefetching, which eliminates PCIe contention by yielding the PCIe Rx to collective communication at chunk boundaries on PCIe-based nodes, and chunk-granular partial parameter residency, a general mechanism that trades GPU memory for prefetch volume whenever the compute window is too short, regardless of interconnect. Experiments on WanVideo, Flux, and HunyuanVideo on commodity L40S and H100 GPUs over PCIe with Ulysses sequence parallelism show that ChunkFlow consistently accelerates layerwise offloading by up to $1.32\times$, matches the no-offload baseline once the workload is large enough while retaining substantial peak-memory savings, and recovers near-zero step-time overhead for small workloads through partial residency.

AI USE STATEMENT

In this work, we used generative AI tools to implement parts of the ChunkFlow runtime and of the experiment harness, to provide feedback on the design of experiments, and to help interpret measurement results. The research ideas, the analytical model and its claims are the authors' own; we have not used generative AI tools to generate synthetic data or to formulate or prove mathematical claims, and the remaining tasks with required disclosure are not applicable to this work. Additionally, we used generative AI tools to draft and polish parts of the paper text, to search for and summarize related literature, and to format references. We have reviewed all AI-assisted work: all code was tested and all text was revised by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Section 4.1 lists the models, resolutions, frame counts and batch sizes, the number of denoising steps, the parallelism configuration, the attention backend and the hardware used in every experiment, together with how step time and peak memory are measured. The chunk-size and residency settings behind each result are stated in Sections 3.2, 4.4 and 4.5. The analytical model is fully specified in Section 3.1, with the per-block FLOP formulas in Appendix B and every calibration constant and resulting $F^\star$ prediction in Appendix C, so the predictions can be recomputed by hand. An anonymized snapshot of the runtime, a fork of SGLang implementing chunked prefetching, communication-aware yielding and partial residency, together with the experiment harness and trace-analysis scripts that produced every number in Section 4, is provided as supplementary material.

REFERENCES

Black Forest Labs. FLUX.1. `https://github.com/black-forest-labs/flux`, 2024.

Tim Brooks, Bill Peebles, Connor Holmes, Will DePue, Yufei Guo, Li Jing, David Schnurr, Joe Taylor, Troy Luhman, Eric Luhman, Clarence Ng, Ricky Wang, and Aditya Ramesh. Video generation models as world simulators. `https://openai.com/research/video-generation-models-as-world-simulators`, 2024.

Shiyi Cao, Shu Liu, Tyler Griggs, Peter Schafhalter, Xiaoxuan Liu, Ying Sheng, Joseph E. Gonzalez, Matei Zaharia, and Ion Stoica. MoE-Lightning: High-throughput MoE inference on memory-constrained GPUs. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2025.

Junsong Chen, Jincheng Yu, Chongjian Ge, Lewei Yao, Enze Xie, Zhongdao Wang, James Kwok, Ping Luo, Huchuan Lu, and Zhenguo Li. PixArt-α: Fast training of diffusion transformer for photorealistic text-to-image synthesis. In *International Conference on Learning Representations (ICLR)*, 2024.

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

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

Weishu Deng, Yujie Yang, Peiran Du, Lingfeng Xiang, Zhen Lin, Chen Zhong, Song Jiang, Hui Lu, and Jia Rao. HGCA: Hybrid GPU-CPU attention for long context LLM inference. *arXiv preprint arXiv:2507.03153*, 2025.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, and Robin Rombach. Scaling rectified flow transformers for high-resolution image synthesis. In *International Conference on Machine Learning (ICML)*, 2024.

Jiarui Fang, Jinzhe Pan, Aoyu Li, Xibo Sun, and Jiannan Wang. PipeFusion: Patch-level pipeline parallelism for diffusion transformers inference. *arXiv preprint arXiv:2405.14430*, 2024a.

Jiarui Fang, Jinzhe Pan, Xibo Sun, Aoyu Li, and Jiannan Wang. xDiT: an inference engine for diffusion transformers (DiTs) with massive parallelism. *arXiv preprint arXiv:2411.01738*, 2024b.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, Shuaiwen Leon Song, Samyam Rajbhandari, and Yuxiong He. DeepSpeed Ulysses: System optimizations for enabling training of extreme long sequence transformer models. *arXiv preprint arXiv:2309.14509*, 2023.

Youhe Jiang, Fangcheng Fu, Xiaozhe Yao, Guoliang He, Xupeng Miao, Ana Klimovic, Bin Cui, Binhang Yuan, and Eiko Yoneki. Demystifying cost-efficiency in LLM serving over heterogeneous GPUs. *arXiv preprint arXiv:2502.00722*, 2025.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Kihyun Kim, Jinwoo Kim, Hyunsun Chung, Myung-Hoon Cha, Hong-Yeon Kim, and Youngjae Kim. Cost-efficient LLM serving in the cloud: VM selection with KV cache offloading. *arXiv preprint arXiv:2504.11816*, 2025.

Weijie Kong, Qi Tian, Zijian Zhang, Rox Min, Zuozhuo Dai, et al. HunyuanVideo: A systematic framework for large video generative models. *arXiv preprint arXiv:2412.03603*, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Tingfeng Lan, Yusen Wu, Bin Ma, Zhaoyuan Su, Rui Yang, Tekin Bicer, Masahiro Tanaka, Olatunji Ruwase, Dong Li, and Yue Cheng. ZenFlow: Enabling stall-free offloading training via asynchronous updates. *arXiv preprint arXiv:2505.12242*, 2025.

Muyang Li, Tianle Cai, Jiaxin Cao, Qinsheng Zhang, Han Cai, Junjie Bai, Yangqing Jia, Ming-Yu Liu, Kai Li, and Song Han. DistriFusion: Distributed parallel inference for high-resolution diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Xiuyu Li, Yijiang Liu, Long Lian, Huanrui Yang, Zhen Dong, Daniel Kang, Shanghang Zhang, and Kurt Keutzer. Q-Diffusion: Quantizing diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. DPM-Solver: A fast ODE solver for diffusion probabilistic model sampling in around 10 steps. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Xinyin Ma, Gongfan Fang, Michael Bi Mi, and Xinchao Wang. Learning-to-cache: Accelerating diffusion transformer via layer caching. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024a.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. DeepCache: Accelerating diffusion models for free. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024b.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4195–4205, 2023.

Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. SDXL: Improving latent diffusion models for high-resolution image synthesis. In *International Conference on Learning Representations (ICLR)*, 2024.

Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. ZeRO-Infinity: Breaking the GPU memory wall for extreme scale deep learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2021.

Jie Ren, Jiaolin Luo, Kai Wu, Minjia Zhang, Hyeran Jeon, and Dong Li. Sentinel: Efficient Tensor Migration and Allocation on Heterogeneous Memory Systems for Deep Learning. In *International Symposium on High Performance Computer Architecture (HPCA)*, 2021a.

Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. ZeRO-Offload: Democratizing Billion-Scale Model Training. In *USENIX Annual Technical Conference*, 2021b.

Jie Ren, Dong Xu, Shuangyan Yang, Jiacheng Zhao, Zhicheng Li, Christian Navasca, Chenxi Wang, Harry Xu, and Dong Li. Enabling large dynamic neural network training with learning-based memory management. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2024.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. FlashAttention-3: Fast and accurate attention with asynchrony and low-precision. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. FlexGen: High-throughput generative inference of large language models with a single GPU. In *International Conference on Machine Learning (ICML)*, 2023.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations (ICLR)*, 2021a.

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations (ICLR)*, 2021b.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In *International Conference on Machine Learning (ICML)*, 2023.

Team Wan, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, et al. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025.

Didem Unat, Ilyas Turimbetov, Mohammed Kefah Taha Issa, Doğan Sağbili, Flavio Vella, Daniele De Sensi, and Ismayil Ismayilov. The landscape of GPU-centric communication. *arXiv preprint arXiv:2409.09874*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Dong Xu, Yuan Feng, Kwangsik Shin, Daewoo Kim, Hyeran Jeon, and Dong Li. Efficient Tensor Offloading for Large Deep-Learning Model Training based on Compute Express Link. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2024.

Amy Yang, Jingyi Yang, Aya Ibrahim, Xinfeng Xie, Bangsheng Tang, Grigory Sizov, Jeremy Reizenstein, Jongsoo Park, and Jianyu Huang. Context parallelism for scalable million-token inference. *arXiv preprint arXiv:2411.01783*, 2024.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

A Roofline View of the Overlap Threshold

The critical compute workload $F^\star$ derived in Section 3.1 admits a natural geometric interpretation. Define the per-layer *operational intensity* as the ratio of block-level FLOPs to the parameter bytes that must be prefetched before the block executes,

$$I_{\text{block}}(B,S) \;:=\; \frac{F_{\text{block}}(B,S)}{B_{\text{pref}}} \quad \text{(FLOPs/byte)}. \tag{5}$$

The overlap condition $F_{\text{block}} \geq F^\star$ is then equivalent to $I_{\text{block}} \geq I^\star$, where

$$I^\star \;=\; \frac{F^\star}{B_{\text{pref}}} \;=\; \frac{\eta_{\text{comp}} P_{\text{peak}}}{\eta_{\text{pref}} BW_{\text{h2d}}}. \tag{6}$$

Plotted as a roofline (Figure 6), $I^\star$ is the turning point between the calibrated GPU compute roof $\eta_{\text{comp}}P_{\text{peak}}$ and the host-to-device bandwidth roof of slope $\eta_{\text{pref}}BW_{\text{h2d}}$. Workloads whose operational intensity lies to the right of $I^\star$ admit full prefetch overlap; workloads to its left are H2D-bound and expose a portion of the transfer on the critical path.

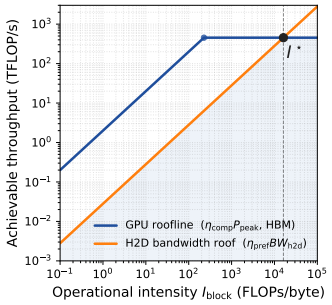


Figure 6: HRM with the GPU roofline and the host-to-device bandwidth roof.

Connection to the Hierarchical Roofline Model. The Hierarchical Roofline Model (HRM) (Cao et al., 2025) provides a general framework for analyzing computations that span multiple memory hierarchies, and has been used to guide policy search for offloaded MoE inference. Our analytical model can be viewed as an adaptation of HRM to the specific question of prefetch–compute overlap in layerwise offloading: by specializing the operational intensity to the per-block prefetch volume B_{pref} and folding empirical calibration factors into the two roofs, the same geometry yields a direct view of when the next layer's prefetch can be hidden behind the current layer's compute.

B Per-Block Compute Model: DiT and MM-DiT

This appendix instantiates the overlap model of Section 3.1 for the two block designs used by our evaluated models: the standard DiT block (WanVideo) and the MM-DiT block (Flux, HunyuanVideo). Both follow the same template— block-level FLOPs summed over the dominant operators, an empirical calibration factor η_{comp}, and the same critical compute workload threshold $F^\star$—but differ in the per-term FLOPs because MM-DiT processes the image and text token streams jointly in a single attention call rather than via a self-attention and a separate text cross-attention.

We reuse the notation of Section 3.1: B is the batch size, S the image/video token length after patchification, L_{ctx} the text context length, d the hidden dimension, and f the FFN expansion. FLOP counts below are written in global (across-GPU) form; under distributed execution each GPU performs only a fraction of them, and the per-GPU breakdown depends on the parallelism scheme. Ulysses sequence parallelism with degree T_{SP} partitions the head dimension, so every FLOP term below is divided by T_{SP} on a per-GPU basis. In all cases the structure of the overlap model is unchanged—only the per-GPU F_{block} and hence $F^\star$ are reparameterized—and the chunked prefetching and partial residency mechanisms apply unmodified. Residual losses (e.g., load imbalance, kernel-launch overhead) are absorbed into the calibration factor η_{comp}.

DiT block. A standard DiT block serially interleaves self-attention, cross-attention to the text context, and an MLP. The dominant FLOPs are

$$F_{\text{self-proj}} = 8BSd^2, \tag{7}$$
$$F_{\text{self-attn}} = 4BS^2d, \tag{8}$$
$$F_{\text{cross-proj}} = 4BSd^2 + 4BL_{\text{ctx}}d^2, \tag{9}$$
$$F_{\text{cross-attn}} = 4BSL_{\text{ctx}}d, \tag{10}$$
$$F_{\text{mlp}} = 4BSdf, \tag{11}$$

where "self-proj" covers the QKV and output projection of the self-attention, and "cross-proj" covers the video-side query, the text-side key/value, and the output projection of the cross-attention. Summing these dominant terms gives the per-block FLOP count

$$F^{\text{DiT}}_{\text{block}}(B,S) = F_{\text{self-proj}} + F_{\text{self-attn}} + F_{\text{cross-proj}} + F_{\text{cross-attn}} + F_{\text{mlp}}. \tag{12}$$

The MM-DiT denoiser is built from two block types (double-stream and single-stream) that we model separately and then combine via an averaged block size.

Double-stream block. A double-stream block (e.g., `transformer_blocks` in Flux, `double_blocks` in HunyuanVideo) maintains separate image-side and text-side weights for QKV projection, output projection, and the FFN, but the two streams are concatenated along the token dimension before the joint attention and split afterwards. The dominant FLOPs are

$$F_{\text{img-proj}} = 8BSd^2, \tag{13}$$
$$F_{\text{txt-proj}} = 8BL_{\text{ctx}}d^2, \tag{14}$$
$$F_{\text{joint-attn}} = 4B(S+L_{\text{ctx}})^2 d, \tag{15}$$
$$F_{\text{img-mlp}} = 4BSdf, \tag{16}$$
$$F_{\text{txt-mlp}} = 4BL_{\text{ctx}}df, \tag{17}$$

where the "proj" terms cover both QKV and output projection on each side ($4d^2$ each, hence the leading coefficient 8). Summing,

$$F_{\text{dbl}}(B,S) = F_{\text{img-proj}} + F_{\text{txt-proj}} + F_{\text{joint-attn}} + F_{\text{img-mlp}} + F_{\text{txt-mlp}}. \tag{18}$$

Compared with the DiT block, the quadratic-in-S contribution is replaced by a quadratic-in-$(S+L_{\text{ctx}})$ contribution from joint attention, and the cross-attention term disappears.

Single-stream block. A single-stream block (e.g., `single_transformer_blocks` in Flux, `single_blocks` in HunyuanVideo) concatenates the image and text tokens at block entry and processes them as one stream. It uses a single fused linear $\texttt{linear}_1$ that produces both QKV and the MLP up-projection, and a single fused linear $\texttt{linear}_2$ that absorbs the attention output and the MLP down-projection:

$$F_{\text{sng-lin1}} = 2B(S+L_{\text{ctx}})d(3d+f), \tag{19}$$
$$F_{\text{sng-attn}} = 4B(S+L_{\text{ctx}})^2 d, \tag{20}$$
$$F_{\text{sng-lin2}} = 2B(S+L_{\text{ctx}})(d+f)d, \tag{21}$$

$$F_{\text{sng}}(B,S) = F_{\text{sng-lin1}} + F_{\text{sng-attn}} + F_{\text{sng-lin2}}. \tag{22}$$

Block-averaged compute time. Let N_{d} and N_{s} be the number of double-stream and single-stream blocks in the model. We collapse the two block types into a single average block when applying the overlap condition:

$$\bar{F}_{\text{block}}(B,S) = \frac{N_{\text{d}}F_{\text{dbl}}(B,S) + N_{\text{s}}F_{\text{sng}}(B,S)}{N_{\text{d}}+N_{\text{s}}}, \qquad \bar{T}_{\text{comp}}(B,S) = \frac{\bar{F}_{\text{block}}(B,S)}{\eta_{\text{comp}}P_{\text{peak}}}. \tag{23}$$

Prefetch model. The prefetch model carries over without change. The per-block parameter volume B_{pref} used in $T_{\text{pref}} = B_{\text{pref}}/(\eta_{\text{pref}}BW_{\text{h2d}})$ is now an average across the two block types: a double-stream block contributes roughly $\beta(20d^2+4df)$ bytes (two attention projection sets, two MLPs, plus modulation), and a single-stream block contributes roughly $\beta(7d^2+2df)$ bytes (the fused $\texttt{linear}_1$ and $\texttt{linear}_2$ together with one modulation), where β is the parameter byte width.

Critical compute workload. With $\bar{F}_{\text{block}}$ and the average B_{pref} above, the overlap condition $\bar{F}_{\text{block}}(B,S) \geq F^\star$ and the critical compute workload $F^\star$ defined in Section 3.1 apply unchanged. Increasing B or S continues to grow $\bar{F}_{\text{block}}$ monotonically, with the same regime-boundary interpretation as the DiT case.

Timing under distributed offloading. Both Flux and HunyuanVideo place the sequence-parallel collectives inside the joint attention call of each block (as well as within the joint attention of the single-stream block, which operates on the concatenated image+text sequence). The block-level timing therefore mirrors Figure 2: in the layerwise-offload schedule, the in-flight H2D prefetch can block the in-block all-to-all at the GPU Rx and expose prefetch latency on the critical path; in ChunkFlow's chunked-prefetch schedule, the prefetch is paused at chunk boundaries and resumes after the collective completes. Because the chunked-prefetch and partial-residency mechanisms of Sections 3.2 and 3.3 operate on parameter chunks without reference to the block's internal structure, they apply to MM-DiT blocks with no structural changes.

C Numerical Instantiation and $F^\star$ Predictions

This appendix instantiates the overlap model of Section 3.1 on our evaluation platforms: it lists the hardware constants, calibration factors, and per-model parameters, then plugs them into the overlap condition to derive the predicted critical configuration for each model.

The H100 node has two H100 PCIe GPUs that share the host PCIe root complex, running BF16 inference with Ulysses sequence parallelism degree 2 and chunked prefetching. The hardware constants P_{peak} and BW_{h2d} are fixed by the platform and are read directly from the device specification: P_{peak} is the peak BF16 Tensor Core throughput of a single H100 PCIe in dense (no-sparsity) mode, and BW_{h2d} is the PCIe Gen5 $\times 16$ effective bandwidth (≈ 63 GB/s) halved because the two GPUs share the host PCIe root.

The calibration factors η_{comp} and η_{pref} are obtained from offline profiling on the same platform: η_{comp} from the BF16 FLOP/s achieved on a single DiT block on each of the three evaluated models, and η_{pref} from the H2D throughput measured under SGLang's layerwise prefetch. The per-model values of η_{comp} differ slightly across the three models; to keep the overlap model simple, we use the average. The four hardware and calibration constants are summarized in Table 2.

The L40S node was calibrated the same way before any experiment was run, with a kernel-level probe that needs no model weights: the GEMM shapes of one DiT layer give 254.6 TFLOP/s of achieved BF16 throughput ($\eta_{\text{comp}} \approx 0.70$ of the 362 TFLOP/s dense peak), and a pinned host-to-device copy of one layer's parameters gives 25.05 GB/s ($\eta_{\text{pref}} \approx 0.80$ of the 31.5 GB/s PCIe Gen4 $\times 16$ peak). The probe takes under a minute and repeats within about 1%. The peak must be measured rather than read off a datasheet: vendor BF16 figures are often quoted with structured sparsity, and taking such a halved number initially moved the predicted crossover by about 40%.

Table 2: Hardware constants and empirical calibration factors used to evaluate $F^\star$ in Section 4.2.

Symbol	H100 PCIe	L40S
P_{peak}	756 TFLOP/s	362 TFLOP/s
BW_{h2d}	31.5 GB/s	31.5 GB/s
η_{comp}	0.60	0.70
η_{pref}	0.89	0.80

The FLOP counts F_{block} in Section 3.1 and Appendix B are the global per-block FLOPs, i.e., summed over the entire batch and sequence before sequence-parallel partitioning; under SP degree 2 each GPU performs $F_{\text{block}}/2$ FLOPs. WanVideo uses the standard DiT block of Section 3.1 with 30 blocks, while Flux and HunyuanVideo use the MM-DiT block of the same appendix with $(N_d, N_s) = (19, 38)$ and $(20, 40)$ double-/single-stream blocks respectively. The other formula parameters and the effective per-block prefetch volume B_{pref} are summarized in Table 3: B is the batch size (swept for Flux; fixed at 1 for the two video models), d the hidden dimension, f the FFN intermediate dimension, L_{ctx} the text context length, and S the image-token sequence length. The S formula in each row reflects the model's input resolution—WanVideo at 704×1280, Flux at 1024×1024, and HunyuanVideo at 720×1280—together with its VAE compression and patchification. B_{pref} is obtained from offline profiling on each model.

Plugged into the per-GPU overlap condition $T_{\text{comp}} = T_{\text{pref}}$ together with the constants in Table 2, the values in Table 3 yield predicted critical configurations of $n^\star \approx 119.2$ frames for WanVideo, $b^\star \approx 11.5$ for Flux, and $n^\star \approx 34.5$ frames for HunyuanVideo. Rounding each to the nearest valid

Table 3: Per-model formula parameters and effective per-block prefetch volume B_{pref}.

Model	B	d	f	L_{ctx}	Sequence length S	B_{pref}
WanVideo	1	3072	14336	512	$220(n{+}3)$	520 MB
Flux	b	3072	12288	512	4096	465 MB
HunyuanVideo	1	3072	12288	161	$900(n{+}3)$	675 MB

swept configuration in Section 4.2 gives 121 frames, batch 12, and 33 frames respectively, which are the third configuration we chose for each model.

D Scaling the Sequence-Parallel Degree

Section 3.1 predicts that partitioning a fixed workload across more GPUs pulls the per-GPU workload back toward $F^\star$: under Ulysses each rank computes $1/N$ of the sequence but still prefetches the full layer, so F_{block} divides by N while B_{pref} does not. Table 4 repeats the WanVideo sweep of Table 1 on a node with four L40S GPUs (same card, PCIe Gen4 $\times$16, no NVLink) at sequence-parallel degree 4, with the frame counts of the H100 sweep.

Table 4: WanVideo on 4$\times$L40S PCIe with Ulysses SP$=4$: average denoising step time in seconds of No Offload (No), Layerwise (LW) and ChunkFlow (CF), with ChunkFlow's speedup over Layerwise in parentheses, and peak GPU memory No Offload$\rightarrow$ChunkFlow in GB.

Frames	No	LW	CF (vs LW)	Peak (GB)
41	0.673	1.385	1.261 (1.10$\times$)	21.0$\rightarrow$12.1 (-42%)
81	1.425	2.089	1.574 (1.33$\times$)	29.3$\rightarrow$12.1 (-59%)
121	2.044	2.681	2.107 (1.27$\times$)	39.6$\rightarrow$12.1 (-69%)
161	3.097	3.659	3.155 (1.16$\times$)	40.5$\rightarrow$12.1 (-70%)

Every trend of Table 1 carries over: ChunkFlow is 1.10–1.33$\times$ faster than Layerwise at every frame count, its step time approaches No Offload as the frame count grows, and it cuts peak memory by 42–70%. What moves is the crossover. At a fixed frame count SP$=4$ sits deeper in the exposed regime than SP$=2$—$+87\%$ over No Offload at 41 frames against $+21\%$, $+10\%$ against $+2\%$ at 81—and ChunkFlow reaches a $+2\%$ overhead only at 161 frames instead of 81, i.e., the critical point shifts to a larger workload, as the model predicts.

E Applicability to LLM Inference

The design of ChunkFlow—the overlap model (Section 3.1), communication-aware chunked prefetching (Section 3.2), and partial parameter residency (Section 3.3)—is not architecturally tied to diffusion transformers: each component depends only on per-GPU compute throughput, host-to-device bandwidth, and the layered structure of the forward pass, all of which are shared by any transformer-based inference workload. We focus on DiT inference rather than LLM inference because the underlying overlap regime is far more favorable for DiT, as outlined below.

Long per-step compute window favors DiT. DiT inference processes the entire post-patchification token sequence in every denoising step; the per-block compute cost F_{block} scales with the global sequence length S (Section 3.1). For high-resolution image and video models, S routinely reaches 10^5–10^6 tokens, pushing F_{block} above the critical compute workload $F^\star$ in the configurations we evaluate. LLM inference operates at a much smaller scale: prefill sequences are typically only on the order of 10^3–10^4 tokens (e.g., the default per-batch token budget of vLLM (Kwon et al., 2023) is 2048), and decode produces a single new token per request per step, so the effective per-step sequence length equals the batch size, which is generally smaller than the prefill length. The corresponding F_{block} is therefore substantially smaller than DiT's, placing typical LLM workloads well below $F^\star$, with decode being the most extreme case.

Memory-bound regimes (decode, MoE) make prefetch even harder. Two important LLM regimes are dominated by memory traffic rather than tensor-core arithmetic: dense-model decode and MoE inference, where the per-layer compute is bottlenecked by HBM weight (and, in decode, KV-cache) reads rather than peak FLOP/s. Since HBM bandwidth exceeds PCIe bandwidth by

roughly an order of magnitude (e.g., $\sim 2\,\text{TB/s}$ vs $63\,\text{GB/s}$ on H100 PCIe), an H2D prefetch of the same weights over PCIe is substantially slower than the HBM-bound compute it must hide behind, so $T_{\text{pref}} \gg T_{\text{comp}}$ once layerwise offloading is enabled. Recovering overlap via partial residency (Section 3.3) would require a resident fraction so high that it approaches the no-offload baseline, eroding the memory-saving rationale for offloading in the first place.

KV-cache offloading further compresses the prefetch bandwidth budget. In memory-constrained LLM deployments, KV-cache offloading is commonly enabled alongside weight offloading: the KV cache is staged in host memory and streamed to the GPU as needed. This adds a second stream of H2D traffic in both prefill and decode, competing with model-weight prefetch on the same PCIe path and effectively shrinking the prefetch calibration factor η_{pref} in the overlap model. The result is to push $F^\star$ even higher, exacerbating the prefetch-bound regime described above.

Summary. All three components of ChunkFlow—the overlap model, chunk-granular pause/resume, and partial residency—are general and could be applied to LLM inference with no architectural change. Their practical effectiveness, however, is governed by the overlap model itself: DiT inference sits comfortably in the regime where prefetch can be hidden, whereas LLM inference (decode in particular, especially with concurrent KV-cache offloading) sits deep in the prefetch-bound regime where neither chunked prefetching nor moderate residency is sufficient. A satisfactory treatment of LLM inference under offloading would likely require co-designing model-weight and KV-cache offloading scheduling, which is beyond the scope of this work.