Making DeepSeek-V4-Flash Practical on a Four-GPU Ampere Workstation: Tensor Parallelism, NUMA-Aware Hybrid MXFP4 Execution, Exact DSpark, and an Acceptance-Guided Fallback

Carlos Alvarado
School of Electrical Engineering and Computer Science
University of North Dakota
Grand Forks, ND 58202 USA
c.alvaradomartinez@und.edu

***Abstract*—This paper documents the engineering required to serve the official DeepSeek-V4-Flash-0731 checkpoint on a dual-socket Lenovo workstation with four 16 GiB NVIDIA RTX A4000 GPUs. The local checkpoint occupies 156 GiB, while the workstation provides only 62.4 GiB of aggregate VRAM and no native FP4 tensor-core path. An unmodified single-rank FREETOKEN launch failed before graph capture because its minimum cache plan exceeded the available device budget. Feasibility was therefore a systems problem, not a launch-flag problem.**

The submitted upstream work is organized as three stacked pull requests. The first adds tensor-parallel ownership and rank-local checkpoint loading. The second implements target-distribution-preserving DSpark decoding, including speculative state commit and release. The third adds measured, hardware-aware verification widths, device-resident rejection sampling, in-place collectives, NUMA-aware placement reporting, route-aware hybrid expert fetching, long-prompt hardening, and a request-local acceptance fallback. At the validated serving point, 146.62 GiB of expert storage resides in host memory and 55.24 GiB of live allocations reside across the GPUs, a counted placement of 72.6 percent CPU and 27.4 percent GPU. The measured hybrid split assigns 28 percent of expert misses to PCIe fetch and the remainder to CPU execution.

Structured code generation sustains approximately 28–30 $\mathrm{tok\,s^{-1}}$, compared with an approximately 18 $\mathrm{tok\,s^{-1}}$ target-only baseline. Doubling the prefill chunk from 1,024 to 2,048 tokens raises full-chunk throughput from approximately 322 to 626 $\mathrm{tok\,s^{-1}}$ and reduces a fixed long-prompt wall time by 40.6 percent. For a matched low-acceptance creative workload, the request-local fallback improves end-to-end generation from approximately 15.7 to 17.18 $\mathrm{tok\,s^{-1}}$, with target-only windows of 17.86–19.59 $\mathrm{tok\,s^{-1}}$. These measurements establish a workstation case study, not a universal hardware or workload claim. The engineering contribution is the measured composition of exact speculative decoding, block-scaled FP4 execution, tensor parallelism, CPU–GPU co-execution, and NUMA placement under a strict memory budget.

***Index Terms*—DeepSeek-V4-Flash, DSpark, MXFP4, mixture of experts, tensor parallelism, NUMA, speculative decoding, heterogeneous inference.**

I. INTRODUCTION

DeepSeek-V4-Flash combines a large sparse expert population with a comparatively small active parameter set, hybrid long-context attention, manifold-constrained hyper-connections, and an attached DSpark drafter [1], [2]. These properties reduce arithmetic per generated token, but they do not make the checkpoint small. The released model also places unusually strict requirements on an inference engine: routed experts use block-scaled FP4 weights, non-expert weights use higher precision, target attention maintains several forms of position-addressed state, and speculative verification must preserve that state through partial acceptance and rejection.

The workstation in this study is far below the datacenter configuration used in the official model card. The card demonstrates DSpark on a four-GPU GB300 node and recommends specialized MXFP4 backends [3]. Our machine instead has four PCIe-attached RTX A4000 cards from the Ampere generation. Each card has 16 GiB of VRAM and 448 $\mathrm{GB\,s^{-1}}$ of device memory bandwidth [4]. Ampere predates native FP4 tensor-core execution. NVIDIA introduced hardware FP4 support with Blackwell, and its NVFP4 format is not identical to the checkpoint's MX-style format [5], [6].

The phrase "the model would not run otherwise" needs a precise scope. It does not mean that no lossy conversion, alternative engine, larger machine, or reduced model could ever serve DeepSeek-V4-Flash. It means that the official 156 GiB local checkpoint could not be served by the then-current single-rank FREETOKEN path on this exact 62.4 GiB-VRAM workstation. The first observed launch exhausted its cache budget before CUDA graph capture. The submitted work changes the ownership, execution, and scheduling of the official weights so the checkpoint can load and generate without converting the routed expert representation to a lower-fidelity format.

The contributions are as follows.

1) We define tensor-parallel ownership for every DeepSeek-V4 component, including quantization scales, and load only rank-owned slices. This removes expert-bank replication and avoids retaining parent tensor storage through views.
2) We implement the checkpoint's three-layer, five-proposal DSpark drafter with exact greedy and probabilistic acceptance. The probabilistic path preserves the target distribution after temperature, top-p, and top-k transforms.
3) We integrate DSpark verification with FREETOKEN's bandwidth-adaptive hybrid MoE backend. Expert misses are divided between overlapped CPU computation and PCIe transfer according to measured bandwidth, while repeated speculative routes receive fetch priority.
4) We make placement NUMA-aware before expert-bank allocation. Each rank's CPU workers, first-touch memory, and PCIe-attached GPU remain on the same socket whenever the topology permits.
5) We add measured adaptive verification and a request-local fallback for persistently low DSpark acceptance. The controller observes target acceptance, does not classify prompt text, runs target-only decoding for a bounded interval, and then probes speculation again.
6) We report end-to-end and steady-state measurements with their caveats, including a negative CUDA-graph result and a capture-memory failure. The report separates confirmed measurements from projections and branch names.

This is an implementation and evaluation paper. It does not claim a new proof of speculative sampling, a new FP4 representation, or a new NUMA policy. The original contribution lies in making these mechanisms agree at the boundaries where workstation inference commonly fails: tensor ownership, quantization metadata, speculative state, rank-local memory, PCIe service, graph replay, and request lifecycle.

II. BACKGROUND AND TERMINOLOGY

A. Sparse model execution

For a token representation x, a routed mixture-of-experts layer can be written as

$$y(x) = E_s(x) + \sum_{e \in \mathcal{T}_k(x)} g_e(x) E_e(x), \tag{1}$$

where E_s is a shared expert, $\mathcal{T}_k(x)$ is the router-selected set of k experts, and g_e is the routing weight. DeepSeek's fine-grained routed experts and shared experts follow the specialization approach introduced in DeepSeekMoE [7]. DeepSeek-V4-Flash activates only a fraction of its total parameters per token, but every expert must remain available for a possible route. Storage capacity is therefore governed by total parameters; per-token latency is governed by the active routes, their residency, and the cost of moving or computing misses.

CPU or disk offload has a substantial history in large-model inference [8]. The relevant distinction in FREETOKEN is that expert misses need not follow one global policy. A cache hit runs on the GPU. Among misses, some expert rows are transferred to the GPU while the remainder execute directly against host-resident banks. These two paths are launched concurrently [9].

B. MXFP4 and NVFP4 are different formats

The DeepSeek-V4 routed experts in this checkpoint use packed E2M1 values with an E8M0 scale shared by each block of 32 values. This matches the defining geometry of OCP MXFP4 [10], [11]. Let $c_{b,i} \in \{0, \dots, 15\}$ be the four-bit code at position i of block b, let e_b be the unsigned E8M0 exponent code, and let $v(c)$ be the E2M1 lookup value. The decoded weight is

$$\widehat{w}_{b,i} = v(c_{b,i}) 2^{e_b - 127}, \qquad i \in \{0, \dots, 31\}. \tag{2}$$

The E2M1 magnitude set used by the implementation is

$$\{0, 0.5, 1, 1.5, 2, 3, 4, 6\}, \tag{3}$$

with one sign bit. Ignoring tensor alignment, a block costs

$$4 + \frac{8}{32} = 4.25 \quad \text{bits per weight.} \tag{4}$$

NVFP4 also uses E2M1 values, but it uses an E4M3 scale per 16 values plus a second-level tensor scale. Its nominal storage is 4.5 bits per weight before the tensor scale [6]. Calling the checkpoint format NVFP4 would be technically incorrect and would conceal why a Blackwell-specific kernel cannot simply be selected on Ampere. The FREETOKEN implementation names this expert layout `ds_fp4`; in this paper we call it MXFP4 when discussing the numeric format and `ds_fp4` when discussing the backend identifier.

Because the A4000 lacks native FP4 matrix instructions [5], [6], the portable GPU kernels read packed nibbles and E8M0 scales directly, decode them within the reduction, and accumulate in FP32. They never materialize a complete BF16 expert copy. The CPU path likewise reads the packed row-major banks in place using AVX-512. This is a software execution path for an MXFP4 checkpoint, not native FP4 arithmetic.

C. Exact speculative decoding

Speculative decoding uses a cheaper drafter distribution q to propose tokens and a target distribution p to verify several positions in one target pass. Its central benefit is exactness: the accepted output has the same distribution as target-only sampling [12], [13]. For a proposal x_i, the acceptance probability is

$$a_i = \min\left(1, \frac{p_i(x_i)}{q_i(x_i)}\right). \tag{5}$$

At the first rejection, the replacement token is drawn from

$$r_i(x) = \frac{\max\{p_i(x) - q_i(x), 0\}}{\sum_z \max\{p_i(z) - q_i(z), 0\}}. \tag{6}$$

If the entire proposal block survives, the target supplies one additional bonus token. Greedy decoding is the point-mass special case: accept a contiguous prefix while drafter tokens equal target argmax tokens, then emit the first target disagreement.

DSpark improves proposal quality by combining a parallel draft backbone with a lightweight sequential Markov head. It also predicts position confidence and uses engine-specific cost information to avoid verifying unprofitable suffixes [2]. The checkpoint used here fixes the maximum proposal count at $\gamma = 5$.

III. Workstation and feasibility boundary

Table I records the hardware as inspected on 25 August 2026. The service was inactive during inspection, but another GPU workload occupied most device memory. Serving measurements cited later come from clean FreeToken test runs documented on 23 August 2026 [14].

The simple capacity inequality is already decisive:

$$M_{\text{checkpoint}} = 156\,\text{GiB} > M_{\text{VRAM,total}} = 62.40\,\text{GiB}. \quad (7)$$

The ratio is 2.5. Further, aggregate VRAM is useful only after a runtime assigns disjoint ownership to the four GPUs. A single-rank process sees one 16 GiB device. The original launch selected offload but failed its minimum cache plan, which required 9.14 GB from a remaining budget of only 4.09 GB. Merely setting tensor parallelism was not enough, because the then-existing DeepSeek-V4 model shapes and loading path assumed one rank and could retain or replicate the largest tensors.

The successful final placement is summarized in Table II [14], [15]. "Counted placement" includes host expert banks and GPU allocations reported after graph capture. It is not the checkpoint file size and should not be interpreted as one-to-one disk-to-memory expansion.

Rank 0 used 14.34 GiB; ranks 1–3 each used 13.63 GiB. Each rank owned 36.66 GiB of host expert banks, nine CPU MoE workers, and one coordinator core. The four GPUs held 88.5 percent of their usable VRAM in aggregate. The host expert banks consumed 29.2 percent of available system RAM.

IV. The upstream pull-request series

The work was submitted as three stacked pull requests to the FlashML-org/FreeToken upstream repository. Review order follows the labels 1/3, 2/3, and 3/3, although GitHub numbered the second layer #69 and the foundation #70. Table III uses GitHub's cumulative diff statistics as inspected on 25 August 2026. A stacked cumulative diff includes its prerequisite commits, so the rows must not be added together.

A. PR 1: tensor-parallel feasibility

PR #70 establishes memory ownership and model algebra before introducing speculation [16]. It contains six commits at the inspected head:

- tensor-parallel DeepSeek-V4 model and checkpoint loading;
- corrected GPU mapping after upstream rebase;
- tests for the numeric contents of expert-bank shards;
- rejection of a TP=1 fast-weight layout when launched under TP> 1;
- pageable CPU-layer banks for hosts with limited pinning quotas; and
- bounded parallel expert prefetch to control host-memory pressure.

The review boundary is deliberately independent of DSpark. Its job is to make one target-model forward mathematically complete and memory-feasible across four ranks.

B. PR 2: exact DSpark and operational state

PR #69 adds two commits above the TP foundation [17]. The first loads the checkpoint drafter, captures target features, constructs draft context KV, proposes a block, verifies it, and implements exact greedy and probabilistic acceptance. The second fixes the issues that appear only under full hybrid serving: auxiliary-row addressing, absolute rotary positions, target-layer taps, NUMA affinity, speculative page release, drafter residency, and service lifecycle.

The distinction between an algorithmic prototype and an operational path is important. Verification changes more state than the request's visible token list. It writes target KV, sliding-window draft KV, recurrent compressor state, sparse-indexer state, allocation frontiers, and detokenizer output. Each owner needs an explicit commit rule after the accepted prefix is known.

C. PR 3: hardware-aware execution

PR #71 adds ten commits above the exact path [15]. They cover adaptive TP verification, GPU-resident probabilistic sampling, exact multi-token detokenization, fused hyper-connection normalization, long-prefill bounds, in-place NCCL, final placement reporting, route-aware expert fetch, and the measured-acceptance fallback. This PR is where the correct implementation becomes a measured workstation runtime.

V. Tensor-parallel model ownership

Tensor parallelism partitions individual operators rather than assigning whole layers to devices. The general approach follows intra-layer model parallelism as used in Megatron-LM [18]. DeepSeek-V4 nevertheless requires an ownership table specific to its attention, expert, and quantization structures.

For a row-parallel linear map, partition the input and weight along the input dimension:

$$x = [x^{(0)}, \ldots, x^{(P-1)}], \qquad W = [W^{(0)}; \ldots; W^{(P-1)}]. \quad (8)$$

Each rank forms a partial output

$$y^{(r)} = x^{(r)} W^{(r)}, \quad (9)$$

and the complete value is

$$y = \sum_{r=0}^{P-1} y^{(r)} = \text{AllReduce}\left(y^{(r)}\right). \quad (10)$$

For column parallelism, each rank owns an output slice and concatenation completes the tensor. Vocabulary embeddings use masked local lookup followed by one all-reduce. The language-model head produces rank-local vocabulary logits and uses rank-ordered all-gather.

Table IV gives the DSV4-specific rules.

TABLE I
LENOVO WORKSTATION USED FOR THE CASE STUDY.

Component	Configuration
CPU	2 × Intel Xeon Gold 6148, 20 physical cores per socket, 2 threads per core
NUMA	2 nodes; Linux distance 10 local and 21 remote
Host memory	502.47 GiB visible to the operating system
GPUs	4 × NVIDIA RTX A4000, 16 GiB class, 62.40 GiB total usable VRAM
GPU bandwidth	448 GB s^{-1} per card, 1,792 GB s^{-1} arithmetic aggregate, not one shared memory fabric
Interconnect	PCIe-only; GPUs 0 and 1 are local to NUMA node 0, GPUs 2 and 3 to node 1
Checkpoint	DeepSeek-V4-Flash-0731, 156 GiB on local storage
Runtime	FREETOKEN, TP=4, one active request, OpenAI-compatible serving

TABLE II
MEASURED FINAL SERVING PLACEMENT AFTER GRAPH CAPTURE.

Location	Ranks	Expert banks	GPU allocation	Assigned CPU
NUMA node 0	0, 1	73.31 GiB	27.97 GiB	20 cores
NUMA node 1	2, 3	73.31 GiB	27.26 GiB	20 cores
Total	0–3	146.62 GiB	55.24 GiB	40 cores
Share of counted placement		72.6%	27.4%	

TABLE III
CURRENT UPSTREAM REVIEW STACK. ALL THREE PRS WERE OPEN AT INSPECTION TIME.

Order	Pull request	Head	Commits	Files	Additions	Deletions
1/3	PR #70: TP runtime	`c066b38`	6	18	903	121
2/3	PR #69: exact DSpark	`2cf938b`	8	53	5,360	193
3/3	PR #71: adaptive verification	`4800af0`	18	73	7,465	335

TABLE IV
TENSOR OWNERSHIP IN THE TP=4 RUNTIME.

Component	Ownership	Completion and rationale
Token embedding	Vocabulary rows	Masked lookup and one all-reduce.
LM head	Vocabulary rows	Rank-ordered all-gather produces full logits.
MLA query projection	Query-head columns	Every rank owns a deterministic head block.
MLA output projection A	Whole output groups	Partial groups are rejected during configuration.
MLA output projection B	Input rows	One all-reduce completes the attention output.
Shared expert	Intermediate dimension	Rank-local partial, added to routed partial before reduction.
Routed experts and scales	Same intermediate slice	Weight and E8M0 scale grids are co-sliced; one MoE all-reduce completes both branches.
Latent KV and paged KV	Replicated	Every local attention head consumes the same latent KV.
Router, compressors, indexer	Replicated	All ranks must select the same experts and sparse blocks.
Hyper-connection control	Replicated	Small tensors whose mixing decisions must agree across ranks.
DSpark Markov and confidence heads	Replicated	Sharding would add a collective at every proposal position.

The shared and routed expert branches are reduced together. If rank r forms $s^{(r)}$ from its shared-expert slice and $m^{(r)}$ from its routed-expert slice, the layer performs

$$y = \mathrm{AllReduce}\left(s^{(r)} + m^{(r)}\right), \tag{11}$$

not two independent reductions. This preserves Equation 1 while removing one collective per layer.

Quantized partitioning must preserve blocks. A requested local dimension that cuts an FP4 32-value block or an FP8 128-value scale grid is rejected before model construction. The loader slices both packed weights and their scale grids on the same axis. It then clones the slice into independent storage. A contiguous `narrow()` can still be a view; retaining the view retained the full parent allocation and was measured to cost 5.1 GiB per GPU at TP=4. This detail alone is large relative to a 16 GiB device.

VI. NUMA-aware hybrid expert execution

A. Why first touch matters

Linux memory policy and page placement are NUMA-sensitive [19], [20]. Anonymous pages are normally allocated on the node of the thread that first writes them. Loading a 36.66 GiB rank-local expert bank before binding the rank can scatter its pages across sockets. A CPU expert worker then reads remote memory, while PCIe DMA may also cross the socket interconnect to reach the rank's GPU.

The runtime now performs the following sequence before large allocation:

1) Read the allowed CPU mask and NUMA CPU lists from Linux sysfs.
2) Resolve each CUDA device's PCI bus identifier and its local NUMA node.
3) Map ranks to their GPU-local nodes when that information is available.
4) Bind rank CPU affinity and record the placement before affinity hides the original multi-node topology.
5) Allocate and first-touch expert banks under that binding.
6) Split the node's physical cores among ranks and reserve one coordinator per rank.

The resulting mapping was ranks $(0, 1)$ to node 0 and ranks $(2, 3)$ to node 1. Every rank received nine AVX-512 expert workers and one coordinator. The final placement reporter gathers these facts from every rank after CUDA graph capture, when memory measurements reflect the serving state rather than an intermediate loader state.

B. Bandwidth-matched split

Let B be the bytes associated with the expert misses in one step, let q be the fraction fetched over PCIe, and let β_g and β_c be achieved PCIe gather and CPU expert-compute bandwidth while both paths run concurrently. The two service times are

$$T_g(q) = \frac{qB}{\beta_g}, \qquad T_c(q) = \frac{(1-q)B}{\beta_c}. \tag{12}$$

With ideal overlap, the miss service time is

$$T_{\text{hybrid}}(q) = \max\{T_g(q), T_c(q)\}. \tag{13}$$

Balancing the two paths gives

$$q^* = \frac{\beta_g}{\beta_g + \beta_c}. \tag{14}$$

The workstation measured $\beta_c = 25.37\ \text{GB}\,\text{s}^{-1}$ and $\beta_g = 9.85\ \text{GB}\,\text{s}^{-1}$ under overlap. Equation 14 yields

$$q^* = \frac{9.85}{9.85 + 25.37} = 0.2797, \tag{15}$$

which is the deployed 28 percent fetch fraction [14]. Standalone host-to-device bandwidth was $12.3\ \text{GB}\,\text{s}^{-1}$; using that isolated figure would ignore the contention regime that hybrid execution actually experiences.

The current 28 percent is a machine-calibrated starting point, not a physical constant. It was derived from a single-rank bandwidth profile, whereas TP=4 serving runs four ranks and places two on each socket. A proper follow-up must sweep nearby fractions under the complete TP and NUMA workload.

C. MXFP4 on CPU and GPU

For a cache hit or fetched miss, the GPU kernel performs the operation

$$z_n = \sum_k a_k\, v(c_{n,k})\, 2^{e_{n,\lfloor k/32 \rfloor} - 127} \tag{16}$$

inside the matrix or vector reduction. Scales are loaded once per 32-weight block and broadcast. Decode uses route-proportional GEMV; sufficiently dense prefill groups routes by expert so each tile decodes an expert weight block once. The CPU executor reads the same packed banks with no repack and returns a rank-local partial compatible with the GPU path.

The checkpoint's W4A8 reference also quantizes activations. FreeToken performs an FP8 activation round trip in a captured GPU kernel before sending the input to the CPU workers. The inherited single-rank optimization was measured at 12.85 to 15.65 $\text{tok}\,\text{s}^{-1}$, a 21.8 percent improvement, with bit-identical output [14]. Our TP work preserves that path for every rank. This number is cited as an inherited backend gain, not as a result introduced by the three PRs.

When several speculative rows request the same missing expert, the later PR orders the finite fetch budget by route multiplicity, then recency and expert ID. If m_e is the number of rows selecting missing expert e, the primary priority is

$$e_1 \prec e_2 \quad \text{when} \quad m_{e_1} > m_{e_2}. \tag{17}$$

This spends one PCIe transfer where it serves the most verification rows. It does not change the global LRU residency rule.

Figure 1 summarizes the final data path.

VII. Checkpoint-faithful DSpark

A. Draft structure

The attached drafter is not a generic one-token MTP head. It contains three MoE draft blocks, a 128-token sliding window, a low-rank Markov head, and a confidence head. It proposes $\gamma = 5$ tokens per pass. Target hidden states tapped at checkpoint-declared layers 40, 41, and 42 are projected into the drafter's context KV. The block begins with the last committed target token as anchor, followed by checkpoint noise tokens. The expensive draft backbone processes the block in parallel, while the light Markov stage samples left to right so proposal i conditions on proposal $i - 1$.

Several details were necessary for useful acceptance:

- target feature layer IDs are interpreted with the checkpoint's actual zero-based convention;
- context rows use absolute rotary positions rather than reusing the position-zero frequency;
- draft layers continue the target layer-ID namespace so target and draft experts share one cache without address collisions;
- draft context KV is updated only from state that actually commits; and

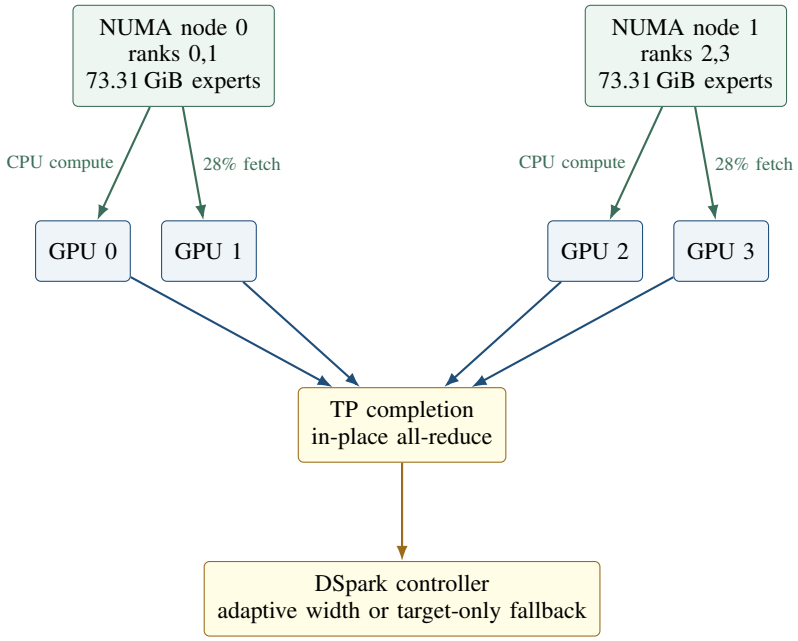

Fig. 1. NUMA-local expert ownership, hybrid CPU–GPU miss service, and TP completion. The diagram shows control and data dependence, not physical bus topology.

- the Markov and confidence heads remain replicated so TP ranks make the same lightweight decisions without a new collective per proposal.

B. Sampling exactness

The runtime forms both p and q after applying the request's temperature, top-p, and top-k policy. A sampled request then uses Equations 5 and 6. Comparing proposals with target argmax would be faster, but it would bias a temperature-1 request toward the mode. The optimized PR keeps proposed token probabilities, uniforms, first rejection, residual construction, and bonus sampling on the GPU. Only two integers cross to the host: accepted length and the emitted target token.

Every TP rank uses the same sampling stream and chooses the same physical graph span. This is required because a rank disagreement changes collective ordering and can deadlock the process group even if each rank's local token choice seems plausible.

C. Speculative state as a transaction

The target verifies anchor plus proposals before it knows the accepted prefix. Consequently, rollback cannot be implemented as a single token-buffer truncate. Table V lists the relevant owners.

The carry journal matters because several positions can share one 128-token ring page. Releasing whole pages cannot restore an earlier recurrent value inside a page that remains live. Similarly, writing a multi-token block through a single-token detokenizer path originally duplicated visible tokens. The final contract advances the engine and tokenizer through the accepted list once. Paged ownership follows the same general motivation as PagedAttention [21], but the DSV4 state set includes sliding-window and recurrent structures beyond ordinary KV.

VIII. Hardware-aware verification and GPU-side gains

A. Measured width selection

Let $c_i \in [0,1]$ be the confidence that proposal i survives, conditioned on the previous prefix. The survival probability through position k is

$$S_k = \prod_{i=1}^{k} c_i. \tag{18}$$

A target verification of width k emits one target token and, in expectation, the surviving draft prefix:

$$E(k) = 1 + \sum_{i=1}^{k} S_i. \tag{19}$$

Let D be measured draft time and $V(k)$ the median replay time of the target graph for anchor plus k proposals. The one-request scheduler selects

$$k^* = \operatorname*{argmax}_{k \in \{0,\dots,\gamma\}} \frac{E(k)}{D + V(k)}. \tag{20}$$

Width zero is legal and still emits one target token. Ties retain the smaller width. Measured graph costs are made monotonic before entering the selector so profiling noise cannot make a larger physical graph appear cheaper.

The runtime captures every legal one-request target span, from one row through six rows. It prices the drafter from the median of five real serving steps. Current confidence is copied asynchronously into one of two pinned host buffers;

TABLE V
COMMIT AND REJECTION RULES AFTER SPECULATIVE VERIFICATION.

State owner	Commit	Rejection or unused suffix
Request token buffer	Accepted proposals plus one target token	Truncate to accepted frontier
Target paged KV	Keep accepted rows	Return unused tail pages immediately
Draft sliding-window KV	Catch up from committed target rows	Do not advance from abandoned rows
SWA slots	Keep accepted positions	Release tail slots immediately
Target auxiliary features	Select the accepted physical row	Never commit a rejected feature row
Compressor and indexer carry	Commit journal row at accepted frontier	Restore the selected row when later speculative rows overwrote the same ring page
Detokenizer	Emit each accepted token exactly once	Stop at EOS or a stop token inside the block
Client lifecycle	Preserve owned committed state	Release speculative allocations on disconnect or abort

the decision consumes the previous ready buffer. This stale-confidence scheme avoids synchronizing current confidence into the same critical path and follows the deployment shape described by DSpark [2].

The measured final costs were approximately $D = 23.47\,\mathrm{ms}$ and

$$\begin{aligned} V(0\ldots2) &= [61.17, 92.24, 106.99]\ \mathrm{ms}, \\ V(3\ldots5) &= [133.64, 148.22, 157.79]\ \mathrm{ms}. \end{aligned} \tag{21}$$

as recorded for the final hybrid configuration [14], [15]. These numbers are specific to the final TP=4 hybrid path and its cache state. They also explain the observed throughput ceiling. With full five-proposal acceptance, a cycle emits six tokens and is bounded by

$$R_{\max} = \frac{6}{(23.47 + 157.79) \times 10^{-3}} = 33.1\ \mathrm{tok\,s^{-1}}. \tag{22}$$

At 84 percent accepted proposals, a simple observed-count estimate emits $1 + 5(0.84) = 5.2$ tokens per cycle, giving

$$R_{0.84} \approx \frac{5.2}{0.18126} = 28.7\ \mathrm{tok\,s^{-1}}, \tag{23}$$

which agrees with the 28–30 $\mathrm{tok\,s^{-1}}$ coding region. This bound shows why confidence tuning alone cannot yield 60 $\mathrm{tok\,s^{-1}}$ without a substantial reduction in draft or target-verify cost.

B. Kernel and collective changes

Three GPU-side changes were retained in the final PR stack.

First, hyper-connection pre-normalization originally materialized a full FP32 copy and a second FP32 square tensor before reducing to one inverse RMS value per token. A fused kernel reads the BF16 source once and accumulates squares in FP32. On an RTX 6000 Ada microbenchmark at the DSV4 hidden geometry, the $T = 8192$ operation fell from 1.944 to 0.304 ms, a 6.4-fold kernel speedup. That upstream optimization originated with Gabriel Devenyi's FreeToken PR #101 [22]; our change extends it to the DSpark drafter and adds long-prefill bounds. The Ada microbenchmark is not presented as an A4000 end-to-end result.

Second, TP all-reduce now operates on its contiguous input in place. The older path copied tensors into and out of a symmetric staging window around every reduction. Direct NCCL removes those two device copies and eliminated the symmetric collective path that dominated an earlier TP=4 verification trace on the PCIe A4000 configuration.

Third, probabilistic verification remains on device as described above. The optimization removes full-vocabulary device-to-host transfers and redundant target argmax work. It does not truncate distributions or approximate the p/q test.

CUDA graphs reduce repeated launch overhead by capturing a fixed operation graph and replaying it [23]. The target verifier benefits because its legal physical widths can be captured separately. A later attempt to graph the drafter reduced the measured drafter stage but did not produce a meaningful end-to-end improvement and increased jitter. Section XI-D reports that result rather than treating graph capture as automatically beneficial.

IX. ACCEPTANCE-GUIDED FALLBACK

A. Motivation

The adaptive selector in Equation 20 chooses how many proposals to verify after the drafter has already run. If a high-entropy request repeatedly rejects proposals, the selector cannot recover the 23.47 ms draft cost already paid. Structured code often has high local predictability; open-ended prose and difficult reasoning can have substantially lower proposal acceptance. A workstation serving one request should not continue paying the same draft tax merely because the prompt was initially eligible for DSpark.

The extension observes actual acceptance rather than prompt text. For a request-local observation window, define

$$\widehat{\alpha} = \frac{A}{G}, \tag{24}$$

where A is the number of accepted draft tokens and G is the number of verified draft proposals since the last decision. With threshold τ, minimum sample m, and target-only interval C, the controller is

$$\text{mode} \leftarrow \begin{cases} \text{target-only for } C \text{ steps}, & G \geq m \text{ and } \widehat{\alpha} < \tau, \\ \text{DSpark}, & G \geq m \text{ and } \widehat{\alpha} \geq \tau. \end{cases} \tag{25}$$

After exactly C target steps, the controller permits a DSpark probe and begins a fresh measurement window. New request identifiers reset all counters so one request cannot inherit another request's decision.

The evaluated settings are

$$(\tau, m, C) = (0.60, 32, 64). \tag{26}$$

The target-only path still commits each generated target feature into the drafter's context KV. A later probe therefore resumes from current state rather than from stale draft context. The controller is disabled by default in the submitted code.

B. Nature of the contribution

The fallback is an engineering extension to DSpark, not a change to the exact acceptance rule. When DSpark runs, Equations 5 and 6 remain unchanged. When target-only mode runs, output is drawn directly from p. Switching between these two target-correct modes preserves the requested sampling distribution.

The useful novelty is the composition of four properties: request-local measured acceptance, a bounded target-only interval, drafter-state maintenance during that interval, and re-entry probes. It addresses a cost that confidence-only width selection cannot reclaim after drafting. We do not make an exhaustive priority claim over every speculative decoder. The narrower claim is verifiable in the artifact: this controller was added to the FreeToken DeepSeek-V4 DSpark path, tested with its hybrid state machinery, and measured on a workload where the drafter was persistently unprofitable.

X. Operational work required for a stable service

The model did not become a service when the first correct token appeared. The following operational changes were required to repeat tests and to preserve the host after failures.

- **Compiler selection.** The workstation's default GCC 16 was newer than the CUDA JIT accepted. The service points NVCC and host C++ compilation to GCC/G++ 15.
- **Memory geometry.** A memory ratio of 0.90 auto-sized 2,509 MoE cache slots and left 1.02 GiB before graph capture, after which DSpark verification capture failed out of memory. A ratio of 0.80 selected 2,016 slots, left 2.56 GiB before capture and 1.26 GiB afterward, and started successfully.
- **KV capacity.** The validated deployment uses 477 full-KV pages, or 61,056 allocated tokens for prompt plus output. The architecture supports a much larger context, but architecture limit and allocated workstation capacity are different quantities.
- **Prefill bound.** The scheduler honors a 2,048-token prefill chunk so a long history does not materialize an unbounded activation. This setting was validated with a 9,139-token prompt followed by a coherent 200-token response.
- **Loading and rendezvous.** Expert loading is serial under the proven profile, and distributed startup timeout is 1,800 seconds to tolerate legitimately skewed rank-local bank loading.
- **Readiness.** The HTTP frontend binds before expert banks finish loading. The manager waits for the engine's `ready to serve` record rather than treating an HTTP response as model readiness.
- **Teardown.** TP workers are detached multiprocessing children and may retain the rendezvous port after the frontend stops. The management script resolves only service-specific ranks and shared-memory objects. It avoids broad process-name termination.
- **Deployment provenance.** Source changes reach the Lenovo checkout only through a named branch pushed to the GitHub fork, followed by fetch, switch or fast-forward pull, and server-side tests at that commit. Direct source-file copies are excluded from the procedure.

These items are part of the feasibility result. A model that starts only after manual process cleanup, or whose long prompt OOMs after a short benchmark, is not a reproducible workstation deployment.

XI. Evaluation

A. Method

Measurements use one active request at TP=4. Sampling is temperature 1.0 unless the diagnostic explicitly says greedy. The hybrid expert fetch fraction is 0.28, MoE cache size is 2,016 slots, and the graph-safe memory ratio is 0.80. The principal measurements are recorded in the branch artifact [14] and in the PR descriptions [15], [17].

Three reporting rules prevent misleading results.

1) Prefill throughput and decode throughput are reported separately. A short prompt cannot test the 2,048-token prefill path.
2) The first status interval after idle time is excluded from steady-state decode because it includes idle or calibration time.
3) Speculative counters are cumulative since server start. Per-request acceptance subtracts the counter values immediately before and after the request.

The evaluation is an engineering A/B record, not a statistically powered model benchmark. Output was inspected for coherence, repetition, termination, and task completion. No automated semantic quality score is claimed.

B. Decode performance

Table VI combines the stable PR-stack results and later controlled experiments. "Steady" denotes server windows after warmup; "end-to-end" includes the request's observed wall time. Rows from different prompts should not be compared as though they were replicates.

The structured-code improvement over the approximately 18 $\mathrm{tok\,s^{-1}}$ target-only baseline is the primary GPU and DSpark gain: 27.7–29.7 $\mathrm{tok\,s^{-1}}$, or 1.54–1.65×. Later controls reached approximately 30 $\mathrm{tok\,s^{-1}}$ at higher acceptance. These results align with the cycle-cost estimate in Equation 22.

The fallback result is narrower but informative. The matched creative request improved from approximately 15.7 to 17.18 $\mathrm{tok\,s^{-1}}$, a rate increase of about 9.4 percent. Wall time fell from approximately 98 seconds to 88.07 seconds, about 10.1 percent, while the natural completion lengths differed slightly. During fallback, target-only windows reached 17.86–19.59 $\mathrm{tok\,s^{-1}}$; mixed probe windows could still fall to approximately 15–17

TABLE VI
SINGLE-REQUEST GENERATION RESULTS ON THE LENOVO WORKSTATION.

Configuration	Workload	Acceptance	Throughput	Interpretation
Target-only baseline	Reference decode	n/a	about 18 $\mathrm{tok\,s^{-1}}$	PR 3 comparison baseline
DSpark PR stack	Structured code	65–70%	27.7–29.7 $\mathrm{tok\,s^{-1}}$	1.54–1.65× baseline
Enhanced-v2 + 2048	Structured code control	81.6%	29.29, 30.08 $\mathrm{tok\,s^{-1}}$	Coherent 600-token output
Enhanced-v2 + 2048	25-horses reasoning	61.0%	18.37 $\mathrm{tok\,s^{-1}}$ mean	Correct strategy; budget cut final proof
Graph-fix lineage	Same reasoning control	51.7%	16.86 $\mathrm{tok\,s^{-1}}$ mean	Regression correlated with small widths
Fallback enabled	Structured code control	high; no useful trip	30.55, 27.33 $\mathrm{tok\,s^{-1}}$	Coherent; 21.11 s wall time
Fallback disabled	Creative prose	low	about 15.7 $\mathrm{tok\,s^{-1}}$ end-to-end	Inferred from matched server timestamps
Fallback enabled	Same creative prompt	30.3–59.4% windows	17.18 $\mathrm{tok\,s^{-1}}$ end-to-end	1,513 tokens in 88.07 s

TABLE VII
FIXED LONG-PROMPT PREFILL A/B.

Chunk size	Full-chunk throughput	Total prefill wall time
1,024 tokens	321.8–323.2 $\mathrm{tok\,s^{-1}}$	25.79 s
2,048 tokens	625.4–627.3 $\mathrm{tok\,s^{-1}}$	15.31 s
Change	approximately 1.94×	40.6% lower

TABLE VIII
MATCHED DRAFTER GRAPH A/B.

Mode	End-to-end rate	Steady mean	Jitter	Fallback entries
Graph	25.03 $\mathrm{tok\,s^{-1}}$	23.97 $\mathrm{tok\,s^{-1}}$	4.84 $\mathrm{tok\,s^{-1}}$ std. dev.	12
Eager	24.78 $\mathrm{tok\,s^{-1}}$	24.23 $\mathrm{tok\,s^{-1}}$	3.72 $\mathrm{tok\,s^{-1}}$ std. dev.	7

$\mathrm{tok\,s^{-1}}$. The completed response was coherent, non-repeating, and stopped naturally. This is one matched run and is reported as evidence for the controller, not a workload-wide average.

High-acceptance code should remain speculative. The 600-token code control did not trip the controller during its useful steady section. A 256-token warmup tripped only in its final 32-proposal tail, after local acceptance fell to 56.2 percent as the request ended. That late event did not affect its output, but it shows that a 32-proposal observation window is reactive and should be tuned with more repeated workloads.

C. Prefill performance

The largest confirmed wall-time improvement came from changing the server profile's default prefill chunk from 1,024 to 2,048 tokens. Table VII uses the same fixed 7,924-token prompt.

The gain comes from denser prefill work and better amortization, not from the acceptance fallback. A separate 9,139-token prompt plus 200-token response completed coherently without OOM, which is necessary because a faster chunk is not useful if it crosses the workstation's graph or activation budget.

D. Negative and bounded results

Several results constrain the claims.

a) Drafter graph.: A matched rich pet-store workload compared a captured drafter backbone with the eager drafter on the same experiment commit:

The graph reduced the drafter substage from about 21.71 to 18.43 ms, but improved end-to-end rate by only about one percent and increased variance and fallback entries. Both outputs were coherent. The deployed service therefore keeps the eager drafter, with graph replay behind an opt-out switch until repeated tests explain the variance.

b) Reasoning termination.: Bounded bridge-and-torch prompts found the correct strategy without repetition, but both high and low reasoning effort consumed a 1,200-token budget while formalizing the proof and failed to enter the final answer channel. A similar 25-horses run found the correct seven-race strategy but was cut during its lower bound. These are serving-quality failures even when intermediate reasoning is correct. Output budgets and termination behavior require a separate fix.

c) Capture budget.: The 0.90 memory-ratio failure demonstrates that more cache residency is not monotonically better. Additional MoE slots consumed the reserve required to capture DSpark verification graphs. The successful 0.80 point retains 1.26 GiB after capture and is the validated setting.

d) Concurrency.: Adaptive width selection is presently specialized to one active request. The DSpark paper's multi-request scheduler uses a global capacity decision across flattened variable-length requests. Padding every request to a single maximum width is not equivalent. No multi-request throughput result is claimed here.

E. Validation coverage

At submission time, the complete three-PR stack passed 192 focused DSV4 tests on the TP=4 server. The fallback lineage passed 202 DSpark and hybrid-fetch tests at its exact pushed commit [14]. Coverage includes:

- TP=1, 2, and 4 shape ownership, quantization alignment, independent storage, vocabulary coverage, and invalid-layout failures;
- checkpoint DSpark wiring, target taps, absolute rotary positions, Markov order, and non-causal draft attention;
- greedy prefix acceptance and sampled p/q acceptance, including disagreement, residual recovery, full acceptance, and width zero;
- every legal target graph width and TP-consistent graph selection;
- request-local stale-confidence and acceptance-fallback reset;
- paged KV, SWA, recurrent carry, auxiliary-feature, and client-disconnect cleanup;

- device-resident probabilistic verification and single host result transfer;
- NUMA placement, bandwidth split, route-aware fetch, and final placement accounting; and
- long-prefill bounds, fused inverse RMS accuracy, and output-budget boundaries.

XII. Discussion

A. What made the model feasible

No single optimization explains the result. MXFP4 reduces expert storage, but the checkpoint still exceeds aggregate VRAM. CPU offload adds capacity, but an expert miss over PCIe can dominate a short decode step. Tensor parallelism adds VRAM, but it also adds collectives and creates four host-memory consumers. DSpark amortizes target work, but low acceptance can make its draft cost counterproductive. NUMA binding improves locality, but only if it occurs before expert-bank first touch. CUDA graphs reduce launch overhead, but only if fixed buffers and state ownership remain correct across replay.

Let $\mathcal{F}$ denote practical serving. The feasibility chain is conjunctive:

$$\begin{aligned}\mathcal{F} = {} & \text{rank-local storage} + \text{MXFP4 execution} \\ & + \text{NUMA-local banks} + \text{hybrid miss service} \\ & + \text{exact speculative state} + \text{bounded graph memory}.\end{aligned} \tag{27}$$

Removing any one term either breaks correctness, exceeds memory, or leaves a large avoidable latency on this machine.

B. Engineering novelty and scientific restraint

Each foundational idea has prior work. Tensor parallelism, speculative sampling, paged state, microscaling formats, offload, CUDA graphs, NUMA placement, and bandwidth-adaptive MoE execution all predate this case study [8], [9], [11]–[13], [18], [20], [21], [23]. DSpark supplies the trained drafter and confidence-scheduled verification [2].

The contribution is at the integration boundary. To our knowledge from the submitted artifacts, this is the first FreeToken implementation that combines the official DeepSeek-V4-Flash MXFP4 checkpoint, TP=4 Ampere execution, rank-local CPU expert banks, exact DSpark state management, measured adaptive verification, and a request-local observed-acceptance fallback. This statement identifies the contribution within the project and PR series. It is not a claim that no unpublished or independent implementation has assembled a similar system.

The fallback is especially useful as a systems result because it turns workload sensitivity into a measured online decision. It does not label poetry, reasoning, or code in advance. It observes whether the exact target accepted the drafter's work and changes execution only when enough evidence accumulates. The matched creative result suggests that this mechanism can recover roughly ten percent on a low-acceptance request while leaving a high-acceptance code request on DSpark. More repetitions are required before selecting production defaults.

C. Next experiments

The measured costs suggest a disciplined order for future work.

1) Sweep hybrid fetch fractions 0.20, 0.24, 0.28, 0.32, and 0.36 under the actual TP=4, two-ranks-per-socket workload. Hold MoE slots and KV pages fixed.
2) After choosing the split, sweep 2,080, 2,144, and 2,208 MoE slots while retaining at least 768 MiB after all graph capture.
3) Improve fallback probes so an unprofitable re-entry block can return to target-only sooner than another full 32-proposal window.
4) Measure confidence calibration against actual prefix survival before changing thresholds or scheduler mathematics.
5) Revisit drafter graph replay only with repeated end-to-end tests that improve mean rate without increasing jitter, changing RNG semantics, or reducing acceptance.
6) Add multi-request variable-width scheduling and then evaluate throughput, latency, memory, tool calls, and disconnect behavior under concurrency.

The protected recovery branch remains `milestone/dsv4-dspark-30tps` at `986823e`. The confirmed production baseline remains `prod/dsv4-dspark-enhanced-v2` at `2ce0070`. Experimental results should not advance either reference point until repeated A/B tests pass throughput, quality, stability, memory, long-context, and tool-use gates.

XIII. Threats to validity

a) One machine.: The CPU bandwidth, PCIe topology, NUMA distances, GPU clocks, and cache behavior are specific to this Lenovo workstation. Equation 14 is portable; its measured inputs are not.

b) One-request focus.: Results measure single-user generation. They do not establish aggregate throughput, tail latency, or fairness under concurrent requests.

c) Limited repetitions.: The fallback comparison uses one matched creative workload. Its baseline wall time was inferred from server timestamps, and natural completion lengths differ. The effect is promising but not statistically conclusive.

d) Qualitative output checks.: Coherence, repetition, and completion were inspected manually. This detects obvious state or detokenization faults but does not replace benchmark accuracy, perplexity, or human preference studies.

e) Changing upstream.: The three PRs were open at inspection time and rebased while under review. Their heads and cumulative diff sizes may change. This paper records exact inspected heads and protected local baselines so claims remain traceable [15]–[17].

f) Format naming and model size.: The public 0731 model card labels the repository as 304B parameters, while the original V4 technical report describes the preview Flash model as 284B total and 13B activated [1], [3]. This report avoids deriving memory from either parameter label and instead uses the inspected 156 GiB checkpoint and measured runtime placement. The expert weight format is called MXFP4 here

because its block geometry is E2M1 plus E8M0 per 32 values; it is not NVIDIA NVFP4.

XIV. Reproducibility and artifact map

The source artifacts are public in the three upstream PRs and Carlos Alvarado's fork. The primary implementation records are the TP runtime in PR #70 at `c066b38`, exact DSpark in PR #69 at `2cf938b`, and the adaptive stack in PR #71 at `4800af0`. The protected recovery point is `986823e`, the production baseline is `2ce0070`, and the performance record cited here is `e915ea3`.

A deployment intended to reproduce the results should use a pushed named branch, preserve any server-side dirty state with a named Git stash, fetch the fork, switch to the pushed branch, and run server-side tests from the exact checked-out commit before serving. Copying source files directly into the server checkout would break provenance and is outside the validated procedure.

At minimum, a reproduction record should include:

- Git commit and whether DSpark fallback and drafter graph are enabled;
- TP rank-to-GPU and rank-to-NUMA mapping;
- CPU worker cores, expert-bank bytes, per-rank VRAM, and post-capture free memory;
- MoE slots, KV pages, memory ratio, prefill chunk, and hybrid fetch fraction;
- target verify curve, median draft cost, and sampling policy;
- request-local accepted and drafted deltas;
- prompt, input tokens, completion tokens, wall time, steady windows, and stop reason; and
- coherence, repetition, final-answer, OOM, disconnect, and tool-call gates.

XV. Conclusion

Serving DeepSeek-V4-Flash-0731 on four RTX A4000 cards required a redefinition of where model state lives and how each token moves through the workstation. The TP runtime shards weights, scales, vocabulary, heads, and expert intermediate dimensions. NUMA placement binds each rank before first-touch allocation. The hybrid backend balances measured CPU expert throughput against PCIe gather throughput. Portable inline-dequant kernels execute the checkpoint's MXFP4 experts without pretending that Ampere provides native FP4. Exact DSpark integrates proposal, verification, rejection, state commit, and detokenization across that heterogeneous path.

The resulting measured placement is 72.6 percent CPU and 27.4 percent GPU. Structured code reaches approximately 28–30 $\mathrm{tok\,s^{-1}}$ from an approximately 18 $\mathrm{tok\,s^{-1}}$ target-only baseline. Long-prompt prefill reaches approximately 626 $\mathrm{tok\,s^{-1}}$ in full 2,048-token chunks. On one matched low-acceptance creative request, the acceptance-guided fallback reaches 17.18 $\mathrm{tok\,s^{-1}}$ compared with an approximately 15.7 $\mathrm{tok\,s^{-1}}$ no-fallback run. These results do not erase the workstation's bandwidth limits. They show that the checkpoint becomes practical when memory ownership, numeric format, speculative correctness, PCIe service, and NUMA locality are designed as one system.

References

[1] DeepSeek-AI, A. Xu, B. Lin *et al.*, "DeepSeek-V4: Towards highly efficient million-token context intelligence," *arXiv preprint arXiv:2606.19348*, 2026. [Online]. Available: https://arxiv.org/abs/2606.19348

[2] X. Cheng, X. Yu, C. Shao, J. Li, Y. Xiong, Y. Qian, J. Zhu, S. Ma, X. Zhang, J. Ye, Q. Chen, C. Deng, J. Yu, D. Dai, Z. Zhang, Y. Wei, Y. Tan, W. Yang, R. Xu, Y. Wu, Z. Xu, X. Wang, M. Chen, R. Tian, X. Bi, Z. Hao, S. Chen, H. Cao, W. Zhang, A. Xu, H. Zhang, D. Zhao, and W. Liang, "DSpark: Confidence-scheduled speculative decoding with semi-autoregressive generation," *arXiv preprint arXiv:2607.05147*, 2026. [Online]. Available: https://arxiv.org/abs/2607.05147

[3] DeepSeek-AI, "DeepSeek-V4-Flash-0731 model card," Hugging Face model repository, 2026, accessed: 25 August 2026. [Online]. Available: https://huggingface.co/deepseek-ai/DeepSeek-V4-Flash-0731

[4] NVIDIA Corporation, "NVIDIA RTX A4000 data sheet," NVIDIA Corporation, Tech. Rep., 2021. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/gtcs21/rtx-a4000/nvidia-rtx-a4000-datasheet.pdf

[5] NVIDIA Corporation, *Ampere Tuning Guide*, NVIDIA Corporation, 2021. [Online]. Available: https://docs.nvidia.com/cuda/archive/11.4.1/ampere-tuning-guide/index.html

[6] E. Alvarez, O. Almog, E. Chung, S. Layton, D. Stosic, R. Krashinsky, and K. Aubrey, "Introducing NVFP4 for efficient and accurate low-precision inference," NVIDIA Technical Blog, Jun. 2025. [Online]. Available: https://developer.nvidia.com/blog/introducing-nvfp4-for-efficient-and-accurate-low-precision-inference/

[7] D. Dai, C. Deng, C. Zhao, R. X. Xu, H. Gao, D. Chen, J. Li, W. Zeng, X. Yu, Y. Wu, Z. Xie, Y. K. Li, P. Huang, F. Luo, C. Ruan, Z. Sui, and W. Liang, "DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models," *arXiv preprint arXiv:2401.06066*, 2024. [Online]. Available: https://arxiv.org/abs/2401.06066

[8] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, B. Chen, P. Liang, C. Ré, I. Stoica, and C. Zhang, "FlexGen: High-throughput generative inference of large language models with a single GPU," in *Proceedings of the 40th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 202. PMLR, Jul. 2023, pp. 31 094–31 116. [Online]. Available: https://proceedings.mlr.press/v202/sheng23a.html

[9] S. Yang, X. Fan, M. Pan, H. Xi, Z. Wang, S. Sun, K. Keutzer, S. Han, M. Zaharia, C. Xu, and I. Stoica, "FreeToken: Efficient edge-native MoE serving with bandwidth-adaptive execution," *arXiv preprint arXiv:2608.16157*, 2026. [Online]. Available: https://arxiv.org/abs/2608.16157

[10] Open Compute Project, "OCP microscaling formats (MX) specification, version 1.0," Open Compute Project Foundation, Tech. Rep., Sep. 2023. [Online]. Available: https://www.opencompute.org/documents/ocp-microscaling-formats-mx-v1-0-spec-final-pdf

[11] B. D. Rouhani, R. Zhao, A. More, M. Hall, A. Khodamoradi, S. Deng, D. Choudhary, M. Cornea, E. Dellinger, K. Denolf, D. Stosic, V. Elango, M. Golub, A. Heinecke, P. James-Roxby, D. Jani, G. Kolhe, M. Langhammer, A. Li, L. Melnick, M. Mesmakhosroshahi, A. Rodriguez, M. Schulte, R. Shafipour, L. Shao, M. Siu, P. Dubey, P. Micikevicius, M. Naumov, C. Verrilli, R. Wittig, D. Burger, and E. Chung, "Microscaling data formats for deep learning," *arXiv preprint arXiv:2310.10537*, 2023. [Online]. Available: https://arxiv.org/abs/2310.10537

[12] Y. Leviathan, M. Kalman, and Y. Matias, "Fast inference from transformers via speculative decoding," in *Proceedings of the 40th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 202. PMLR, Jul. 2023, pp. 19 274–19 286. [Online]. Available: https://proceedings.mlr.press/v202/leviathan23a.html

[13] C. Chen, S. Borgeaud, G. Irving, J.-B. Lespiau, L. Sifre, and J. Jumper, "Accelerating large language model decoding with speculative sampling," *arXiv preprint arXiv:2302.01318*, 2023. [Online]. Available: https://arxiv.org/abs/2302.01318

[14] C. Alvarado, "DeepSeek-V4-Flash DSpark performance log, 23 august 2026," FreeToken engineering artifact, 2026. [Online]. Available: https://github.com/calvarado2004/FreeToken/blob/e915ea3/docs/dspark-performance-log-2026-08-23.md

[15] C. Alvarado, "[3/3] perf(dsv4): add hardware-aware adaptive verification," FlashML-org/FreeToken pull request 71, 2026, open upstream pull

request, head 4800af0 as inspected 25 August 2026. [Online]. Available: https://github.com/FlashML-org/FreeToken/pull/71

[16] C. Alvarado, "[1/3] feat(dsv4): add tensor-parallel DeepSeek-V4 runtime," FlashML-org/FreeToken pull request 70, 2026, open upstream pull request, head c066b38 as inspected 25 August 2026. [Online]. Available: https://github.com/FlashML-org/FreeToken/pull/70

[17] C. Alvarado, "[2/3] feat(dsv4): implement exact DSpark speculative decoding," FlashML-org/FreeToken pull request 69, 2026, open upstream pull request, head 2cf938b as inspected 25 August 2026. [Online]. Available: https://github.com/FlashML-org/FreeToken/pull/69

[18] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019. [Online]. Available: https://arxiv.org/abs/1909.08053

[19] Linux kernel contributors, *NUMA Memory Policy*, The Linux Kernel Documentation, 2026, accessed: 25 August 2026. [Online]. Available: https://docs.kernel.org/admin-guide/mm/numa_memory_policy.html

[20] B. Lepers, V. Quéma, and A. Fedorova, "Thread and memory placement on NUMA systems: Asymmetry matters," in *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. Santa Clara, CA: USENIX Association, Jul. 2015, pp. 277–289. [Online]. Available: https://www.usenix.org/conference/atc15/technical-session/presentation/lepers

[21] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with PagedAttention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023, pp. 611–626. [Online]. Available: https://arxiv.org/abs/2309.06180

[22] G. Devenyi, "perf(dsv4): compute the hyper-connection pre-norm in one kernel," FlashML-org/FreeToken pull request 101, 2026, open upstream pull request, head 84c5b82 as inspected 25 August 2026. [Online]. Available: https://github.com/FlashML-org/FreeToken/pull/101

[23] NVIDIA Corporation, *CUDA Programming Guide: CUDA Graphs*, NVIDIA Corporation, 2026, accessed: 25 August 2026. [Online]. Available: https://docs.nvidia.com/cuda/cuda-programming-guide/04-special-topics/cuda-graphs.html