CACHE COMPLETE CONE PREFILL FOR LOCAL-ATTENTION DECODERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Prefill processes a prompt before generation begins and contributes directly to the time to first token. We propose *Cache Complete Cone Prefill* (C3P), a training- and calibration-free accelerator for local-attention decoders that retains pretrained weights and the native decoding interface. C3P traces dependencies backward from the requested logits and the cached state read by later decoding, then executes only the resulting *cone*. We prove that it reproduces both exactly in real arithmetic and is minimal among schedules that only delete original operations. We derive closed-form expressions for how many token positions each layer must process, explaining how prompt length, attention window size, and model depth determine the available savings. On an A100, C3P reduces BF16 Mistral-7B prefill latency by up to 10.6% over a FlashAttention-2 baseline that already uses final-layer slicing. On synthetic FP32 local decoders, it achieves up to $10.65\times$ speedup and 80.30% lower peak allocated memory over final-layer slicing. We further ask what information must be retained to preserve the prompt logits and future outputs when the decoding state is encoded differently. For a single attention layer with fixed keys, we determine the minimum information that must be retained from cached values to reproduce future outputs exactly, and how this requirement depends on the number of decoding steps. For purely local decoder stacks, we establish conditions under which the prompt logits and the logits of a sufficiently long continuation together uniquely determine the input vectors at every prompt position in the cone, so re-encoding cannot discard the information in those inputs. In sum, C3P combines training-free, plug-and-play prefill acceleration with rigorous guarantees on exactness and computational minimality.

1 INTRODUCTION

Before generating its first token, a decoder normally passes every prompt position through every layer. This *dense prefill* produces the requested prompt logits and the cached information read by subsequent decoding, which we call the *continuation state*. For next-token generation, only the last prompt position needs logits. This raises a practical question: how much prefill work can an existing model skip while preserving both results, without retraining or changing its decoding interface?

Local attention offers a structural opportunity. Each output depends on a recent window, so even after several layers its dependencies may cover only a suffix of a long prompt. Mistral-7B uses sliding-window attention throughout its stack (Jiang et al., 2023); other models mix local and global layers (Gemma Team, 2025; OpenAI, 2025; Amini et al., 2025). Computing every position at every layer can therefore do work that neither the requested logits nor future decoding will use.

A simple shortcut computes the final layer's attention output and feed-forward block only at the last position. We call this *final-layer slicing* (FLS). Extending it to earlier layers requires more care: the last query still needs keys and values from its window, those projections require lower-layer outputs, and later decoding needs its own cached values. These dependencies must be planned together across layers.

We propose *Cache Complete Cone Prefill* (C3P) to perform this planning. Starting from the logits and continuation state, it follows dependencies backward and evaluates only the resulting *cache-complete cone* (Figure 1). **C3P needs no training or calibration**: it uses the original weights and plugs into generation by returning the native state consumed by decoding. We prove preservation of the logits

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.

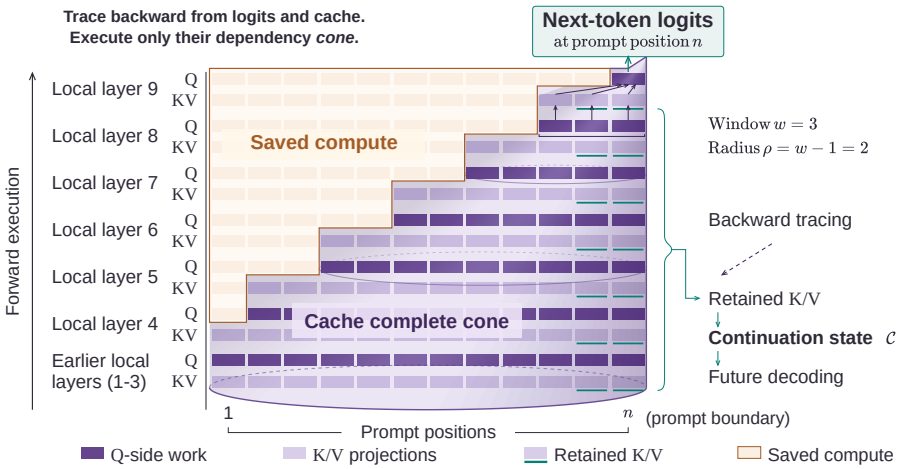

Figure 1: **Trace backward from both logits and the state needed for decoding.** A 12-token prompt, nine layers, and a three-token window ($n = 12$, $L = 9$, $w = 3$). Purple Q bands require full outputs, including queries, attention outputs, and feed-forward work; purple KV bands require key/value projections. Orange work is skipped, and teal entries form the continuation state. The top layer needs one full-output row and three K/V rows. Their dependencies expand the suffix by two positions per layer toward the input.

and state in real arithmetic, and minimality among schedules that only delete original operations. With FlashAttention-2, it saves 55–269 ms per Mistral-7B prefill beyond FLS on an A100 at 16,384 and 32,512 tokens.

The native cache representation need not itself be the smallest sufficient representation. For example, if future attention gives several values identical weights, their combined effect can be represented by a sum. We therefore also ask what information an exact implementation must retain when it may change the encoding. We answer this question for a fixed-key attention layer and for multilayer local decoders satisfying explicit observability conditions, connecting the execution cone to an information lower bound.

Our contributions are:

- **Training-free, plug-and-play prefill acceleration.** We introduce C3P, retaining pretrained weights and the native decoding interface while removing unnecessary computation. We prove exact preservation of the requested logits and continuation state in real arithmetic (Section 2).
- **Practical gains beyond optimized baselines.** We demonstrate 1.045–$1.119\times$ BF16 Mistral-7B speedups over FLS with FlashAttention-2, saving 55–269 ms per request. Synthetic FP32 local decoders achieve up to $10.65\times$ speedup and 80.30% lower peak allocated memory over FLS (Section 4).
- **Minimal computation and predictable work savings.** We prove that the cone is the unique inclusion-minimal choice among deletion-only schedules and derive closed-form full-output and K/V row counts for local attention (Sections 2 to 3).
- **Information limits beyond the native cache.** We derive minimum continuous sufficient value-state dimensions for fixed-key attention. For purely local multilayer decoders satisfying explicit observability conditions, we prove that the prompt logits and sufficiently long legal continuations identify the cone's input rows, constraining even re-encoded states (Section 5).

2 Cache Complete Cone Prefill

To skip work without changing the model's results, we first identify what prefill must produce. We then retain each computation needed to produce those values, working backward through the model.

2.1 Outputs and Continuation State

For a length-n prompt, the requested outputs are usually the logits at its last position n. The continuation state consists of the values and metadata that later decoding may read: cached keys

and values, convolution buffers where applicable, and their positions. A *legal continuation* appends tokens within the declared context length.

Represent dense prefill by a static computation graph $G = (\mathcal{V}, \mathcal{E})$. Its nodes are computed values, such as hidden rows, projections, attention outputs, and cache entries. An edge $u \to v$ means that the operator producing v can read u under its mask or receptive field. The architecture, prompt length, and requested interface determine this graph before any activations are evaluated. Let $\mathcal{O}$ and $\mathcal{C}$ be the nodes for the requested outputs and continuation state. Their union $\Omega = \mathcal{O} \cup \mathcal{C}$ gives the *targets* of the computation.

2.2 Backward Closure

If a value is needed, so are the inputs of its original operator. Repeating this rule backward retains all ancestors of the targets:

$$\mathrm{Cone}(\Omega) = \Omega \cup \mathrm{Anc}_G(\Omega). \tag{1}$$

We call this set the *cache-complete cone*. It is cache-complete because it includes the state needed by later decoding, as well as the requested logits. Its cone shape in a local decoder comes from dependencies widening toward the input. C3P evaluates its nodes with the original operators, installs the native continuation state, and skips the remaining nodes.

2.3 Exactness and Minimality

Every input of a retained operation is itself retained. The operation therefore receives the same values as in dense prefill, which gives the following guarantee.

Theorem 1 (Equality in real arithmetic)**.** *Suppose that G contains every input of every operator and every entry of the continuation state. Evaluate $\mathrm{Cone}(\Omega)$ with the original deterministic operators, parameters, inputs, masks, absolute positions, and cache conventions, and skip every other node. In real arithmetic, the result equals dense prefill on the requested outputs $\mathcal{O}$ and continuation state $\mathcal{C}$. Consequently, every legal continuation produces the same outputs as after dense prefill.*

Induction in graph order establishes equal retained values. Decoding then starts from equal states and receives the same tokens, so induction over decoding steps establishes equal future outputs (Appendix A).[1]

Could a schedule delete more while still using the same operators and state representation? A *deletion-only* schedule evaluates a subset of the original operations, each with its original inputs, while keeping the operators and state encoding unchanged.

Theorem 2 (Minimality among deletion-only schedules)**.** *Consider deletion-only schedules that select retained nodes from the static graph and interface alone, and produce the original targets Ω with their original operators and predecessor values. Every such schedule retains $\mathrm{Cone}(\Omega)$, which is therefore the unique inclusion-minimal choice.*

Deleting an ancestor would leave a retained operator without an original input. Thus **no deletion-only schedule planned from the static graph can skip more work than C3P**.

2.4 Full-Output and K/V Rows

To turn the graph into a schedule, group values by layer and position. A *row* is a hidden vector at one position; layer ℓ maps layer-$(\ell-1)$ rows to layer-ℓ rows, for $\ell = 1, \ldots, L$. Attention combines positions, whereas normalization, projections, residual addition, and the feed-forward block act separately at each position. Let $\mathcal{A}_\ell(i)$ contain the input positions read by attention for output i, and let $\mathcal{B}_\ell$ contain those needed to build the layer's continuation state. Other operators follow the same rule with their own input supports (Appendix B).

Let $\mathcal{R}_\ell$ be the positions requiring full layer outputs, starting from $\mathcal{R}_L = \{n\}$. Such rows need their queries, attention outputs, output projections, and feed-forward computation, collectively called

[1] Floating-point execution introduces rounding error; deriving bounds on the resulting deviations is left to future work in numerical analysis of neural-network inference.

Table 1: **Plan backward, execute forward.** Each attention operation uses the original parameters, absolute positions, masks, head mapping, and cache conventions.

Stage	Operation
Initialize	Set $\mathcal{R}_L = \{n\}$; read attention supports $\mathcal{A}_\ell(\cdot)$ and state supports $\mathcal{B}_\ell$.
Plan backward	For $\ell = L, \dots, 1$, form $\mathcal{P}_\ell$ and $\mathcal{R}_{\ell-1}$ by Equation (2).
Execute forward	For $\ell = 1, \dots, L$, normalize and project K/V on $\mathcal{P}_\ell$, and compute full outputs on $\mathcal{R}_\ell$.
Install and read out	Store the continuation state with its metadata; apply the final normalization and output head to the requested row.

Q-side work and labeled Q in Figure 1. Keys and values are needed both for these outputs and for the future cache. Their positions form $\mathcal{P}_\ell$, giving

$$\mathcal{P}_\ell = \mathcal{B}_\ell \cup \bigcup_{i \in \mathcal{R}_\ell} \mathcal{A}_\ell(i), \qquad \mathcal{R}_{\ell-1} = \mathcal{R}_\ell \cup \mathcal{P}_\ell. \tag{2}$$

The second union also retains each output's own input for its residual path. Positions in $\mathcal{P}_\ell \setminus \mathcal{R}_\ell$ need only K/V production, with no full output. This recurrence gives exactly the cone's required rows (Lemma 1). Planning runs backward and execution forward, as summarized in Table 1.

For example, in Figure 1, only position 12 needs a full output from layer 9. Its three-token window requires keys and values at positions 10–12, produced from layer-8 outputs; the cache for the next query retains positions 11–12, already included in that set. Thus layer 8 needs full outputs at all three positions, whose overlapping windows together require K/V at positions 8–12. In layer 9, positions 10 and 11 supply K/V without running their own attention outputs or feed-forward blocks. The same positions require full outputs one layer below: a row's role is determined separately at each layer. Continuing backward, layer 4 needs 11 full-output rows but K/V at all 12 positions. Layers 1–3 consequently run in full. The full-output and K/V counts therefore track different parts of the computation and can reach the prompt length at different layers.

3 Local-Attention Geometry

The recurrence identifies which rows to compute. For sliding-window attention these rows form prompt suffixes, so their lengths describe how much work remains. We derive these lengths and then examine how global layers change them.

3.1 Suffix Lengths

A causal window of size w_ℓ includes the current position and $\rho_\ell = w_\ell - 1$ earlier positions. Thus

$$\mathcal{A}_\ell(i) = \{\max(1, i - \rho_\ell), \dots, i\}. \tag{3}$$

The first decoding query, at position $n + 1$, reads the last ρ_ℓ prompt positions, so a minimal rolling cache retains $b_\ell = \min(n, \rho_\ell)$ of them (Jiang et al., 2023). More generally, allow a state buffer of b_ℓ positions. If the required full outputs form a suffix of length $q_\ell = |\mathcal{R}_\ell|$, the attention window expands it backward and the buffer can extend it further:

$$q_{\ell-1} = \min\{n, \max(q_\ell + \rho_\ell, b_\ell)\}, \qquad q_L = 1. \tag{4}$$

Both the attention inputs and the cached positions are suffixes ending at n. Their union therefore takes the larger length, explaining the maximum in Equation (4); the outer minimum clips it at the prompt start. The K/V rows form this same suffix, with $p_\ell = |\mathcal{P}_\ell| = q_{\ell-1}$. When $b_\ell \le q_\ell + \rho_\ell$ at every layer, as for the minimal rolling cache, the buffer adds no further dependencies. Hence

$$q_\ell = \min\Big\{n,\ 1 + \sum_{\ell'=\ell+1}^{L} \rho_{\ell'}\Big\}. \tag{5}$$

For uniform radius ρ, the full-output and K/V counts reduce to

$$q_\ell = \min\{n, 1 + (L - \ell)\rho\}, \qquad p_\ell = \min\{n, q_\ell + \rho\}. \tag{6}$$

K/V production extends ρ positions beyond full-output work until the suffix reaches the prompt start, explaining the two bands in Figure 1.

3.2 Work and Memory Savings

Call a layer *sliced* when it needs fewer than n full outputs. For $n > 1$ and $\rho > 0$, the number of such layers is

$$L_{\text{slice}} = \min\Big\{L, \Big\lceil \frac{n-1}{\rho} \Big\rceil\Big\}. \tag{7}$$

FLS slices only the last layer. Narrow windows let C3P slice many more; once the cone reaches the prompt start, additional lower layers are dense in both schedules and dilute the relative saving.

These row counts also determine the retained matrix computation. For blocks whose total query width equals the hidden width d, let d_{kv} be the total key or value width and d_{ff} the gated feed-forward width, as in our synthetic models and Mistral-7B. For a uniform window w, the last q queries read

$$N_{\text{pair}}(q) = \sum_{i=n-q+1}^{n} \min(w, i) \tag{8}$$

query-key pairs. The layer's multiply-accumulate (MAC) count is

$$M_\ell = q_\ell(2d^2 + 3dd_{\text{ff}}) + 2p_\ell dd_{\text{kv}} + 2dN_{\text{pair}}(q_\ell). \tag{9}$$

The terms count full-output projections and feed-forward work, K/V projections, and the two attention contractions. The total M sums these terms and the common output-head cost dn_{voc}, where n_{voc} is the vocabulary size. FLS has $q_\ell = p_\ell = n$ below the last layer, $q_L = 1$, and $p_L = \min(n, w)$. This count covers matrix products; measured latency also includes elementwise operations, memory traffic, and kernel overheads. Appendix A.6 evaluates the formula on 1,344 configurations.

The savings come from computation and intermediate tensors. With uniform minimal rolling caches of length $b = \min(n, \rho)$, all schedules retain the same parameters and $2Lbd_{\text{kv}}$ cache entries. C3P can reduce the peak memory used to produce them while keeping the native cache layout. Queries, attention outputs, and feed-forward activations are produced only for full-output rows, and K/V projections only for K/V rows. Their intermediate tensors therefore shrink with q_ℓ and p_ℓ.

3.3 Global Layers

A global attention layer reads and caches K/V from all prompt positions. Every layer below it must therefore produce all positions, although the global layer itself can still restrict its full-output rows. Only that layer and the layers above it can be sliced. This explains why local/global mixtures can offer a smaller opportunity than fully local stacks (Appendix B). We next test whether these structural predictions translate into measured gains, including with optimized attention kernels.

4 Experiments

We evaluate three consequences of the schedule: how window size and depth affect performance, whether gains persist with FlashAttention-2, and how global layers constrain the gains in mixed architectures.

4.1 Setup and Schedule Validation

Dense, FLS, and C3P use the same weights, masks, positions, and continuation-state layout. FLS isolates the saving beyond the final-layer shortcut. The synthetic experiments share a tiled PyTorch scaled dot-product attention (SDPA) executor and use 8,192-token prompts at batch size one on an A100-SXM4-40GB. Depths $L \in \{4, 8, 16, 32, 64\}$ and windows $w \in \{32, 128, 512, 2048, 8192\}$ give 25 configurations, each with three independent initializations. FP32 runs disable TF32. Timings cover planning through readout, including cache installation; speedups are geometric means of paired FLS/C3P latency ratios over 20 repetitions and the three initializations (Appendix C).

We check planning against explicit graph reachability in 1,764 configurations, with identical row sets. An independent dense CPU implementation checks execution in FP64 on eight small configurations, with maximum errors below 9×10^{-16}. On all 75 FP32 cases, both sliced schedules match dense outputs and caches within absolute and relative tolerances of 2×10^{-5} and 2×10^{-4} at prefill and eight fixed continuation inputs. Appendix C.1 gives the checks and error metrics.

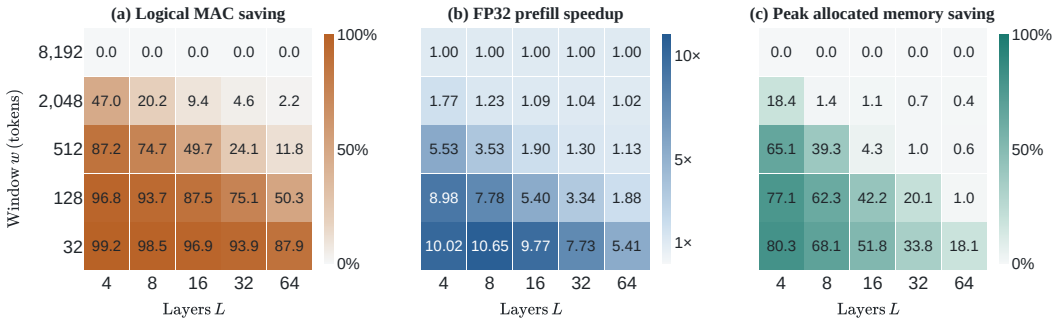

Figure 2: **The geometry predicts which configurations benefit.** All 25 depth and window configurations with 8K-token prompts, relative to FLS. (a) Logical MAC saving from Equation (9). (b) A100 FP32 prefill speedup over three initializations and 20 paired repetitions each. (c) Peak allocated memory reduction, using each schedule's maximum across initializations. Parameters and cache layouts are identical across schedules.

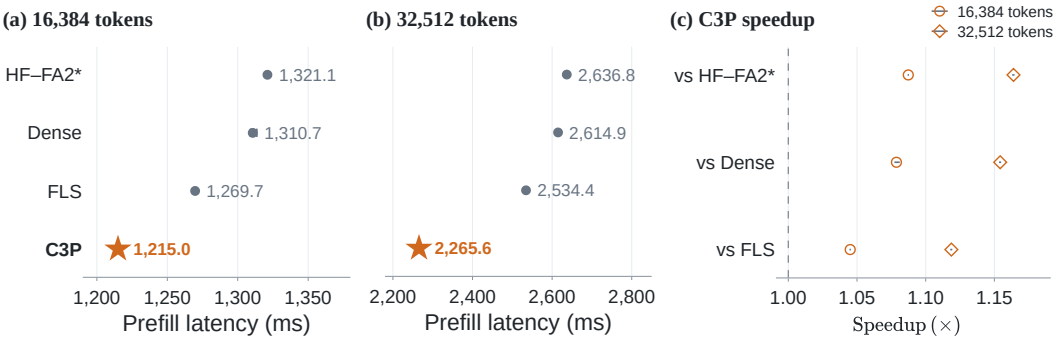

Figure 3: **C3P gains persist with FA2.** BF16 Mistral-7B on an A100 40GB. (a,b) Mean prefill latency at the indicated prompt lengths, with separate horizontal scales; stars mark C3P. (c) Paired geometric speedups over each comparator; circles denote 16,384 tokens and diamonds 32,512. Error bars are 95% within-text paired bootstrap intervals over three texts and 20 paired repetitions; those in (c) are very short. FLS is final-layer slicing; HF–FA2* aligns the window convention with the shared FA2 executor (Appendix C.3).

4.2 Synthetic Grid

Figure 2 follows the geometric prediction: narrow windows leave a smaller cone, while a full-prompt window removes no additional MACs. For every window shorter than the prompt, C3P is faster than FLS, by 1.021–10.647×. The maximum occurs at $L = 8$, $w = 32$; peak allocated memory falls by up to 80.30% at $L = 4$, $w = 32$. At $L = 32$, $w = 128$, logical MACs fall by 75.12%, prefill runs 3.34× faster, and peak allocated memory falls by 20.12%.

At $w = 2048$, the measured speedups are within 6.3% below the MAC ratios; for $L = 4$, the speedup is 1.769× against a MAC ratio of 1.887×. Thus the counts explain the performance pattern while actual latency also reflects execution overhead.

4.3 Mistral-7B with FlashAttention-2

Does removing rows still help when attention itself is optimized? We run BF16 Mistral-7B with FlashAttention-2 (FA2) (Dao, 2024) on an A100 40GB at batch size one, using three fixed text fragments at lengths 16,384 and 32,512. Dense, FLS, and C3P share the FA2 executor; HF–FA2* is a Hugging Face comparator with matching window endpoints. Every method returns the last-position logits and installs the cache. Timings include alignment, padding, and cache installation (Appendix C.3).

FLS is the fastest measured comparator at both lengths. C3P improves on it by 1.0451× and 1.1187×, saving 54.75 and 268.88 ms per prefill (Figure 3). Every text–length pair has a 95% paired interval above one. Speedups over HF–FA2* are 1.0873× and 1.1639×. **Cross-layer row selection remains useful after optimizing the attention kernel.**

The growth with prompt length follows the local geometry. Mistral has 32 layers and a 4,096-token window, so Equation (7) gives five sliced layers at 16,384 tokens and eight at 32,512, compared with

Table 2: **Prefill gains across workloads.** A100 GPUs; K denotes 1,024 tokens. Synthetic ranges cover the 20 configurations with windows shorter than the prompt. Gemma and LFM2 use their protocols' median aggregates.

Workload / precision	Prompt length	Comparator	Speedup
Synthetic / FP32	8K	FLS	1.021–10.647×
Synthetic / BF16	8K	FLS	1.020–9.886×
Mistral / BF16, FA2	16,384 / 32,512	FLS	1.0451/1.1187×
Mistral / BF16, SDPA	8,192 / 16,384 / 32,736	FLS	1.012/1.045/1.120×
Gemma-3-4B / BF16	32K / 64K / 128K	Native HF SDPA	1.115/1.134/1.152×
LFM2-2.6B / BF16	8K / 16K / 32K	FLS	1.067/1.073/1.081×

one for FLS. Its wide windows make the cone reach the prompt start within a few layers, leaving most of the stack dense. A longer prompt lets the narrow upper-layer suffix persist for more layers. This explains both the smaller gains than on narrow-window synthetic models and the larger gains at the longer prompt.

Appendix D.2 provides per-text results, and Appendix D.3 reports the separate SDPA comparison.

4.4 Mixed Architectures

The global-layer rule predicts where hybrid models can benefit. C3P slices five of Gemma-3-4B's 34 layers and three of LFM2-2.6B's 30, starting at each model's last global layer (Appendix B.3). This yields 1.115–1.152× speedups over native Hugging Face SDPA on Gemma and 1.067–1.081× over FLS on LFM2. Table 2 summarizes these and the additional precision/backend results; Appendix D gives every configuration.

5 Information a Continuation State Must Keep

The experiments accelerate prefill while handing the decoder its original cache. Our execution theorem fixes that representation; we now ask whether changing it could save storage as well. An exact alternative must support the same future logits, but it can combine values that future outputs never distinguish. Consider uniform attention, which averages three cached scalar values and the current token's value. The caches $(1, -1, 0)$ and $(0, 0, 0)$ produce the same next attention output for every supplied token, since their sums agree. At the following step, their oldest entries expire, leaving old-value sums of -1 and 0, respectively. Identical appended tokens then give different attention outputs. Thus **reading a cached value does not imply that it must be stored separately**: the required information depends on how far decoding continues.

Definition 1 (Exact prefill)**.** *A prefill execution is* exact *if it returns the same requested outputs as dense prefill, together with a state under which every legal continuation produces the same outputs as after dense prefill.*

An exact prefill needs a state equivalent to the dense one over the remaining legal horizon. To make this precise for a finite horizon H, fix the model, prompt length, positions, and legal continuation inputs. Let $F(c, u)$ be the sequence of full logit vectors from state c under a supplied token sequence u. Two states are *equivalent through H steps* when

$$c \sim_H c' \quad \Longleftrightarrow \quad F(c, u) = F(c', u) \text{ for every legal } u \text{ with } 1 \le |u| \le H. \tag{10}$$

A *sufficient encoding* stores enough information to reproduce all these traces. States with identical future behavior can share an encoding. We measure storage by its number of real coordinates, including any prompt or prompt logits kept for later reuse. The dimension results allow cache values or prompt input vectors to vary over an open domain, so every coordinate can change slightly, and require the encoder to depend continuously on these variables. Continuity means that small changes in the original values cause small changes in the stored coordinates; dimension then measures how many independently variable components the encoding can preserve.

One layer with fixed keys. Fixing the keys makes the dependence on cached values linear. Consider an attention layer with window $b + 1$ and $b \ge 1$ cached rows, oldest first. Initial keys $\mathbf{k}_j$ are fixed and known; values $\mathbf{v}_j \in \mathbb{R}^{d_v}$ vary. Appended queries, keys, and values depend only on the supplied token and position. Each token has a position-independent query in a fixed finite set $\mathbf{q}_1, \ldots, \mathbf{q}_J$, each available at every step, and every token sequence through H is legal. The logits are $\mathbf{g}_t(u) + W_{\text{logit}}\mathbf{o}_t$,

where $\mathbf{o}_t$ is the attention output and $\mathbf{g}_t$ is known from the supplied tokens. The rank $r = \operatorname{rank} W_{\text{logit}}$ counts the independent value components visible in the logits.

Theorem 3 (Minimum sufficient value information)**.** *For $H \geq 1$, set $h = \min(H, b)$ and*

$$W_h = \left[e^{\mathbf{q}_a^\top \mathbf{k}_j}\right]_{1 \leq a \leq J,\ h \leq j \leq b},$$

with the attention scale absorbed into the queries. The minimum continuous sufficient encoding dimension on an open value-cache domain is

$$d_H = r\big(h - 1 + \operatorname{rank} W_h\big). \tag{11}$$

An encoding attaining it stores $W_{\text{logit}}\mathbf{v}_1, \ldots, W_{\text{logit}}\mathbf{v}_{h-1}$ and a row basis of the weighted tail sums $\sum_{j=h}^{b} e^{\mathbf{q}_a^\top \mathbf{k}_j} W_{\text{logit}}\mathbf{v}_j$, expressed in r readout coordinates.

The fixed keys and supplied tokens make the attention denominators and new-token contributions known. Removing these contributions reveals weighted sums of the surviving cached values. Comparing successive suffix sums exposes the first $h-1$ projected values separately; the remaining values appear only through the independent weighted sums counted by $\operatorname{rank} W_h$. Each row of W_h describes one query's weights on the tail, so its rank counts independent views of those values. For uniform attention this rank is one and $d_H = rh$: one step needs only the projected total; each additional step exposes one more projected value, until all b projected values are recoverable. With three scalar values and an identity readout, sufficient states for horizons one, two, and three are $(v_1 + v_2 + v_3)$, $(v_1, v_2 + v_3)$, and (v_1, v_2, v_3), respectively. The extra stored value at each longer horizon lets the decoder subtract the next expired value from the remaining total. More varied query weights can reveal several independent sums in a single step. Appendix E.1 proves sufficiency and minimality.

What future outputs reveal across layers. In a decoder stack, the caches all arise from the same prompt, so their independent degrees of freedom cannot simply be added. Instead, we ask which prompt inputs future logits reveal. For local radii ρ_ℓ, write $\Delta = \sum_{\ell=1}^{L} \rho_\ell$ and $s = \min(n, \Delta)$. Information can travel from an input to an output Δ positions later along the dependency chain through each window's oldest position. Reconstruction is possible if each layer's output uniquely determines that oldest input when newer inputs are fixed—that is, the map is *injective in its oldest input*.

Theorem 4 (A continuation identifies the prompt suffix)**.** *Consider a purely local width-d stack with continuous layer and native-cache maps, minimal rolling caches, and fixed positions and metadata. Suppose every full-window map is injective in its oldest input, and radius-zero maps and the final logit readout are injective. If $H \geq \Delta$ steps are legal, the logit trace of any fixed length-Δ token sequence identifies the last s prompt input rows. The minimum continuous sufficient encoding dimension on an open suffix domain is sd.*

For example, take two layers with two-token windows, so $\Delta = 2$, and a prompt of at least two tokens. The final prompt row reaches the second future output through layer-1 position $n+1$ and layer-2 position $n+2$. The other inputs along this path come from the two known appended tokens. Injectivity therefore lets this output identify that prompt row. The first future output then identifies the second-to-last row, since all newer inputs are now known. The general proof repeats this backward reconstruction through the suffix. Appendix E.2 handles the prompt-start boundary and proves the dimension bound. Appendix E.3 constructs a normalized nonlinear decoder satisfying these assumptions and extends them to an open set of content-dependent decoders, with an injective logit readout, an invertible combined value–output map on the full hidden space, and sufficiently small query/key and FFN output weights.

Adding the requested prompt logits. Future outputs identify the suffix needed for continuation, but prefill also returns the current prompt logits. Their dependencies extend one position farther back than those of the first future output. Once the newer suffix is known, the same injectivity argument identifies this additional row whenever it lies within the prompt.

Corollary 1 (Information in prompt logits and future outputs)**.** *Under Theorem 4, the prompt-boundary logits and future outputs together identify exactly the last $\min(n, \Delta + 1)$ input rows. Their minimum continuous sufficient representation dimension on an open input domain is $d \min(n, \Delta + 1)$.*

For the two-layer example, the future outputs identify the last two prompt rows; adding the prompt logits identifies a third when $n \geq 3$. These are exactly the $q_0 = \min(n, 1 + \Delta)$ input rows of the

cone in Equation (5). Under these conditions, **the cone also defines an information boundary**: re-encoding may change how this information is stored, but cannot remove it. The continuation state alone requires the sd coordinates of Theorem 4; the additional row, when present, is identified through the returned prompt logits.

6 Related Work

C3P connects three questions: which computations are needed, how to execute them efficiently, and what information future decoding can distinguish.

Selecting computation. Program slicing retains statements affecting a chosen result (Weiser, 1984; Ferrante et al., 1987; Horwitz et al., 1990). We use both logits and continuation state as targets and derive full-output/K/V schedules and local geometry. LazyLLM, POP, and UniPrefill skip work through importance, calibration, or dynamic sparsity (Fu et al., 2024; He et al., 2026; Fan et al., 2026); H2O, SnapKV, StreamingLLM, and Quest alter the retained or accessed context (Zhang et al., 2023; Li et al., 2024; Xiao et al., 2024; Tang et al., 2024). C3P instead selects dependencies while preserving the original operators and state.

Executing computation. FlashAttention, FlashMask, and FlashInfer optimize attention kernels (Dao et al., 2022; Dao, 2024; Wang et al., 2025; Ye et al., 2025). PagedAttention, SGLang, Sarathi-Serve, and DistServe organize memory, reuse, and requests (Kwon et al., 2023; Zheng et al., 2024; Agrawal et al., 2024; Zhong et al., 2024). C3P selects rows across layers, and our FA2 experiments demonstrate gains alongside kernel optimization. PrefillOnly studies workloads without continued decoding (Du et al., 2025).

Representing information. Minimal-state theory distinguishes states through future input–output behavior (Nerode, 1958; Hermann & Krener, 1977). Attention-space bounds (Haris & Onak, 2025; Chen et al., 2026), sufficient memories under composition (Nowak et al., 2026), and Transformer injectivity (Nikolaou et al., 2026) address related information questions. We characterize finite-horizon value information with expiring windows and conditional recovery of a multilayer prompt suffix. Appendix F gives detailed comparisons.

7 Discussion

Optimize for the generation interface. C3P shows that existing models can serve the same generation interface with less prefill work than dense execution performs. The opportunity comes from asking which values generation will actually consume and tracking their dependencies across layers. This principle connects model structure to serving cost: local windows permit narrow suffixes, while global layers determine how far pruning can propagate. It suggests designing attention layouts and compilers together, so that useful local structure becomes an executable saving.

From latency to resource use. For long-context assistants and retrieval-augmented generation, reducing work on each fresh prompt can lower both waiting time and the compute needed to serve repeated requests. Our measured Mistral savings correspond to approximately **15–75 GPU-hours per million prefills** at the tested lengths, creating potential capacity and cost savings.[2] On the synthetic FP32 grid, GPU energy per prefill falls by up to 94.14% relative to FLS (Table 6). Extending these measurements to complete serving workloads would quantify the environmental benefit alongside latency.

Beyond the native cache. Our information results open a second direction. In the fixed-key model, queries and continuation length determine which value combinations suffice, suggesting exact compact states for workloads with bounded generation. In purely local multilayer decoders satisfying the observability conditions, the prompt logits and future outputs identify information that no exact re-encoding can discard. Exact compact states could be particularly useful for short-answer generation and bounded-length structured outputs. A complementary direction is to integrate dependency planning into inference compilers for other static attention patterns and requested output positions. Combining such planning with batching and prefix reuse could extend the gains from individual prefills to complete serving workloads.

[2]This scales the measured batch-one latency differences linearly; realized provisioning and billing savings depend on the serving workload.

AI USE STATEMENT

We used generative AI tools for assisting in the writing of proofs, proposing or refining hypotheses, debugging code, revising the text, and polishing figures. We have not used generative AI tools for providing critical ingredients for proving mathematical claims, generating synthetic data sets, formulating mathematical claims, helping develop theoretical models or conceptual frameworks, designing or providing feedback on research methodology or experiments, implementing methods, assisting with translation, or interpreting results. Cleaning and reformatting datasets and supporting qualitative and thematic data analysis are not applicable to this work. We have reviewed all AI-assisted work. We checked the LLM-revised text and the LLM-assisted proofs, and verified the polished figures against the recorded measurements. We take responsibility for the final content of this work.

ETHICS STATEMENT

This work studies exact prefill for existing autoregressive language models and evaluates it on synthetic decoders, publicly released checkpoints, and public text. No human participants or personal data are involved.

REPRODUCIBILITY STATEMENT

Section 2 states the assumptions of Theorems 1 and 2, and the full proofs are provided in Appendices A and E. Operator supports, experimental protocols, and per-configuration results are provided in Appendices B, C, and D. Code and experimental data will be released upon publication and can be shared privately with reviewers upon request during the discussion period.

REFERENCES

A. Agrawal, N. Kedia, A. Panwar, et al. Taming throughput-latency tradeoff in LLM inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 117–134. USENIX Association, 2024.

J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebron, and S. Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.298.

A. Amini, A. Banaszak, H. Benoit, et al. LFM2 technical report. *arXiv preprint arXiv:2511.23404*, 2025. URL https://arxiv.org/abs/2511.23404.

I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020. URL https://arxiv.org/abs/2004.05150.

J. Y. Chen, Y. Feng, P. Indyk, M. Kapralov, E. Kochetkova, and B. Prokhorov. Towards tight bounds for streaming attention. *arXiv preprint arXiv:2606.07205*, 2026. URL https://arxiv.org/abs/2606.07205.

R. Child, S. Gray, A. Radford, and I. Sutskever. Generating long sequences with sparse transformers. arXiv preprint arXiv:1904.10509, 2019.

M. Cho, M. Rastegari, and D. Naik. KV-Runahead: Scalable Causal LLM Inference by Parallel Key-Value Cache Generation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 8578–8592. PMLR, 2024.

J. Cooper, I. Diakonikolas, M. Ma, and F. Sala. Expressivity-efficiency tradeoffs for hybrid sequence models. *arXiv preprint arXiv:2603.08859*, 2026. URL https://arxiv.org/abs/2603.08859.

T. Dao. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. In *International Conference on Learning Representations*, volume 2024, pp. 35549–35562, 2024.

T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, volume 35, pp. 16344–16359, 2022. doi: 10.52202/068431-1189.

K. Du, B. Wang, C. Zhang, et al. PrefillOnly: An inference engine for prefill-only workloads in large language model applications. *arXiv preprint arXiv:2505.07203*, 2025. URL https://arxiv.org/abs/2505.07203.

Q. Fan, H. Huang, Z. Wu, B. Wang, and R. He. UniPrefill: Universal long-context prefill acceleration via block-wise dynamic sparsification. *arXiv preprint arXiv:2605.06221*, 2026. URL https://arxiv.org/abs/2605.06221.

J. Ferrante, K. J. Ottenstein, and J. D. Warren. The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems*, 9(3):319–349, 1987. doi: 10.1145/24039.24041.

Q. Fu, M. Cho, T. Merth, S. Mehta, M. Rastegari, and M. Najibi. LazyLLM: Dynamic Token Pruning for Efficient Long Context LLM Inference. arXiv preprint arXiv:2407.14057, 2024. Presented at the ICML 2024 Efficient Systems for Foundation Models workshop.

Gemma Team. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024. doi: 10.48550/arXiv.2408.00118. URL https://arxiv.org/abs/2408.00118.

Gemma Team. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025. URL https://arxiv.org/abs/2503.19786.

I. Gim, G. Chen, S.-s. Lee, N. Sarda, A. Khandelwal, and L. Zhong. Prompt Cache: Modular attention reuse for low-latency inference. In *Proceedings of Machine Learning and Systems*, volume 6, pp. 325–338, 2024.

T. Haris and K. Onak. Compression barriers in autoregressive transformers. In *Proceedings of Thirty Eighth Conference on Learning Theory*, volume 291 of *Proceedings of Machine Learning Research*, pp. 2757–2785. PMLR, 2025. URL https://proceedings.mlr.press/v291/haris25a.html.

J. He, Z. Fu, J. Wang, and Q. Li. POP: Prefill-Only Pruning for Efficient Large Model Inference. arXiv preprint arXiv:2602.03295, 2026.

P. Henderson, J. Hu, J. Romoff, E. Brunskill, D. Jurafsky, and J. Pineau. Towards the systematic reporting of the energy and carbon footprints of machine learning. *Journal of Machine Learning Research*, 21(248):1–43, 2020. URL https://jmlr.org/papers/v21/20-312.html.

N. W. Henry, G. L. Marchetti, and K. Kohn. Geometry of lightning self-attention: Identifiability and dimension. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2408.17221.

R. Hermann and A. J. Krener. Nonlinear controllability and observability. *IEEE Transactions on Automatic Control*, 22(5):728–740, 1977. URL https://www.math.ucdavis.edu/~krener/1-25/10.IEEETAC77.pdf.

S. Horwitz, T. Reps, and D. Binkley. Interprocedural slicing using dependence graphs. *ACM Transactions on Programming Languages and Systems*, 12(1):26–60, 1990. doi: 10.1145/77606.77608.

A. Q. Jiang, A. Sablayrolles, A. Mensch, et al. Mistral 7B. *arXiv preprint arXiv:2310.06825*, 2023. URL https://arxiv.org/abs/2310.06825.

M. Khalaf, Y. Shamshoum, N. Hodos, Y. Sieradzki, and A. Schuster. QKV Projections Require a Fraction of Their Memory. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 64896–64918, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/6a3eb26cbfdc79d79b9230f25e8526fb-Paper-Conference.pdf.

H. Kim, G. Papamakarios, and A. Mnih. The Lipschitz constant of self-attention. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 5562–5571. PMLR, 2021. URL https://proceedings.mlr.press/v139/kim21i.html.

W. Kwon, Z. Li, S. Zhuang, et al. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626. Association for Computing Machinery, 2023. doi: 10.1145/3600006.3613165.

Y. Li, Y. Huang, B. Yang, et al. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems*, volume 37, pp. 22947–22970, 2024. doi: 10.52202/079017-0722.

A. S. Luccioni, Y. Jernite, and E. Strubell. Power hungry processing: Watts driving the cost of AI deployment? In *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency*, pp. 85–99. Association for Computing Machinery, 2024. doi: 10.1145/3630106.3658542. URL https://arxiv.org/abs/2311.16863.

L. Mariot and A. Leporati. Sharing secrets by computing preimages of bipermutive cellular automata. In *Cellular Automata: 11th International Conference on Cellular Automata for Research and Industry, ACRI 2014*, volume 8751 of *Lecture Notes in Computer Science*, pp. 417–426. Springer, 2014. doi: 10.1007/978-3-319-11520-7_43. URL https://lucamariot.org/files/papers/ml_acri_2014_postprint.pdf.

S. Merity, C. Xiong, J. Bradbury, and R. Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017.

A. Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4): 541–544, 1958.

G. Nikolaou, T. Mencattini, D. Crisostomi, A. Santilli, Y. Panagakis, and E. Rodolà. Language models are injective and hence invertible. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2510.15511.

F. Nowak, R. Cotterell, and R. Boumasmoud. A compositional theory of causally masked transformers. *arXiv preprint arXiv:2607.26988*, 2026. URL https://arxiv.org/abs/2607.26988.

OpenAI. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025. doi: 10.48550/arXiv.2508.10925. URL https://arxiv.org/abs/2508.10925.

R. Pan, Z. Wang, Z. Jia, et al. Marconi: Prefix caching for the era of hybrid LLMs. In *Proceedings of Machine Learning and Systems*, volume 7. MLSys, 2025.

V. J. Reddi, C. Cheng, D. Kanter, et al. MLPerf Inference Benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 446–459, 2020. doi: 10.1109/isca45697.2020.00045.

B. Saha and C. Ye. I/O complexity of attention, or how optimal is FlashAttention? In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 43024–43042. PMLR, 2024. URL https://proceedings.mlr.press/v235/saha24a.html.

R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni. Green AI. *Communications of the ACM*, 63(12):54–63, 2020. doi: 10.1145/3381831.

J. Shah, G. Bikshandi, Y. Zhang, V. Thakkar, P. Ramani, and T. Dao. FlashAttention-3: Fast and accurate attention with asynchrony and low-precision. arXiv preprint arXiv:2407.08608, 2024.

N. Shazeer. Fast transformer decoding: One write-head is all you need. arXiv preprint arXiv:1911.02150, 2019.

A. Simoulin, N. Park, X. Liu, and G. Yang. Memory-efficient fine-tuning of transformers via token selection. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 21565–21580. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.1202.

J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu. RoFormer: Enhanced transformer with rotary position embedding. arXiv preprint arXiv:2104.09864, 2021.

J. Tang, Y. Zhao, K. Zhu, G. Xiao, B. Kasikci, and S. Han. QUEST: Query-aware sparsity for efficient long-context LLM inference. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 47901–47911. PMLR, 2024.

G. Wang, J. Zeng, X. Xiao, et al. FlashMask: Efficient and Rich Mask Extension of FlashAttention. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 27641–27661, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/44c624d8f4e802453ad12593b2aa8e38-Paper-Conference.pdf.

M. Weiser. Program slicing. *IEEE Transactions on Software Engineering*, SE-10(4):352–357, 1984. doi: 10.1109/TSE.1984.5010248.

G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2024.

J. Yao, H. Li, Y. Liu, et al. CacheBlend: Fast large language model serving for RAG with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 94–109. Association for Computing Machinery, 2025. doi: 10.1145/3689031.3696098.

Z. Ye, L. Chen, R. Lai, et al. FlashInfer: Efficient and Customizable Attention Engine for LLM Inference Serving. In *Proceedings of Machine Learning and Systems*, volume 7. MLSys, 2025.

M. Zaheer, G. Guruganesh, K. A. Dubey, et al. Big Bird: Transformers for Longer Sequences. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 17283–17297. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/c8512d142a2d849725f31a9a7a361ab9-Paper.pdf.

Z. Zhang, Y. Sheng, T. Zhou, et al. H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 34661–34710. Curran Associates, Inc., 2023. doi: 10.52202/075280-1506. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/6ceefa7b15572587b78ecfcebb2827f8-Paper-Conference.pdf.

L. Zheng, L. Yin, Z. Xie, et al. SGLang: Efficient Execution of Structured Language Model Programs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583. Curran Associates, Inc., 2024. doi: 10.52202/079017-2000.

Y. Zhong, S. Liu, J. Chen, et al. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 193–210. USENIX Association, 2024.

APPENDIX CONTENTS

A Execution and Geometry Proofs **13**
A.1 Equality in Real Arithmetic 13
A.2 Minimality 14
A.3 Row Recurrence 14
A.4 Suffix Lengths 15
A.5 Sliced Layers 15
A.6 Complete Analytical Grid 15
B Operator Supports and Mixed Stacks **16**
B.1 Mixer and State Supports 16
B.2 Global Layers 17
B.3 Evaluated Models 17
C Experimental Protocols **17**
C.1 Synthetic Model and Checks 18
C.2 Timing, Memory, and Statistics 18
C.3 Mistral with FlashAttention-2 19
C.4 Additional Backends and Hybrid Models 19
D Additional Results **19**
D.1 Complete Synthetic Grid 19
D.2 Complete FA2 Timing Results 20
D.3 Mistral with Tiled SDPA 21
D.4 Gemma-3-4B and LFM2-2.6B 22
E Proofs for Sufficient Continuation State **22**
E.1 Fixed Keys: Exact Rank and Distinguishing Continuations 23
E.2 Local Stacks: Forgetting and Input Reconstruction 24
E.3 Verifiable Conditions for Normalized Attention 25
F Extended Related Work **28**

A EXECUTION AND GEOMETRY PROOFS

This appendix proves that the retained computations preserve the requested interface, that no further original operations can be deleted, and that the row recurrence and local row counts describe exactly these dependencies.

Setting and assumptions. We use the static graph $G = (\mathcal{V}, \mathcal{E})$ and the targets $\Omega = \mathcal{O} \cup \mathcal{C}$ of Section 2, and abbreviate $S = \mathrm{Cone}(\Omega) = \Omega \cup \mathrm{Anc}_G(\Omega)$. Fix a legal prompt and the model parameters. For a node v, $\mathrm{Pred}_G(v)$ denotes its immediate predecessors, and z_v and $\widehat{z}_v$ denote its values under dense prefill and under C3P. A node without predecessors is an external input, such as a prompt token or a parameter tensor. Every other node v applies its original deterministic operator F_v to the values of its predecessors, taken in their original order. Both executions use the same external inputs, parameters, masks, absolute positions, and cache conventions. The graph contains every input of every operator and every entry of the continuation state. C3P changes only which nodes are evaluated, and all equalities below hold in real arithmetic.

A.1 EQUALITY IN REAL ARITHMETIC

Proof of Theorem 1. We first show that every retained node keeps its value, and then that equal states give equal continuations.

Step 1: Retained operators keep all of their inputs. Let $v \in S$ and $u \in \mathrm{Pred}_G(v)$. If v is a target, the edge $u \to v$ makes u an ancestor of that target. Otherwise v has a path to some target, and prefixing the edge $u \to v$ gives a path from u to the same target. In both cases $u \in S$, so

$$\mathrm{Pred}_G(v) \subseteq S \qquad \text{for every } v \in S.$$

Skipping the nodes outside S therefore removes no input of a retained operator and never calls for a replacement value.

Step 2: Induction over the retained nodes. Since G is acyclic, restricting a topological order of G to S lists the retained nodes as $v_1, \ldots, v_{|S|}$, each after all of its predecessors. An external input has

the same value in both executions. For any other node v_j, Step 1 shows that all of its predecessors are retained, and the order places them before v_j. If they agree, then

$$\widehat{z}_{v_j} = F_{v_j}\big((\widehat{z}_u)_{u\in\mathrm{Pred}_G(v_j)}\big) = F_{v_j}\big((z_u)_{u\in\mathrm{Pred}_G(v_j)}\big) = z_{v_j},$$

because both sides apply the same operator with the same parameters, mask, and position convention to the same tuple of values. Induction gives $\widehat{z}_v = z_v$ for every $v \in S$, in particular for every requested output in $\mathcal{O}$ and every entry of the continuation state $\mathcal{C}$.

Step 3: Induction over a continuation. Let c_0 and $\widehat{c}_0$ be the continuation states after dense prefill and after C3P, meaning the values and metadata that later steps read, not unused memory. Step 2 gives $\widehat{c}_0 = c_0$. For any legal tokens $x_1, \ldots, x_H$, both executions apply the same deterministic decoding step

$$(y_t, c_t) = D_\theta(c_{t-1}, x_t), \qquad (\widehat{y}_t, \widehat{c}_t) = D_\theta(\widehat{c}_{t-1}, x_t),$$

with fixed parameters θ and output y_t at step t. If $\widehat{c}_{t-1} = c_{t-1}$, the two steps receive identical arguments, so $\widehat{y}_t = y_t$ and $\widehat{c}_t = c_t$. Induction from $\widehat{c}_0 = c_0$ covers every step up to the horizon H.

Equal logits yield equal tokens under any deterministic selection rule. Under random sampling, sharing the random draws yields the same tokens, and without shared draws the output distributions agree. □

A.2 Minimality

Proof of Theorem 2. Let $S' \subseteq \mathcal{V}$ be the retained set of an admissible deletion-only schedule. Admissibility requires producing every target in Ω and evaluating each retained node with its original operator on its original predecessor values. A skipped value cannot be replaced by a constant, an algebraic rewrite, or a different state encoding. We show $S \subseteq S'$ and then uniqueness.

Step 1: Every target is retained. The schedule must produce every requested output and every original state entry, so $\Omega \subseteq S'$. Omitting a target violates this requirement even if another node happens to hold the same value.

Step 2: Every ancestor of a target is retained. Take $v \in S \setminus \Omega$ and a path

$$v = v_0 \longrightarrow v_1 \longrightarrow \cdots \longrightarrow v_J = \omega, \qquad \omega \in \Omega.$$

Suppose that $v \notin S'$. Since $\omega \in S'$ by Step 1, the path contains an edge $v_{j-1} \to v_j$ with $v_{j-1} \notin S'$ and $v_j \in S'$. The retained operator at v_j then lacks one of its original inputs, which admissibility forbids. Hence $v \in S'$, and together with Step 1 this gives $S \subseteq S'$.

Step 3: The cone is the unique minimal set. By Theorem 1, retaining exactly S is admissible. Every admissible set contains S, so any admissible set other than S strictly contains S and is not inclusion-minimal. Hence S is the unique inclusion-minimal retained set. □

A.3 Row Recurrence

Layer interface. Write $\mathbf{h}_{\ell-1,j}$ for input row j of layer ℓ. The targets of layer ℓ are its output rows in $\mathcal{R}_\ell$ and its entries of the continuation state. The sets $\mathcal{A}_\ell(i)$ and $\mathcal{B}_\ell$ are the exact static row supports of the mixer and of the state construction at the chosen graph granularity. All other operations of the layer are position-wise. They may add intermediate nodes but no further input rows.

Lemma 1. *Under this interface, Equation (2) returns exactly the input rows of layer ℓ that belong to the cache-complete cone.*

Proof. Let $I_{\ell-1}$ be the input rows that have a path to a target within layer ℓ, and let

$$U_{\ell-1} = \mathcal{R}_\ell \cup \mathcal{B}_\ell \cup \bigcup_{i\in\mathcal{R}_\ell} \mathcal{A}_\ell(i).$$

We prove both inclusions.

Step 1: $I_{\ell-1} \subseteq U_{\ell-1}$. A target output $i \in \mathcal{R}_\ell$ reads its residual input $\mathbf{h}_{\ell-1,i}$ and its mixer inputs in $\mathcal{A}_\ell(i)$, and position-wise operations before or after the mixer read no further rows. The state construction reads exactly the rows in $\mathcal{B}_\ell$. These are all the input rows that the targets depend on.

Step 2: $U_{\ell-1} \subseteq I_{\ell-1}$. A row in $\mathcal{R}_\ell$ reaches its own output through the residual path. A row in $\mathcal{A}_\ell(i)$ with $i \in \mathcal{R}_\ell$ reaches output i through the mixer. A row in $\mathcal{B}_\ell$ reaches a state entry through the state construction. These are graph paths rather than single edges, since normalization or projection nodes may lie between an input row and its target.

Step 3: Identify the recurrence. The two inclusions give $I_{\ell-1} = U_{\ell-1} = \mathcal{R}_\ell \cup \mathcal{P}_\ell$. These are exactly the outputs required from layer $\ell - 1$, which is Equation (2). A position-wise output head requesting only row n starts the recurrence at $\mathcal{R}_L = \{n\}$. Applying the argument layer by layer, with each layer's state targets added, gives the row projection of the whole cone. □

Full-output and K/V work. The rows in $\mathcal{P}_\ell$ need normalization and the key and value projections. Queries, attention outputs, output projections, and the feed-forward block are needed only on $\mathcal{R}_\ell$. Computing K/V rows at every prompt position therefore does not require full-output work at every position in the same layer.

A.4 Suffix Lengths

Suppose that $\mathcal{R}_\ell = \{n - q_\ell + 1, \ldots, n\}$ with $1 \le q_\ell \le n$, that each mixer support is the clipped contiguous window $\mathcal{A}_\ell(i) = \{\max(1, i - \rho_\ell), \ldots, i\}$, and that $\mathcal{B}_\ell$ is the suffix of length $0 \le b_\ell \le n$, empty when $b_\ell = 0$. Then

$$\bigcup_{i \in \mathcal{R}_\ell} \mathcal{A}_\ell(i) = \{\max(1, n - q_\ell + 1 - \rho_\ell), \ldots, n\},$$

a suffix of length $\min\{n, q_\ell + \rho_\ell\}$ that contains $\mathcal{R}_\ell$. The union with $\mathcal{B}_\ell$ is the longer of the two suffixes, which gives Equation (4). If $b_\ell \le q_\ell + \rho_\ell$ in every layer, the recurrence reduces to $q_{\ell-1} = \min\{n, q_\ell + \rho_\ell\}$, and iterating from $q_L = 1$ gives Equation (5). Otherwise the longer buffer propagates through the maximum.

A small example shows why the buffer condition matters. With $n \ge 5$, $q_\ell = 1$, and $\rho_\ell = 2$, the mixer needs three input rows. A buffer of $b_\ell = 3$ rows adds nothing, whereas $b_\ell = 5$ extends the required suffix to five rows. The buffer length, and not only the look-back radius, therefore enters the schedule.

A.5 Sliced Layers

If $n = 1$, every layer has one full-output row and $L_{\text{slice}} = 0$. If $n > 1$ and $\rho = 0$, every layer needs only the last row, so $L_{\text{slice}} = L$. For $n > 1$ and $\rho > 0$, under the conditions of Equation (6), let $k = L - \ell$ count the backward steps from the top layer. A layer has fewer than n full-output rows exactly when $1 + k\rho < n$, that is, when $k < (n-1)/\rho$. The nonnegative integers satisfying this inequality are $0, \ldots, \lceil (n-1)/\rho \rceil - 1$, and at most L of them correspond to layers. This gives Equation (7). For $n > 1$, FLS slices the last layer only, so C3P slices $L_{\text{slice}} - 1$ additional layers.

A.6 Complete Analytical Grid

Figure 4 evaluates the logical work of Equation (9) on a finer grid of depths and windows than the measurements.

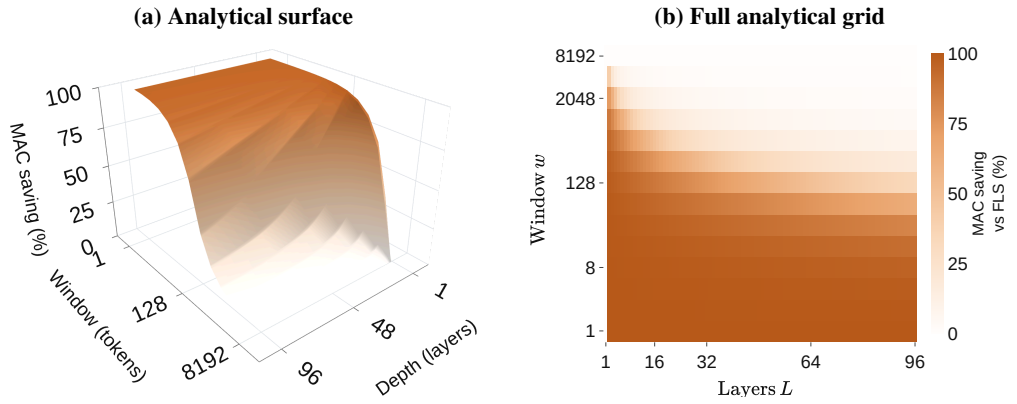

Figure 4: **Depth and window size determine the logical saving over FLS.** FLS denotes final-layer slicing. Both panels show the same 1,344 configurations, $L = 1, \ldots, 96$ and $w \in \{2^0, 2^1, \ldots, 2^{13}\}$, with logarithmic window axes. Each value is $100(1 - M^{\mathrm{C3P}}/M^{\mathrm{FLS}})$, where M sums Equation (9) over layers and includes the output head. We fix $n = 8192$, $d = 512$, $d_{\mathrm{kv}} = 128$, $d_{\mathrm{ff}} = 2048$, and $n_{\mathrm{voc}} = 512$, and every schedule keeps the same rolling buffer cache of $\min(n, w - 1)$ rows per layer.

B Operator Supports and Mixed Stacks

The suffix formulas in Section 3 describe local attention. Real decoders can also contain global attention and causal convolutions. We give the input positions and stored values that these operators require, then apply the same dependency recurrence to the evaluated architectures.

B.1 Mixer and State Supports

The mixer support is the set of input positions read by an output row; the state support is the set needed to construct values read by later decoding. Table 3 lists both, with every interval intersected with the prompt positions $\{1, \ldots, n\}$. A physical buffer can allocate additional capacity, but only entries read by a legal future step contribute to the state support.

Table 3: **Mixer supports and state supports play separate roles.** The window size w_ℓ includes the current position. For a convolution, w_ℓ is the kernel width. A suffix buffer is sized by what later decoding steps actually read.

Operator	Mixer support $\mathcal{A}_\ell(i)$	State support $\mathcal{B}_\ell$
Causal local attention	$[i - w_\ell + 1, i]$	Suffix of keys and values read by later steps
Causal global attention	$[1, i]$	All prompt keys and values
Causal convolution	$[i - w_\ell + 1, i]$	Native buffer of recent projected inputs
Position-wise operation	$\{i\}$	Empty unless its interface stores state

Position-wise preprocessing remains necessary on every projection path that survives. RMSNorm or LayerNorm over the features of one row does not mix positions, so it can be evaluated on the required rows only. The feed-forward block and the residual update are needed only for rows in $\mathcal{R}_\ell$. A K/V row still needs the normalization and projections that produce its original keys and values.

Grouped-query attention changes the number of key/value heads, not the positions that a query reads (Ainslie et al., 2023). The executor keeps the original mapping from query heads to key/value heads, and d_{kv} denotes the actual total key/value width. Rotary position embeddings keep their original absolute positions (Su et al., 2021). Storing a suffix compactly in memory must not relabel its positions, change a position-dependent scaling, or alter the causal mask. The schedule selects existing rows and preserves their identities.

A causal convolution needs both its receptive field and the stored projected inputs read by incremental decoding. For kernel width three, output i reads positions $i - 2$, $i - 1$, and i. An implementation may allocate a three-row buffer even though its look-back radius is two. The state support follows the buffer's read and update order, rather than its allocated capacity. With one required output row, the

three-row mixer expansion already covers either a two-row or a three-row suffix. A larger readable buffer would propagate through the maximum in Equation (4).

B.2 Global Layers

A global layer needs every prompt key and value, but it can still restrict its own full-output work. Let ℓ_g be the last global-attention layer, with only contiguous local mixers above it whose buffers satisfy $b_\ell \le q_\ell + \rho_\ell$. The required output suffix of layer ℓ_g then has length

$$q_{\ell_g} = \min\Big\{n,\, 1 + \sum_{\ell'=\ell_g+1}^{L} \rho_{\ell'}\Big\}. \tag{12}$$

Because layer ℓ_g caches every prompt position, $\mathcal{P}_{\ell_g} = \{1, \dots, n\}$, and Equation (2) gives $\mathcal{R}_{\ell_g - 1} = \{1, \dots, n\}$. Every layer below ℓ_g is therefore computed in full, while the full-output work of layer ℓ_g is restricted to $\mathcal{R}_{\ell_g}$.

Here the continuation state does not enlarge the arithmetic, because the last query of layer ℓ_g already reads every prompt key and value. Execution installs the full original cache of layer ℓ_g for subsequent decoding.

B.3 Evaluated Models

Table 4 lists the evaluated models and the architecture example from the introduction. Window sizes include the current query position; the support sets follow the causal mask and the values read by incremental decoding.

Table 4: **Architectures with local operators and their role in this paper.** Layers are numbered from one throughout the paper.

Model	Architecture	Role here
Mistral-7B	32 sliding-window attention layers, window size 4,096 (Jiang et al., 2023)	BF16 prefill measurements
Gemma-3-4B	34 layers, five local layers per global layer, local window size 1,024 (Gemma Team, 2025)	BF16 prefill measurements
LFM2-2.6B	30 layers mixing full attention and width-three causal convolutions (Amini et al., 2025)	BF16 prefill measurements
gpt-oss	Alternating banded and dense attention (OpenAI, 2025)	Architecture example

The evaluated Gemma-3-4B configuration has eight query heads, four key/value heads, and head dimension 256. Its last global layer is layer 30 and is followed by four local layers. C3P can therefore restrict the full-output work of layers 30 to 34, five of the 34 layers, while the full cache of layer 30 keeps every earlier layer dense.

The evaluated LFM2-2.6B configuration has full-attention layers 3, 6, 10, 14, 18, 22, 25, and 28. Its final two layers are causal convolutions of width three with native buffers allocated for three projected-input rows. Tracing back from the output gives one row at the top, three rows at the interface below it, and five full-output rows at the last attention layer, whose full cache keeps every earlier layer dense. C3P therefore restricts three of the 30 layers, and the same recurrence handles the attention and convolution states together.

Mistral-7B has no global layer, but its 4,096-token window covers the prompt after a few backward steps. Appendix D.3 derives the resulting sliced-layer counts and logical savings for the tiled-SDPA measurements. Section 4.3 evaluates the schedule on FlashAttention-2.

C Experimental Protocols

The protocols below specify the synthetic model, the timing and memory boundaries, and the inputs and aggregation used for each pretrained-model comparison.

C.1 Synthetic Model and Checks

The synthetic model implements exactly the operators of the local analysis. Every layer has pre-normalization, rotary positions, grouped-query causal local attention, and a gated SiLU feed-forward block. Dense prefill, FLS, and C3P share the weights, masks, positions, query tiling, and rolling buffer cache, and the planner changes only their row sets. Their common executor calls PyTorch SDPA with explicit Boolean local masks and repeated key/value heads, using 128-row query tiles on the GPU.

The GPU grid fixes $n = 8192$, $d = 512$, $d_{\mathrm{kv}} = 128$, $d_{\mathrm{ff}} = 2048$, $n_{\mathrm{voc}} = 512$, eight query heads, and two key/value heads. Each of the 25 configurations of Section 4.1 uses three independent random initializations, giving 75 cases in total. The context is long enough to contain the prompt and eight fixed continuation inputs.

An independent CPU oracle checks the row schedule without calling the planner or the tiled attention. It evaluates dense attention with explicit contractions and softmax and recomputes the whole prefix for each continuation input. Eight small configurations include the example of Figure 1 ($n = 12$, $L = 9$, $w = 3$), a one-token prompt, a single layer, self-only attention, and windows at or beyond the prompt length. Two random initializations, three schedules, and two precisions give 96 combinations, each checked at the prompt and at four fixed continuation inputs (Table 5).

Table 5: **Observed errors in the independent CPU checks.** Each precision covers 48 combinations of schedule, configuration, and seed. Cache errors include keys and values at every layer.

Precision	Max logit absolute error	Max K/V absolute error	Absolute / relative tolerance
FP64	6.66×10^{-16}	8.88×10^{-16}	10^{-10} / 10^{-9}
FP32	4.17×10^{-7}	5.96×10^{-7}	2×10^{-5} / 2×10^{-4}

The CPU checks also verify shapes, cache positions, and finite values. Negative controls reject corrupted keys, wrong cache positions, and non-finite logits, and a horizon check rejects a continuation beyond the declared context. Separately, explicit enumeration of the row graph covers 1,764 local, global, and mixed configurations, and fixed-order FP64 execution checks 108 instances.

On the GPU, the FP32 checks compare FLS and C3P with dense prefill on the same inputs before timing, at the prompt and at eight fixed continuation inputs. This gives 1,350 output comparisons and 33,480 per-layer cache comparisons, all passing absolute and relative tolerances of 2×10^{-5} and 2×10^{-4}. The largest C3P logit and key/value errors are 1.90735×10^{-6} and 3.81470×10^{-6}.

C.2 Timing, Memory, and Statistics

The synthetic FP32 measurements use A100-SXM4-40GB GPUs with TF32 disabled. PyTorch SDPA chooses its backend automatically for the explicit local mask. All schedules for a given case run on the same device.

Every timed call starts and ends with device synchronization. After five warm-up calls per schedule, each synthetic case receives 20 repetitions of all three schedules in random order. The timed region includes planning, embeddings, projections, tiled attention, cache cloning and installation, and the final-row readout, and it excludes model loading, tokenization, validation, and later decoding. The Mistral timings use the same boundary.

Paired ratios give every repetition and seed a defined role. Let $T^{\mathrm{FLS}}_{s,r}$ and $T^{\mathrm{C3P}}_{s,r}$ be the FLS and C3P latencies of repetition r for seed s in one configuration. The reported speedup is

$$\gamma_s = \exp\Big(\frac{1}{20}\sum_{r=1}^{20} \log \frac{T^{\mathrm{FLS}}_{s,r}}{T^{\mathrm{C3P}}_{s,r}}\Big), \qquad \gamma = \exp\Big(\frac{1}{3}\sum_{s=1}^{3} \log \gamma_s\Big). \tag{13}$$

For Mistral, a text fragment takes the role of a seed. For the tiled-SDPA measurements, reported Mistral latencies are the median over fragments of each fragment's median time. The FA2 measurements report arithmetic mean latencies and retain the same paired geometric speedup statistic.

Peak memory uses the same maximum-based rule for both synthetic precisions. For each schedule and seed, the recorded peak is the maximum over repetitions, and a configuration uses the maximum

across seeds:

$$100\Big(1 - \frac{\max_s m_s^{\mathrm{C3P}}}{\max_s m_s^{\mathrm{FLS}}}\Big), \tag{14}$$

where m_s is the peak number of bytes allocated through the PyTorch allocator. All synthetic schedules hold the same parameters and cache payload, so the saving comes from execution allocations.

C.3 Mistral with FlashAttention-2

The FA2 timing measurements use Mistral-7B, BF16, batch one, and an A100 40GB. Prompt lengths are 16,384 and 32,512, each using the same three fixed text fragments, labeled Prompts 1–3. The four methods are HF–FA2*, Dense, FLS, and C3P, with the common window convention specified below. Five warm-up calls precede 20 paired repetitions per method and text. There are $2 \times 3 \times 4 \times 20 = 480$ timed calls. All methods produce last-position logits and an installed cache, and their timing includes the padding and handoff needed to complete that interface. The aggregate uses all paired repetitions.

Window and query alignment. HF–FA2* uses the same local causal support $\{\max(1, i - w + 1), \ldots, i\}$ at absolute query position i, with $w = 4096$. In the shared executor, absolute-position alignment places queries into 128-row blocks. Any added query rows have discarded outputs, and their computation is included in timing. In particular, the final FLS and C3P layers each evaluate one requested query together with 127 dummy queries. The window convention and query padding are included in the comparator definitions and timing.

Intervals and comparators. Mean latency is the arithmetic mean of the recorded times. Speedup is the geometric mean of paired baseline/C3P ratios, with equal weight for each of the three texts. The reported 95% intervals use 10,000 paired bootstrap resamples within each text and condition on those fixed texts. The fastest baseline is selected from the three comparators defined above by minimum geometric mean latency, and this selection is repeated inside every bootstrap resample. FLS is the fastest in each measured case and in both length aggregates. Tables 7 and 8 report the per-prompt results and aggregates.

C.4 Additional Backends and Hybrid Models

The tiled-SDPA measurements evaluate BF16 `mistralai/Mistral-7B-v0.1`, with prompts of 8,192, 16,384, and 32,736 tokens drawn from three fixed WikiText validation fragments at each length (Merity et al., 2017). Each of the nine cases has 20 paired repetitions of matched HF, dense, FLS, and C3P.

Gemma uses the instruction-tuned `unsloth/gemma-3-4b-it` checkpoint, with three fixed text fragments at each prompt length of 32K, 64K, and 128K tokens. These lengths use 3, 2, and 1 warm-up calls and 20, 10, and 6 paired repetitions per fragment, respectively. Its tiled SDPA implementation uses 1,024-row query tiles, and its comparator, native Hugging Face SDPA, is the fastest measured baseline for every fragment. LFM2 uses `LiquidAI/LFM2-2.6B`, with three fixed text fragments at each prompt length of 8K, 16K, and 32K tokens. Each case has three warm-up calls and ten paired repetitions of native SDPA, FLS, and C3P. Both models use BF16 on A100 GPUs with TF32 disabled, synchronize the device around timed regions, and draw text from the WikiText-2 raw validation split. For these two models we report median speedup estimates and the maximum peak-memory ratio across the three fragments.

D Additional Results

The complete results follow the experimental questions in Section 4: the synthetic geometry, optimized-kernel gains, and the effect of mixed architectures. We also report the Mistral comparison using tiled SDPA.

D.1 Complete Synthetic Grid

The BF16 measurements repeat every configuration and seed of the FP32 grid with the same three schedules and 20 paired repetitions, 4,500 timings in total, aggregated as in Appendix C.2. Its largest speedup is $9.886\times$ at $L = 8$, $w = 32$, and its largest memory saving is 78.53% at $L = 4$, $w = 32$,

so BF16 keeps the pattern of the FP32 grid. Table 6 places both precisions next to the logical MAC saving and the GPU energy saving.

GPU energy. An independent FP32 experiment measures energy on the same 25 configurations and three seeds, with three batches for each schedule and seed. A batch reads the cumulative NVML energy counter of the GPU before and after its prefill calls and divides the difference by the number of completed calls. The three batches of a schedule and seed are averaged to give e_s, and Table 6 reports

$$100\Big[1-\exp\Big(\frac{1}{3}\sum_{s=1}^{3}\log\frac{e_s^{\mathrm{C3P}}}{e_s^{\mathrm{FLS}}}\Big)\Big]. \tag{15}$$

The energy saving reaches 94.14% at $L=4$, $w=32$ and follows the logical saving across the grid.

Table 6: **Complete synthetic performance grid.** All savings and speedups are relative to FLS at the same configuration, with $n=8192$. The MAC saving is analytical. Speedup, peak allocated-memory saving, and GPU energy saving are measured, the energy in the independent FP32 experiment. Full-window speed ratios keep four decimals.

L	w	MACs saved (%)	Speedup (×) FP32	Speedup (×) BF16	Peak memory saved FP32 (%)	Peak memory saved BF16 (%)	Energy saved FP32 (%)
4	32	99.21	10.018	8.863	80.30	78.53	94.14
4	128	96.81	8.977	7.851	77.06	75.13	91.97
4	512	87.22	5.533	4.653	65.13	63.74	83.93
4	2048	47.01	1.769	1.785	18.38	17.99	42.70
4	8192	0.00	1.0002	0.9999	0.00	0.00	1.85
8	32	98.46	10.647	9.886	68.10	66.92	93.95
8	128	93.72	7.783	7.184	62.30	61.21	89.25
8	512	74.73	3.533	3.189	39.27	38.54	72.57
8	2048	20.18	1.227	1.219	1.37	1.35	19.84
8	8192	0.00	0.9999	0.9997	0.00	0.00	-2.02
16	32	96.95	9.774	9.254	51.83	51.08	92.44
16	128	87.53	5.402	5.205	42.16	41.46	83.92
16	512	49.70	1.904	1.832	4.29	4.23	46.16
16	2048	9.42	1.095	1.094	1.06	1.04	9.52
16	8192	0.00	1.0000	1.0001	0.00	0.00	0.78
32	32	93.92	7.729	7.465	33.81	33.62	89.81
32	128	75.12	3.341	3.281	20.12	19.45	71.85
32	512	24.05	1.299	1.284	0.95	0.95	23.38
32	2048	4.56	1.044	1.046	0.72	0.71	3.40
32	8192	0.00	1.0000	1.0000	0.00	0.00	0.14
64	32	87.86	5.412	5.253	18.10	18.01	83.38
64	128	50.31	1.884	1.870	1.04	0.66	45.99
64	512	11.84	1.128	1.136	0.59	0.60	12.09
64	2048	2.24	1.021	1.020	0.44	0.44	2.16
64	8192	0.00	0.9999	1.0000	0.00	0.00	0.06

D.2 Complete FA2 Timing Results

Tables 7 and 8 give the full latency summaries and comparator-specific gains behind Figure 3. The aggregates combine all six text–length cases. The measured speedup over FLS ranges from $1.0448\times$ to $1.0452\times$ at 16,384 tokens and from $1.1184\times$ to $1.1191\times$ at 32,512 tokens. The speedup intervals for every individual case remain above one.

Table 7: **Complete FA2 prefill latency in milliseconds.** Arithmetic means over 20 repetitions per method and text, or 60 for each length aggregate. HF–FA2* is the window-corrected Hugging Face comparator.

Prompt tokens	Prompt	HF–FA2*	Dense	FLS	C3P
16,384	1	1324.08	1312.50	1272.41	1217.40
16,384	2	1315.32	1303.70	1264.12	1209.94
16,384	3	1323.86	1315.74	1272.71	1217.64
16,384	All three	1321.09	1310.65	1269.75	1214.99
32,512	1	2629.78	2607.95	2527.97	2260.29
32,512	2	2648.81	2627.90	2546.47	2275.43
32,512	3	2631.93	2608.90	2528.87	2260.94
32,512	All three	2636.84	2614.92	2534.43	2265.55

Table 8: **C3P speedups on FA2.** Paired geometric means with 95% confidence intervals from 10,000 within-text paired bootstrap resamples. Saved latency is the difference of arithmetic means.

Prompt tokens	Comparator	Speedup	95% CI	Saved ms
16,384	HF–FA2*	$1.0873\times$	$[1.0868, 1.0878]$	106.10
16,384	Dense	$1.0787\times$	$[1.0772, 1.0811]$	95.66
16,384	FLS (fastest)	$1.0451\times$	$[1.0445, 1.0456]$	54.75
32,512	HF–FA2*	$1.1639\times$	$[1.1629, 1.1646]$	371.29
32,512	Dense	$1.1542\times$	$[1.1533, 1.1549]$	349.36
32,512	FLS (fastest)	$1.1187\times$	$[1.1179, 1.1194]$	268.88

The fastest valid comparator is reselected from the prespecified non-C3P pool within each bootstrap resample. FLS is fastest in all six measured cases and both length aggregates. The intervals condition on these three fixed texts.

D.3 Mistral with Tiled SDPA

We apply the same schedule to the pretrained Mistral-7B-v0.1 (Jiang et al., 2023), whose 32 layers all use a 4,096-token window. Prompts of 8,192, 16,384, and 32,736 tokens come from three fixed WikiText validation fragments per length (Merity et al., 2017). Four BF16 schedules each receive 20 paired repetitions per fragment, 720 timings in total. Dense, FLS, and C3P share one executor that runs the Hugging Face modules layer by layer on the selected rows with the tiled attention above. Dense denotes this shared executor without row removal. The fourth schedule, matched HF, is the Hugging Face forward pass with its attention call and final cache replaced by the same tiled attention and rolling buffer cache. FLS is again the fastest baseline, 3.1–3.5% faster than dense and matched HF.

C3P runs $1.012\times$, $1.045\times$, and $1.120\times$ faster than FLS at the three lengths, with median prefill times of 1.190, 2.504, and 4.862 seconds (Figure 5 shows every fragment). The MAC counts account for the observed gains and their growth with prompt length. With $\rho = 4095$, Equation (7) gives $L_{\text{slice}} = 3$, 5, and 8 of the 32 layers, and the sliced layers nearest the prompt start lose few rows. Evaluating Equation (9) with the Mistral dimensions $d = 4096$, $d_{\text{kv}} = 1024$, $d_{\text{ff}} = 14336$, and $n_{\text{voc}} = 32000$, C3P needs 1.50%, 4.67%, and 11.09% fewer MACs than FLS, which corresponds to MAC ratios of $1.015\times$, $1.049\times$, and $1.125\times$. Each measured speedup lies within 0.4% of this ratio, and the measured time saving amounts to 81–97% of the logical saving.

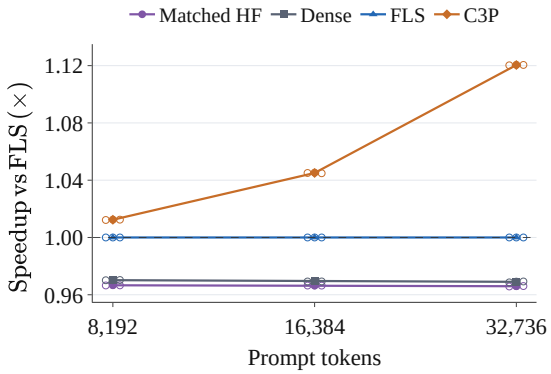

Figure 5: **Mistral-7B prefill gains grow with prompt length.** BF16 Mistral-7B-v0.1 on an A100, with speed normalized to FLS. Each length uses three text fragments and 20 paired repetitions per fragment and schedule. Lines connect geometric means of paired speed ratios across fragments, and markers show individual fragments. Dense denotes prefill in the shared executor without row removal. Matched HF denotes the Hugging Face forward pass with matched tiled attention and rolling cache; FLS denotes final-layer slicing. All four schedules use the same attention kernel and cache conventions.

D.4 Gemma-3-4B and LFM2-2.6B

Table 9 reports both hybrid models with the comparator of each protocol. Gemma's speedups and memory ratios are relative to native Hugging Face SDPA, the fastest measured Gemma baseline, and LFM2's are relative to FLS. Each memory ratio is the maximum across the three fragments.

Table 9: **BF16 prefill of the hybrid architectures.** All cases use an A100 and three text fragments per prompt length. Paired repeats are totals over the fragments. Speedups are median aggregates, and peak ratios are maxima across fragments. K denotes 1,024 tokens.

Gemma-3-4B

Prompt	Paired repeats	Speedup / native HF	Peak ratio / native HF
32K	60	1.115×	0.9975
64K	30	1.134×	0.9977
128K	18	1.152×	0.9981

LFM2-2.6B

Prompt	Paired repeats	Speedup / FLS	Speedup / native HF	Peak ratio / FLS
8K	30	1.067×	1.101×	0.9972
16K	30	1.073×	1.107×	0.9950
32K	30	1.081×	1.113×	0.9918

E Proofs for Sufficient Continuation State

This appendix proves the results of Section 5. All equalities use real arithmetic. An observation is the complete logit vector after each externally supplied continuation token. Positions, prompt length, model parameters, and the legal remaining horizon H are fixed. The legal input set is shared by compared states and closed under prefixes, and the constructions below use token strings of the stated lengths. Write $c \sim_H c'$ when every legal continuation of length at most H has the same logit trace from the two states.

A sufficient initial encoding may change the representation and permit arbitrary later recomputation. It includes every initially prompt-dependent resource available during decoding, such as a retained prompt or reusable prompt-boundary logits. Dimension statements concern continuous encoders on a stated open domain, and bit counts concern arbitrary encodings of reachable states.

In the one-layer results, b is the number of cached rows, J is the number of distinct possible queries, and r is the rank of the linear logit readout. For a stack, d is the hidden width, ρ_ℓ is the look-back

radius of layer ℓ, $\Delta_\ell = \sum_{i=1}^{\ell} \rho_i$, $\Delta = \Delta_L$, and $s = \min(n, \Delta)$. Norms are Euclidean or their induced matrix operator norms.

E.1 Fixed Keys: Exact Rank and Distinguishing Continuations

Model and known contributions. Consider one causal softmax-attention layer with window $b+1$ including the current row and an initially full cache of $b \geq 1$ rows. The initial keys $\mathbf{k}_1, \ldots, \mathbf{k}_b \in \mathbb{R}^{d_k}$ are fixed and known, and only the values $\mathbf{v}_j \in \mathbb{R}^{d_v}$ vary. Rows are indexed oldest first. A token has a cache-independent, position-independent query in $\{\mathbf{q}_1, \ldots, \mathbf{q}_J\}$, and every query is realized by a legal token at every step. Every token sequence through the horizon H is legal. Appended keys and values are known functions of the supplied token and position. The logits are

$$\mathbf{z}_t = \mathbf{g}_t(u) + W_{\text{logit}} \mathbf{o}_t, \qquad W_{\text{logit}} : \mathbb{R}^{d_v} \to \mathbb{R}^{n_{\text{voc}}}, \qquad \operatorname{rank} W_{\text{logit}} = r, \tag{16}$$

where $\mathbf{o}_t$ is the attention output and $\mathbf{g}_t(u)$ is known from the continuation. Attention's usual scale is absorbed into $\mathbf{q}_a$.

At step $t \leq b$, the surviving initial rows are exactly $t, \ldots, b$. For a given continuation ending in query $\mathbf{q}_a$, let $D_{t,a} > 0$ be the softmax denominator and $\mathbf{c}_{t,a}$ the projected numerator contribution of appended tokens. Both are known independently of the initial values. Therefore

$$D_{t,a}(\mathbf{z}_t - \mathbf{g}_t) - \mathbf{c}_{t,a} = \mathbf{y}_{t,a} := \sum_{j=t}^{b} e^{\mathbf{q}_a^\top \mathbf{k}_j} W_{\text{logit}} \mathbf{v}_j. \tag{17}$$

Different prefixes may have different known denominators, which does not affect the recovery of $\mathbf{y}_{t,a}$. Every pair (t, a) is realized by a fixed legal prefix of length $t-1$ followed by a query-$\mathbf{q}_a$ token. At $t > b$, no initial value remains.

Proof of Theorem 3. Let $h = \min(H, b)$, $w_{aj} = e^{\mathbf{q}_a^\top \mathbf{k}_j}$, and define

$$\Phi_{(t,a),j} = 1\{j \geq t\} w_{aj}, \qquad W_h = [w_{aj}]_{1 \leq a \leq J,\ h \leq j \leq b}.$$

Equation (17) shows that H-equivalence is exactly equality of the linear observations with coefficient rows Φ. For $t < h$,

$$\Phi_{(t,a),\cdot} - \Phi_{(t+1,a),\cdot} = w_{at} \mathbf{e}_t^\top.$$

Since $w_{at} > 0$, the first $h-1$ coordinate rows belong to the row space. Removing those coordinates leaves precisely the row space of W_h, supported on columns $h, \ldots, b$. The supports are disjoint, so

$$\operatorname{rank} \Phi = h - 1 + \operatorname{rank} W_h, \qquad d_H = r\big(h - 1 + \operatorname{rank} W_h\big). \tag{18}$$

The full observation matrix is $\Phi \otimes W_{\text{logit}}$, up to coordinate ordering, and therefore has rank d_H. A row-space basis gives a sufficient linear encoding in d_H real coordinates.

For necessity, intersect the open value-cache domain with a small affine slice complementary to the observation kernel. This slice has dimension d_H, and the observation map is injective on it. Any sufficient encoder must also be injective on this slice. A continuous injection from an open subset of $\mathbb{R}^{d_H}$ into $\mathbb{R}^m$, $m < d_H$, is impossible: composing it with the coordinate inclusion into $\mathbb{R}^{d_H}$ contradicts invariance of domain, since the image would be open and contained in a proper linear subspace. Only continuity of the encoder enters this dimension argument.

□

A closed form for exponentially weighted queries. Suppose $\mathbf{q}_a = \tau_a \boldsymbol{\xi}$ with distinct real τ_a, and the projected keys $\lambda_j = \boldsymbol{\xi}^\top \mathbf{k}_j$ are distinct. A nonzero sum of N exponentials $\sum_{j=1}^{N} c_j e^{\lambda_j \tau}$ has at most $N-1$ distinct real zeros. To prove this, discard zero coefficients, divide by the exponential with smallest exponent, and differentiate. Rolle's theorem and induction apply to the remaining at most $N-1$ terms, with a nonzero constant as the base case. Thus every square submatrix of $[e^{\tau_a \lambda_j}]$ with distinct rows and columns is nonsingular. Consequently

$$\operatorname{rank} W_h = \min(J, b - h + 1), \qquad d_H = r \min\{b, H + J - 1\}.$$

Equation (18) itself holds without the distinctness condition.

An optimal encoding and continuations that recover it. Set $\bar{\mathbf{v}}_j = W_{\text{logit}}\mathbf{v}_j$, express each vector in r coordinates on $\operatorname{im} W_{\text{logit}}$, and let I_h index a row basis of W_h. An optimal sufficient encoding is

$$E_H(\mathbf{v}_1, \ldots, \mathbf{v}_b) = (\bar{\mathbf{v}}_1, \ldots, \bar{\mathbf{v}}_{h-1}, (\mathbf{s}_a)_{a \in I_h}), \qquad \mathbf{s}_a = \sum_{j=h}^{b} w_{aj}\bar{\mathbf{v}}_j. \tag{19}$$

All other $\mathbf{s}_a$ follow by known linear combinations. At most J token sequences of length h identify it: choose a token a_0 and use $a_0^{h-1}a$ for each query-realizing token a. Last-step outputs recover the tail sums. The shared-prefix outputs together with the final output for a_0 give $\mathbf{y}_{t,a_0}$ for $t = 1, \ldots, h$, and adjacent differences recover the individual $\bar{\mathbf{v}}_t$. In fact, $\operatorname{rank} W_h$ sequences suffice: choose a_0 from a tail row basis and use only that basis's final tokens. When $h = 1$, this is simply a row-basis measurement. Each sequence is a separate continuation from the same initial state.

Uniform attention. If every query is zero, Equation (19) becomes

$$\left(W_{\text{logit}}\mathbf{v}_1, \ldots, W_{\text{logit}}\mathbf{v}_{h-1}, W_{\text{logit}}\sum_{j=h}^{b}\mathbf{v}_j\right), \qquad d_H = r\min(H, b). \tag{20}$$

For $H < b$, adding a vector to $\mathbf{v}_h$ and subtracting it from $\mathbf{v}_{h+1}$ preserves all required observations. Attention reads both rows, so their redundancy lies in the continuation behavior and not in the absence of a graph path.

E.2 Local Stacks: Forgetting and Input Reconstruction

Consider a fixed-width local stack

$$\mathbf{h}_{0,t} = \mathbf{x}_t, \qquad \mathbf{h}_{\ell,t} = F_{\ell,t}(\mathbf{h}_{\ell-1,t-\rho_\ell}, \ldots, \mathbf{h}_{\ell-1,t}), \qquad \mathbf{z}_t = G_t(\mathbf{h}_{L,t}). \tag{21}$$

The actual causal mask applies at the prompt start. Position dependence is allowed, but supports and metadata are fixed, and the stack has no global layer or other recurrent state. For continuous-state claims, layer maps and the native-cache construction are continuous. A full-window map is *injective in its oldest input* when fixing all newer inputs leaves an injective function of the first argument. Radius-zero maps must be injective in their sole argument. The condition is needed only for full windows, because the reconstruction below never passes through a window truncated by the prompt start.

Lemma 2 (Finite forgetting and suffix sufficiency)**.** *With minimal rolling caches retaining at most ρ_ℓ previous rows, two initialized states with the same metadata and appended tokens have identical caches through layer ℓ after step $t \geq \Delta_\ell$. Their layer-ℓ outputs agree for $t > \Delta_\ell$. Thus equivalence through a legal horizon H is determined by continuations of length at most $\min(H, \Delta)$. For reachable states from a length-n prompt, both the initial cache and future outputs depend only on the last $s = \min(n, \Delta)$ input rows.*

Proof. Layer-0 rows are fixed by the supplied tokens. Layer 1 has expired its original cache after ρ_1 steps. Inductively, at $t \geq \Delta_\ell$ the newly installed layer-ℓ rows originate at lower-layer positions

$$t - \rho_\ell + 1, \ldots, t,$$

all strictly greater than $\Delta_{\ell-1}$, so they agree. For the current output, the oldest read position is $t - \rho_\ell$, which exceeds $\Delta_{\ell-1}$ when $t > \Delta_\ell$. Radius-zero caches are empty and their rowwise maps preserve equality. After step Δ, the complete states coincide, and later outputs add no distinction. Output Δ can still differ even though the post-step caches agree.

For reachable states, layer-ℓ cache rows at the prompt boundary originate at positions $n-\rho_\ell+1, \ldots, n$. Their earliest input ancestor is $n - \Delta_\ell + 1 \geq n - \Delta + 1$. A future output's earliest prompt ancestor is also no earlier than $n + 1 - \Delta$. Intersecting with the sequence domain gives exactly the stated sufficient suffix. Recomputation uses the original absolute positions. □

Lemma 3 (The maximal-delay path)**.** *If every full-window map in Equation (21) is injective in its oldest input, then $\mathbf{h}_{\ell,t}$ is injective in $\mathbf{x}_{t-\Delta_\ell}$ with all newer inputs fixed, whenever that interval lies inside the sequence. If the oldest-input derivatives are square and nonsingular, the composite derivative is nonsingular.*

Proof. Only the oldest layer-$(\ell-1)$ argument, at position $t-\rho_\ell$, can depend on $\mathbf{x}_{t-\Delta_\ell}$. All other arguments have strictly later earliest ancestors. By induction that oldest argument is injective in the input of interest, and composition with $F_{\ell,t}$ proves the claim. The derivative is the product along this unique maximal-delay path. Every window on the path lies inside the sequence because its earliest input does, and zero-radius maps compose at the same position. □

Proof of Theorem 4. If $\Delta = 0$, the rolling state is empty and the theorem is immediate. Assume $\Delta > 0$, the local oldest-input condition, and an injective final readout. Appendix E.3 gives normalized-attention conditions that imply these hypotheses. Fix any one legal token sequence of length Δ. Index the relevant prompt suffix as $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(s)}$, oldest first. Row $\mathbf{x}^{(j)}$ occurs at position $n-s+j$. Its output Δ positions later is observed at continuation time

$$t_j = \Delta - s + j \in \{1, \dots, \Delta\}. \tag{22}$$

This output depends on that row and newer inputs, and on no earlier row. At $j = s$, all newer inputs are the fixed continuation. Lemma 3 and the injective readout recover $\mathbf{x}^{(s)}$. Successively using $j = s-1, \dots, 1$ recovers every suffix row. This remains valid when $n < \Delta$: the selected output's earliest input is $n - s + j \geq 1$, so every window on its maximal-delay path is full. Equivalently, the most recent differing suffix row distinguishes two prompts at its selected time. Equal suffixes give equal future traces by Lemma 2.

On an open continuous suffix domain, every sufficient continuous initial encoder, composed with the continuous reachable-cache map, must be injective on sd dimensions. Invariance of domain gives the lower bound sd, and storing the suffix during prefill and recomputing gives the upper bound. □

Discrete token states. Under the same hypotheses, for N_{emb} distinct token embeddings with all relevant prompt and continuation sequences legal, there are exactly N_{emb}^s distinguishable reachable suffixes. The necessary and attainable fixed-width initial bit count is therefore $\lceil s \log_2 N_{\text{emb}} \rceil$. Here N_{emb} counts distinct embeddings, which need not equal the vocabulary size.

Proof of Corollary 1. Include the requested prompt-boundary logits $\mathbf{z}_n$ together with the future trace of the fixed length-Δ sequence. Let $s_+ = \min(n, \Delta+1)$. If $n \leq \Delta$, Theorem 4 already identifies every prompt row, so the joint observation identifies its last $s_+ = n$ rows. If $n > \Delta$, the future trace identifies $\mathbf{x}_{n-\Delta+1}, \dots, \mathbf{x}_n$. With those newer rows fixed, the boundary output $\mathbf{z}_n$ is injective in $\mathbf{x}_{n-\Delta}$ by Lemma 3 and injectivity of its readout. Its full input interval starts at $n - \Delta \geq 1$, so the lemma applies. This recovers precisely one additional row. Conversely, the boundary logits depend only on $\mathbf{x}_{n-\Delta}, \dots, \mathbf{x}_n$, and the continuation state only on the newer Δ rows, so equality of the last s_+ rows gives equal joint observations. When $\Delta = 0$, the injective rowwise stack and boundary readout identify the final prompt row directly.

For the continuous dimension claim, fix all prompt coordinates before these last s_+ rows and take a sufficiently small open affine slice varying the $s_+ d$ remaining coordinates inside the declared open input domain. The joint observation is injective on that slice. Any continuous representation sufficient to reproduce both the boundary output and future traces must therefore also be injective there. Invariance of domain gives the lower bound $s_+ d$. Storing the last s_+ input vectors and recomputing both responses supplies a continuous representation of that dimension, proving equality.

The joint response thus distinguishes exactly the last $\min(n, \Delta+1)$ input rows and has minimum continuous representation dimension $d \min(n, \Delta+1)$, in agreement with the input support of the cache-complete cone. The additional row is carried by the returned prompt logits, so the continuation state alone keeps the sd coordinates of Theorem 4. □

E.3 Verifiable Conditions for Normalized Attention

Positive-epsilon RMSNorm. For $\epsilon > 0$, define the unit-gain map

$$\mathcal{N}_\epsilon(\mathbf{x}) = \frac{\mathbf{x}}{\sqrt{\epsilon + \|\mathbf{x}\|^2/d}}. \tag{23}$$

It is a smooth bijection from $\mathbb{R}^d$ onto the open ball of radius $\sqrt{d}$. If $\mathbf{r} = \mathcal{N}_\epsilon(\mathbf{x})$, its inverse is

$$\mathbf{x} = \frac{\sqrt{\epsilon}\,\mathbf{r}}{\sqrt{1 - \|\mathbf{r}\|^2/d}}. \tag{24}$$

Writing $a = (\epsilon + \|\mathbf{x}\|^2/d)^{1/2}$ gives

$$D\mathcal{N}_\epsilon(\mathbf{x}) = a^{-1}I - \frac{\mathbf{x}\mathbf{x}^\top}{da^3}.$$

The tangential eigenvalues are a^{-1} and the radial eigenvalue is ϵ/a^3, all positive. An invertible diagonal gain preserves injectivity and nonsingularity.

A nonlinear decoder satisfying the assumptions. Set the query and key projections to zero, turn off the FFN output, and choose a combined value–output map on the full hidden space $\alpha_\ell I$ with $\alpha_\ell > 0$. Every full-window block becomes

$$\mathbf{h}_{\ell,t} = \mathbf{h}_{\ell-1,t} + \frac{\alpha_\ell}{\rho_\ell + 1} \sum_{j=0}^{\rho_\ell} \mathcal{N}_{\epsilon_\ell}(\mathbf{h}_{\ell-1,t-j}). \tag{25}$$

Fixed RoPE acts on zero query and key vectors and has no effect. For $\rho_\ell > 0$, holding newer rows fixed leaves a constant plus a nonzero multiple of the injective normalized oldest row. For $\rho_\ell = 0$, the map is $\mathbf{x} + \alpha_\ell \mathcal{N}_{\epsilon_\ell}(\mathbf{x})$. Its Jacobian is positive definite, and integration along a line segment shows strict monotonicity and therefore global injectivity. All oldest-input derivatives are nonsingular. A final positive-epsilon RMSNorm followed by a full-column-rank logit readout is injective. A tied logit readout is compatible when the vocabulary has at least d tokens, by choosing its embedding matrix to have column rank d.

This is an explicit nonlinear residual decoder that satisfies Theorem 4, and standard multihead attention spanning the full hidden width realizes αI by partitioning value coordinates among heads. The following conditions also admit nonzero, content-dependent attention and nonzero FFNs.

Lemma 4 (One varying normalized softmax row)**.** *Let* $\mathbf{r}$ *vary in a convex open set containing* $\mathbf{b}_0$*, and let* $D_0 > 0$ *and* $\mathbf{a}_0$ *be fixed. Define*

$$f(\mathbf{r}) = \mathbf{b}_0 + \alpha(\mathbf{r})(\mathbf{r} - \mathbf{b}_0), \qquad \alpha(\mathbf{r}) = \frac{e^{\mathbf{a}_0^\top \mathbf{r}}}{D_0 + e^{\mathbf{a}_0^\top \mathbf{r}}}.$$

If $1 + (1 - \alpha(\mathbf{r}))\mathbf{a}_0^\top(\mathbf{r} - \mathbf{b}_0) > 0$ *throughout the domain,* f *is globally injective and its derivative is nonsingular. On an open radius-*R_0 *ball, the condition* $2R_0\|\mathbf{a}_0\| \le 1$ *is sufficient.*

Proof. Since $\alpha > 0$, $\mathbf{b}_0$ is the unique input mapped to $\mathbf{b}_0$. Any other equal outputs require input displacements on the same positive ray from $\mathbf{b}_0$. On a ray $\mathbf{r} = \mathbf{b}_0 + t\mathbf{v}$, the output displacement length is proportional to $t\alpha(\mathbf{b}_0 + t\mathbf{v})$, whose derivative is

$$\alpha(\mathbf{r})\big[1 + (1 - \alpha(\mathbf{r}))\mathbf{a}_0^\top(\mathbf{r} - \mathbf{b}_0)\big] > 0.$$

The ray segment between candidates stays in the convex domain, proving injectivity. Direct differentiation and the matrix determinant lemma give

$$\begin{aligned} Df(\mathbf{r}) &= \alpha(\mathbf{r})\big[I + (1 - \alpha(\mathbf{r}))(\mathbf{r} - \mathbf{b}_0)\mathbf{a}_0^\top\big], \\ \det Df(\mathbf{r}) &= \alpha(\mathbf{r})^d\big[1 + (1 - \alpha(\mathbf{r}))\mathbf{a}_0^\top(\mathbf{r} - \mathbf{b}_0)\big] > 0. \end{aligned} \tag{26}$$

For the ball, $\mathbf{a}_0^\top(\mathbf{r} - \mathbf{b}_0) > -2R_0\|\mathbf{a}_0\| \ge -1$, which implies the required strict positivity. □

Single-head content dependence. Consider a bias-free local head with $\rho \ge 1$, hidden width d, key width d_k, and an invertible combined value/output map. Normalize every row by $\Gamma\mathcal{N}_\epsilon$, with positive epsilon and invertible diagonal gain. Write $\mathbf{r}_i = \mathcal{N}_\epsilon(\mathbf{x}_i)$, $\widetilde{W}_\mathrm{Q} = W_\mathrm{Q}\Gamma$, $\widetilde{W}_\mathrm{K} = W_\mathrm{K}\Gamma$, and $A = W_\mathrm{O}W_\mathrm{V}\Gamma$. The query and key vectors use fixed orthogonal positional rotations, and scores are dot products divided by $\sqrt{d_k}$. With the oldest row varying and all newer rows fixed, the current query $\mathbf{q}$ is fixed. If the oldest key rotation is S_old, put

$$\mathbf{a}_0 = \widetilde{W}_\mathrm{K}^\top S_\mathrm{old}^\top \mathbf{q}/\sqrt{d_k}, \qquad D_0 = \sum_{i \in \mathrm{newer}} e^{\mathbf{q}^\top \mathbf{k}_i/\sqrt{d_k}}, \qquad \mathbf{b}_0 = D_0^{-1} \sum_{i \in \mathrm{newer}} e^{\mathbf{q}^\top \mathbf{k}_i/\sqrt{d_k}} \mathbf{r}_i.$$

The current row is among the newer rows, so $D_0 > 0$, and their convex mean $\mathbf{b}_0$ lies inside the radius-$\sqrt{d}$ ball. The attention output is exactly $Af(\mathbf{r})$, with f from Lemma 4. Orthogonality gives

$$\|\mathbf{a}_0\| \le \frac{\sqrt{d}\,\|W_\mathrm{K}\Gamma\|\|W_\mathrm{Q}\Gamma\|}{\sqrt{d_k}}.$$

Consequently a sufficient uniform condition is

$$\frac{2d\,\|W_{\mathrm{K}}\Gamma\|\|W_{\mathrm{Q}}\Gamma\|}{\sqrt{d_k}} < 1, \qquad A \text{ invertible.} \tag{27}$$

It implies oldest-input injectivity and a nonsingular derivative after composing with RMSNorm and A. The current-row residual is fixed, so adding it changes neither conclusion. The non-strict inequality also suffices for injectivity, and the strict inequality defines an open parameter set. With identity positional rotations, the sharper condition $2d\|\widetilde{W}_{\mathrm{K}}^{\top}\widetilde{W}_{\mathrm{Q}}\|/\sqrt{d_k} \le 1$ suffices.

A nonzero normalized gated-SiLU residual. As in invertible residual self-attention (Kim et al., 2021), a row map $T(\mathbf{x}) = \mathbf{x} + g(\mathbf{x})$ is injective whenever g has global Lipschitz constant $\kappa < 1$, because

$$\|T(\mathbf{x}) - T(\mathbf{y})\| \ge (1-\kappa)\|\mathbf{x}-\mathbf{y}\|.$$

If g is differentiable, $I + Dg$ is nonsingular as well. For the bias-free branch

$$g(\mathbf{x}) = W_2[\mathrm{SiLU}(W_1\mathbf{r}) \odot W_3\mathbf{r}], \qquad \mathbf{r} = \Gamma_f\mathcal{N}_{\epsilon_f}(\mathbf{x}), \qquad R_f = \sqrt{d}\|\Gamma_f\|,$$

a valid global bound is

$$\mathrm{Lip}(g) \le \frac{\|\Gamma_f\|}{\sqrt{\epsilon_f}}\|W_2\|\|W_1\|\|W_3\|R_f\left(2 + \frac{\|W_1\|R_f}{4}\right). \tag{28}$$

Indeed, $\|\mathbf{r}\| < R_f$, $\|D\mathbf{r}\| \le \|\Gamma_f\|/\sqrt{\epsilon_f}$, $|\,\mathrm{SiLU}(u)| \le |u|$, and $|\,\mathrm{SiLU}'(u)| \le 1 + |u|/4$. Differentiating the Hadamard product, its first term is bounded by $\|W_3\|R_f(1 + \|W_1\|R_f/4)\|W_1\|$ and its second by $\|W_1\|R_f\|W_3\|$, and multiplication by W_2 and $D\mathbf{r}$ proves the displayed bound. A sufficiently small nonzero W_2 makes the bound less than one, and the attention and FFN gains and epsilons are separate parameters.

Proposition 1 (A criterion for multihead attention)**.** *At a positive-radius layer, suppose all heads share the same pre-attention RMSNorm and the same full visible window of w rows. The query and key projections are bias-free with dot-product scores, and fixed head-specific orthogonal RoPE is permitted. Let*

$$C_h = W_{\mathrm{O},h}W_{\mathrm{V},h}\Gamma, \qquad A = \sum_h C_h, \qquad \beta_h = \frac{d\|W_{\mathrm{K},h}\Gamma\|\|W_{\mathrm{Q},h}\Gamma\|}{\sqrt{d_{k,h}}}.$$

Assume A is invertible and

$$\eta = \sum_h \|A^{-1}C_h\|\left[e^{2\beta_h}(1+2\beta_h) - 1\right] < 1. \tag{29}$$

Then the attention output $\mathbf{o}$ is globally injective in its oldest normalized input with newer rows fixed, its derivative is nonsingular, and

$$\|\mathbf{o}(\mathbf{r}) - \mathbf{o}(\mathbf{r}')\| \ge \frac{\sigma_{\min}(A)(1-\eta)}{w}\|\mathbf{r}-\mathbf{r}'\|. \tag{30}$$

Proof. Each head has normalized mean $f_h(\mathbf{r}) = \mathbf{b}_h + \alpha_h(\mathbf{r})(\mathbf{r} - \mathbf{b}_h)$, hence

$$Df_h = \alpha_h[I + (1-\alpha_h)(\mathbf{r}-\mathbf{b}_h)\mathbf{a}_h^{\top}].$$

Its scores have magnitude at most β_h, so $e^{-2\beta_h} \le w\alpha_h \le e^{2\beta_h}$ and $\|(\mathbf{r}-\mathbf{b}_h)\mathbf{a}_h^{\top}\| \le 2\beta_h$. Therefore

$$\|wDf_h - I\| \le e^{2\beta_h} - 1 + 2\beta_h e^{2\beta_h}.$$

Since $\mathbf{o} = \sum_h C_h f_h$ and $\sum_h A^{-1}C_h = I$,

$$\left\|D[wA^{-1}\mathbf{o}] - I\right\| \le \eta.$$

Integrate this derivative perturbation along a segment in the convex normalized-input ball:

$$\|wA^{-1}(\mathbf{o}(\mathbf{r}) - \mathbf{o}(\mathbf{r}')) - (\mathbf{r}-\mathbf{r}')\| \le \eta\|\mathbf{r}-\mathbf{r}'\|.$$

The reverse triangle inequality and multiplication by A/w give Equation (30). After invertible scaling, the derivative differs from the identity by a matrix of norm less than one, so it is nonsingular. Invertibility is required of the combined map $A = \sum_h C_h$; individual head maps may have lower rank and differ across heads. □

Composing either attention criterion with the injective pre-normalization, the fixed current residual, and a residual FFN whose bound in Equation (28) is below one verifies the hypotheses of Theorem 4 on the full-window maps of the maximal-delay paths. Radius-zero layers must in addition be injective row maps, as in the construction above. Standard multihead attention spanning the full hidden width can make A invertible, and for such value and output weights, sufficiently small nonzero query/key and FFN output matrices give an open parameter region that satisfies the strict bounds.

F Extended Related Work

We expand the comparisons of Section 6 in three groups: methods that select or reuse work, systems and kernels that execute the retained work, and theory that constrains the information needed for exact outputs.

Local and sparse attention architectures. Sparse Transformers factorize attention into sparse patterns (Child et al., 2019), Longformer combines local and global attention (Beltagy et al., 2020), and Big Bird also includes random connections (Zaheer et al., 2020). Mistral-7B, Gemma 2, and Gemma 3 bring sliding-window or interleaved local and global attention to pretrained decoders (Jiang et al., 2023; Gemma Team, 2024; 2025). When such a pattern is causal and fixes the reads of every row in advance, Equation (2) applies with the pattern's supports, and each global layer requires computation at all prompt positions in every lower layer (Appendix B.2). Multi-query and grouped-query attention share key/value heads across queries (Shazeer, 2019; Ainslie et al., 2023). They shrink the cache but leave unchanged the positions that each query reads.

Cache compression and selection. H2O, SnapKV, and StreamingLLM select or retain subsets of cached context (Zhang et al., 2023; Li et al., 2024; Xiao et al., 2024), and Quest keeps the full KV cache but selects the pages to read using the current query (Tang et al., 2024). These methods change what later decoding stores or reads, so they do not guarantee the outputs of dense execution. C3P keeps the continuation state intact and removes only prefill work on which no requested output or state entry depends, so a compression policy can also be applied to the state that C3P produces.

Cache reuse and serving systems. PagedAttention manages KV memory (Kwon et al., 2023). SGLang reuses shared prefixes, Prompt Cache reuses schema-defined prompt modules, and CacheBlend selectively recomputes cached chunks to fuse them (Zheng et al., 2024; Gim et al., 2024; Yao et al., 2025). Marconi manages prefix caching for hybrid attention and recurrent models (Pan et al., 2025). KV-Runahead parallelizes prompt processing, Sarathi-Serve schedules chunked prefills, and DistServe separates prefill from decoding (Cho et al., 2024; Agrawal et al., 2024; Zhong et al., 2024). PrefillOnly specializes execution for workloads that need a single output (Du et al., 2025). These systems organize memory, reuse, and scheduling across requests and devices, while C3P reduces the work of a single prefill.

Attention kernels. FlashAttention uses IO-aware tiling (Dao et al., 2022), FlashAttention-2 improves work partitioning (Dao, 2024), and FlashAttention-3 exploits asynchronous execution on Hopper GPUs and also provides a low-precision option (Shah et al., 2024). FlashMask skips statically masked blocks (Wang et al., 2025), and FlashInfer provides flexible attention and cache execution (Ye et al., 2025). These kernels determine how efficiently a given set of rows is executed, whereas C3P determines which rows are executed at all.

Training-time selection and compression. Token selection and activation compression also reduce the cost of training. TokenTune approximates gradient computation through selected tokens during fine-tuning (Simoulin et al., 2024), and PAMM stores compressed input activations and approximates the weight gradients (Khalaf et al., 2026). Both approximate a training computation, whereas C3P applies to inference with fixed weights and removes only computation that cannot affect the requested outputs or the continuation state.

Evaluating efficiency. Green AI argues for treating efficiency as an evaluation criterion alongside accuracy (Schwartz et al., 2020). Measurements of inference energy reinforce the need to evaluate the work performed for each task (Luccioni et al., 2024), and reporting energy and carbon footprints systematically requires explicit accounting boundaries (Henderson et al., 2020). MLPerf Inference defines workloads, scenarios, and quality targets so that inference measurements are comparable (Reddi et al., 2020). Our measurements report prefill latency, peak allocated memory, and GPU energy under the protocols of Appendix C, each with its comparator and measurement boundary.

State information and lower bounds. Attention has been analyzed through space lower bounds for streaming computation (Haris & Onak, 2025; Chen et al., 2026), I/O complexity (Saha & Ye, 2024), parameter identifiability (Henry et al., 2025), compositional prefix memories (Nowak et al., 2026), and expressivity–efficiency tradeoffs for hybrid sequence models (Cooper et al., 2026). In bipermutive cellular automata, permutivity in the extreme input variables lets preimages be computed from boundary values (Mariot & Leporati, 2014). Section 5 reconstructs the prompt suffix from future outputs along maximal-delay paths under explicit observability conditions. Its single-layer result gives the minimum continuous value-state dimension for fixed known keys, while the multilayer result concerns the input-suffix information distinguished by the outputs.