TOKENGATE: FAST AND MEMORY-EFFICIENT PREFILL WITH TRAINABLE TOKEN PRUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Expert human readers process large inputs (e.g., code repositories and long documents) by skimming to closely read only what matters. On the other hand, language models process all input tokens through each model layer, resulting in significant processing costs for long-context inputs. We introduce TOKENGATE, a small, drop-in component that extends existing trained models to enable end-to-end learnable token pruning over long-context inputs for fast and memory-efficient prefill. To enable end-to-end learning despite the non-differentiable top-K operator for token pruning, we introduce *token-gated attention*, a mechanism inspired by Mixture-of-Experts that re-weights attention logits using token importance scores. Across diverse tasks, TOKENGATE achieves 1.4-$11.3\times$ faster time-to-first token (TTFT) and 1.5-$3.4\times$ lower peak memory at no accuracy degradation for prefill in transformers. Compared to the best-performing prior methods for token-pruning, TOKENGATE gives up to $6.4\times$ faster prefill and $2\times$ lower peak memory. TOKENGATE natively extends both standard transformers (e.g., Qwen3) and hybrid-attention models (e.g., Qwen3.5) with simple lightweight adaptation comparable to standard parameter-efficient fine-tuning.

1 INTRODUCTION

Long-context inputs underpin exciting recent language model deployments, where the model's ability to process vast swaths of personalized user data and live content during inference has led to transformative capabilities in deep research, large-scale document processing, coding agents, and data agents (OpenAI, 2025b; Gemini, 2025; Anthropic, 2025b;a; Anysphere, 2025; GitHub, 2025). When processing large inputs, like code or long documents, experts readers skim to closely read only what matters. Yet, transformers process every token of their input through each layer, whether or not the token bears critical information. The cost of this full read is attention compute that grows quadratically with input length, as well as MLP compute and KV-cache memory that grow linearly, making long-context prefill a key bottleneck in model serving.

Recent research proposes to address this efficiency challenge for long-context prefill through two primary approaches. First, one body of work proposes *training-free token pruning*, using attention-based scoring heuristics to prune token hidden states, forwarding only the highest-scoring tokens to upper layers (Jo et al., 2025; Long et al., 2025; Jin et al., 2025; Shi et al., 2024; Yang et al., 2024a; Fu et al., 2024; Chen et al., 2026; Wang et al., 2021). These methods are inexpensive to deploy, requiring no model training, but demonstrate limited speedups before degrading accuracy. Alternatively, a large body of work instead develops *architectural modifications* to the standard transformer architecture (Vaswani et al., 2017). These methods offer promising prefill-efficiency improvements, but are expensive to train, often requiring pre-training from scratch and substantially different hardware and software than existing ecosystems (Dehghani et al., 2018; Bae et al., 2025; Graves, 2016; Raposo et al., 2024; Ainslie et al., 2023; Kim et al., 2021; He et al., 2025; Gu & Dao, 2023; Dao & Gu, 2024). To bridge this gap, we ask the question, *can we endow existing trained models with improved long-context prefill efficiency while maintaining strong accuracy?*

In this work, we introduce TOKENGATE, a small, drop-in component that extends transformer-based models, enabling end-to-end trainable token pruning over long-context inputs for efficient prefill. The model forward pass using TOKENGATE decomposes into three parts, as Figure 1a shows. First, layers below the gate module perform a standard forward pass, processing all tokens through each

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.

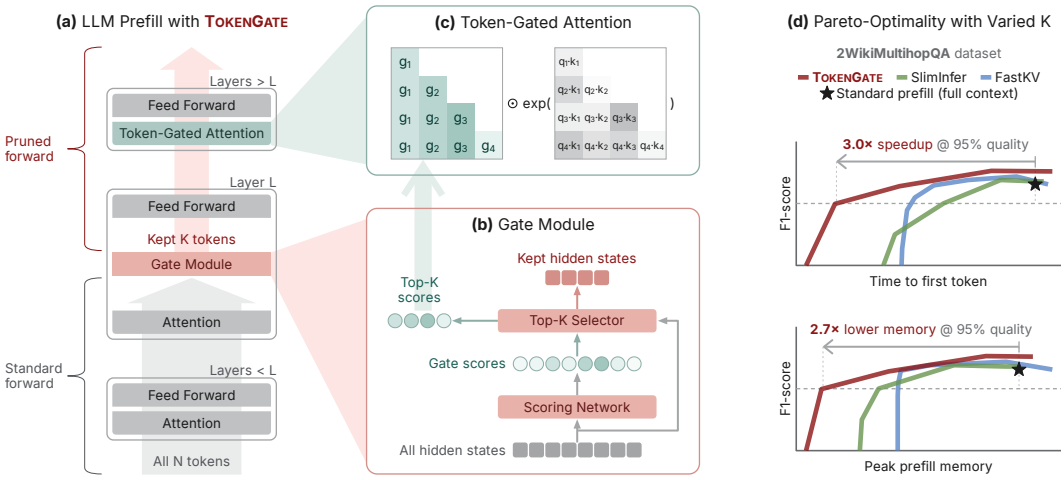


Figure 1: **Overview of TokenGate. (a)** Layers up to L prefill all N context tokens as usual; at layer L the gate module keeps K of them, and every layer above L processes only those K tokens. **(b)** The gate module scores each hidden state with a small scoring network, and a top-K selector forwards the kept hidden states together with their importance scores. **(c)** Token-gated attention multiplies each attention weight by the gate value g_j of its key, which enables end-to-end training. **(d)** On 2WikiMultihopQA question answering, with Qwen3-1.7B, TokenGate pushes the Pareto frontier of accuracy versus time-to-first-token and peak prefill memory; at 95% of the full-context accuracy it prefills 3.0× faster and with 2.7× less memory, outperforming the best training-free token pruning methods, like FastKV (Jo et al., 2025) and SlimInfer (Long et al., 2025).

layer. Second, tokens pass through the gate module, which produces a score for each token, retaining only the top-K for continued processing. Third, the layers above the gate module process the selected tokens with *token-gated attention*, a mechanism that re-weights the attention computation using the generated importance scores. TokenGate reduces prefill compute and memory because pruned tokens are never forwarded through the post-gate layers and these layers never generate KV cache entries.

Token pruning uses a non-differentiable top-K operator, which presents a challenge for end-to-end learning. We introduce *token-gated attention* to overcome this issue. Our method takes inspiration from Mixture-of-Experts (MoE) (Shazeer et al., 2017; Fedus et al., 2022), which achieve sparsity over *activated model parameters* by gating expert sub-networks. We use a similar form, but instead weight attention logits to achieve *sparsity over processed tokens*. Using this technique, we demonstrate that we can adapt existing trained models with cost similar to standard adaptation techniques. For instance, we can fine-tune existing models with TokenGate using simple next-token-prediction loss, with no auxiliary loss and total train time similar to supervised fine-tuning on the base model. The resulting TokenGate model matches or exceeds accuracy over the trained base model, while significantly reducing prefill latency and memory.

We evaluate TokenGate on diverse tasks, including code completion, multi-hop QA, multi-doc summarization, medical extraction, and needle-in-a-haystack, using both traditional transformer models and a recent hybrid attention one. Overall, we show the following findings **(1)** TokenGate *reduces TTFT and peak prefill memory* by 1.4-11.3× and 1.5-3.4×, respectively, at no accuracy degradation compared to the finetuned model baseline with full-context, and outperforms all training-free token pruning methods, by up to 6.4× faster prefill (TTFT) and 2× lower peak prefill memory relative to the next-best baseline, **(2)** TokenGate *natively extends existing recent hybrid-attention architectures*, like Qwen3.5-2B (Qwen Team, 2026), where it achieves 3.8× lower TTFT over the full-context baseline at no accuracy degradation, and 2.1× lower TTFT than the next best-performing token-pruning method at fixed accuracy. **(3)** *Training models is efficient* with TokenGate, incurring minimal overhead compared to standard parameter-efficient fine-tuning techniques, like LoRA (Hu et al., 2021).

Overall, we summarize our contributions as follows:

- We introduce TokenGate, a small, drop-in component that extends existing trained models to enable end-to-end learnable token pruning for fast and memory-efficient prefill.

- We propose *token-gated attention*, a mechanism for training models end-to-end despite the gate module's non-differentiable top-K operator.
- Our systematic evaluation finds that TOKENGATE reduces TTFT and peak prefill memory by up to $6.4\times$ and $2\times$, respectively, compared to the next best-performing baselines, TOKENGATE natively extends transformer-based models, including recent hybrid-attention architectures. Further, training is efficient, with total train time comparable to standard parameter-efficient fine-tuning techniques.

2 RELATED WORKS

Training-free Token Pruning. One body of prior work optimizes prefill using training-free token-pruning methods (e.g., FastKV, SlimInfer, DASH, and GemFilter) (Jo et al., 2025; Long et al., 2025; Jin et al., 2025; Shi et al., 2024; Yang et al., 2024a; Fu et al., 2024; Chen et al., 2026; Wang et al., 2021). These methods prune token hidden states at a mid-layer of the model using attention-based heuristics to rank tokens, forwarding only the highest-scoring tokens to upper layers of the model. This body of work is closest to ours, however, unlike our work, these methods are not trainable. As we observe in (Section 5), these methods offer limited speedups before degrading accuracy because they employ heuristic attention-based token-ranking functions that under-perform our end-to-end trained scoring network, and they prevent co-adaptation of the model backbone with the pruning function, which our end-to-end trainable design enables.

Architectural Modifications. Several lines of work explore architectural modifications to the self-attention-based transformer (Vaswani et al., 2017), reducing the compute and memory of processing long-context inputs. First, one body of work studies architectures that achieve *adaptive compute per token* through dynamic token routing or learned pruning decisions, but are expensive to train. These works require pre-training from scratch (Dehghani et al., 2018; Bae et al., 2025; Graves, 2016; Raposo et al., 2024; Ainslie et al., 2023) or expensive fine-tuning with auxiliary loss terms that fix the token budget during train time, requiring re-training for each inference-time token budget (Kim et al., 2021; He et al., 2025; Goyal et al., 2020; Rao et al., 2021). In contrast to all of these works, TOKENGATE natively extends popular pre-trained language models and is simple and efficient to train. We show that with a single fine-tuning pass, the resulting TOKENGATE model generalizes across input lengths and token budgets (Section 5, Appendix D.3).

Other works on *sub-quadratic architectures* for language modeling introduce promising, new algorithms that reduce the *per token* compute and memory of attention operations, including new compute primitives and sparse attention mechanisms (Gu & Dao, 2023; Dao & Gu, 2024; Yang et al., 2024c;b; Katharopoulos et al., 2020; Sun et al., 2024; Gao et al., 2024; Liu et al., 2025; Yuan et al., 2025; Lu et al., 2025; Xiao et al., 2024; Huang et al., 2025; Jiang et al., 2024; Lai et al., 2025). However, unlike TOKENGATE, these architectures still process all tokens through each model layer. In Section 5 we demonstrate TOKENGATE natively extends recent hybrid architectures with sub-quadratic layers (Qwen Team, 2026), and enables even more efficient prefill.

Conditional Computation. Mixture-of-experts layers achieve sparsity over the activated model parameters, making it possible to efficiently scale model capacity (Shazeer et al., 2017; Fedus et al., 2022; Liu et al., 2024). Our work is inspired by the success of this line of work, however, we focus on achieving sparsity over a different axis, the input context length, for scalable long-context prefill.

Decode Optimizations. A large body of work targets the cost of decoding rather than prefill, evicting or merging KV-cache entries after prefill (Zhang et al., 2023; Li et al., 2024; Xiao et al., 2023; Ge et al., 2023; Cai et al., 2024; Kim et al., 2025), and performing early exit or per-token routing for decoded tokens (Schuster et al., 2022; Elhoushi et al., 2024). These methods do not improve prefill efficiency and are complementary to prefill optimizations like ours.

3 PRELIMINARIES

Transformer Models and Notation. Given a sequence of tokens $x = (x_1, \ldots, x_N)$, the language model embeds each token x_i into a vector $h_i^0 \in \mathbb{R}^{d_{\text{model}}}$ and passes the sequence through a stack of N_L transformer layers. Layer $\ell \in \{0, \ldots, N_L - 1\}$ reads the *residual stream* h_i^ℓ and applies a self-attention sub-layer followed by a token-wise MLP. Both sub-components are preceded by a

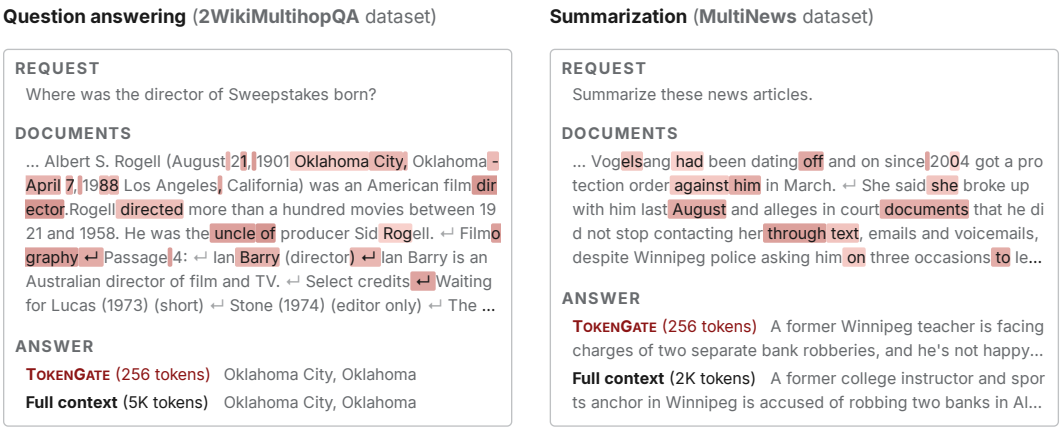


Figure 2: Visualization of TOKENGATE applied on sample inputs from the 2WikiMultihopQA dataset (left) and MultiNews dataset (right), with $K = 256$ kept tokens, highlighted in red (darker denotes higher token importance score by the gate). Detailed visualizations of real samples from all datasets are in Appendix E.

normalization layer and add their output back to the residual stream, as given by:

$$a_i^\ell \;=\; h_i^\ell + \mathrm{Attn}^\ell\big(\mathrm{LN}_1(h^\ell)\big)_i, \qquad h_i^{\ell+1} \;=\; a_i^\ell + \mathrm{MLP}^\ell\big(\tilde{a}_i^\ell\big), \qquad \tilde{a}_i^\ell \;=\; \mathrm{LN}_2(a_i^\ell). \tag{1}$$

We call a_i^ℓ the post-attention residual of token i at layer ℓ. The MLP acts on each position independently, whereas attention mixes information across positions.

In standard self-attention, the normalized inputs $\mathrm{LN}_1(h^\ell)$ are projected to queries, keys, and values $q_i, k_j, v_j \in \mathbb{R}^{d_{\text{head}}}$, and token i attends causally to itself and the tokens before it:

$$\mathrm{Attn}_i \;=\; \sum_{j\le i} \alpha_{ij}\, v_j, \quad \alpha_{ij} \;=\; \mathrm{softmax}_{j\le i}\big(z_{ij}\big) \;=\; \frac{\exp(z_{ij})}{\sum_{j'\le i}\exp(z_{ij'})}, \quad z_{ij} \;=\; \frac{q_i^\top k_j}{\sqrt{d_{\text{head}}}}. \tag{2}$$

Prefill, decode, and the KV cache. Generation proceeds in two phases. *Prefill* runs Eq. 1 over the whole input of N tokens and stores every layer's keys and values, the *KV cache* $\{(k_j^\ell, v_j^\ell)\}_{j\le N}$. *Decode* then produces one token per step, computing attention against the cache and appending to it. The cost of both phases grows with the input length N. Prefill compute scales as $O(N^2 \cdot N_L)$ for attention and $O(N \cdot N_L)$ for the MLPs, and the KV cache occupies $O(N \cdot N_L)$ memory.

3.1 PROBLEM SETTING

We consider an LLM input prompt $x = [x_R, x_D]$, where $x_R = (x_1, \ldots, x_r)$ is a request instruction or question and $x_D = (x_{r+1}, \ldots, x_N)$ is the context needed to answer it, such as research articles, code repositories, medical records, legal documents, or financial filings. In practice, x_D may be supplied by the user or retrieved by a system or an agent given x_R. As an example, a user uploads a document study as a PDF to an AI assistant and asks "what are the side effects reported in this study?" (Google, 2023; OpenAI, 2025a; Anthropic, 2024; Google, 2025; Microsoft, 2025; Adobe, 2025; Perplexity AI, 2025; ChatPDF, 2025). Here, x_R corresponds to the user's question, and x_D corresponds to the attached document, which can be thousands to hundreds of thousands of tokens. Figure 2 shows two such input prompts from our datasets.

In this setting, the documents are provided only at request time, so its KV cache cannot be precomputed. Our goal is two-fold: we aim to *maximize quality*, measured by task accuracy, and *minimize cost*, both prefill time-to-first-token (TTFT), which is the latency users experience before seeing any response, and prefill memory footprint, which limits throughput and concurrent user requests.

4 TRAINABLE TOKEN PRUNING WITH TOKENGATE

TOKENGATE is a drop-in neural network component, which places a gate module within a specified transformer layer[1], L, to score and prune tokens, as shown in Figure 1. In this section, we describe how TOKENGATE enables *trainable* token pruning using two key components, the *gate module* (Section 4.1) and *token-gated attention* (Section 4.2). We will then overview an end-to-end training recipe for adapting existing models (Section 4.3).

Overall, the full TOKENGATE forward pass with gating at layer L decomposes into three phases. First, tokens pass through layers below the gate module, using a *standard forward* pass. Second, the *gate module* processes all tokens, producing a score for each, and selecting a subset to prune. Third, the model performs a *pruned forward*, processing only selected tokens in layers above the gate module, using *token-gated attention*, which re-weights attention scores using the importance scores. TOKENGATE reduces both the compute and memory footprint during prefill by preventing model layers after the gate module from processing pruned tokens or generating KV-cache entries for them.

4.1 GATE MODULE

The gate module is placed within a transformer layer, L, and selects a subset of tokens to continue processing through the remaining model layers during prefill. As Figure 1b shows, the gate module consists of two simple sub-components: a *scoring network* and a *top-K selector*.

Scoring Network. For each token j in the set of prunable tokens $\mathcal{P}$, an MLP reads its normalized post-attention residual at layer L, $\tilde{a}_j^L$ (Eq. 1), and emits a scalar score $\tilde{b}_j$, which we normalize along the sequence with RMSNorm (Zhang & Sennrich, 2019) into the importance score b_j, following:

$$\tilde{b}_j = \mathrm{MLP}_\theta\big(\tilde{a}_j^L\big), \qquad b_j = \frac{\tilde{b}_j}{\sqrt{\frac{1}{|\mathcal{P}|}\sum_{j'\in\mathcal{P}} \tilde{b}_{j'}^2 + \epsilon}} \text{ for } j \in \mathcal{P}, \qquad b_j := 0 \text{ for } j \in \mathcal{N}, \tag{3}$$

where the scores are fixed to zero for the *non-prunable tokens* $\mathcal{N}$, which can consist of answer tokens during model training, decoded tokens, or any optionally specified tokens during preill, e.g., a user question. The MLP is initialized to output zero, so that $b_j = 0$ for all j at the start of training.

Top-K Selector. The gate selects the token-set S, consisting of the K highest-scoring prunable tokens, as well as any non-prunable tokens, for continued processing:

$$S = \text{top-}K_{j\in\mathcal{P}}\, b_j \;\cup\; \mathcal{N}. \tag{4}$$

The residual stream for pruned tokens is stopped at layer L, and the selected tokens S are forwarded through the layer L MLP and layers $\ell > L$.

4.2 TOKEN-GATED ATTENTION

The gate module's top-K selector introduces a non-differentiability, creating a challenge for end-to-end learning. To overcome this challenge, we introduce *token-gated attention*, which integrates token importance scores generated by the gate module into upper model layers, providing a gradient path. Our method is inspired by Mixture-of-Experts (MoE) (Shazeer et al., 2017). In MoE, a router network weights the outputs of expert sub-networks to achieve *sparsity over activated model parameters*. In contrast, token-gated attention re-weights attention logits to achieve *sparsity over processed tokens*.

Token-gated attention multiplies each token's attention weight by the gate value g_j, computed from it's importance score b_j (Eqs. 5), in layers $\ell > L$. Specifically, for self-attention, in contrast to Eq. 2, the attention weights with token-gated attention take the form of Eq. 5. At the start of training with a base model, token-gated attention is equivalent to top-K attention as the importance scores are 0.

$$\alpha_{ij} = \frac{\exp(z_{ij})\, g_j}{\sum_{j'\in S,\, j'\le i} \exp(z_{ij'})\, g_{j'}}, \qquad i, j \in S, \qquad g_j = \mathrm{softmax}_{j\in S}(b_j) \tag{5}$$

[1]The design generalizes to multiple gate modules at different layers, each selecting from the tokens kept by the layers below it and contributing to the token-gated attention in layers above.

We make two modifications to this basic form: *head-wise scaling*, to allow different heads to have different scales for the gate value, and *temperature scaling*, to enable length generalization so that trained models perform robustly at various pruning ratios. We ablate both decisions in Appendix D.

Head-wise. We scale the normalized importance scores b_j by a learnable gain factor, $\gamma^{\ell h}$, indexed by layer ℓ and head h, and the gate value is the *softmax* of the result (Eq. 6).

$$g_j^{\ell h} = \mathrm{softmax}_{j \in S}\left(\mathrm{softplus}(\gamma^{\ell h})\, b_j\right) \tag{6}$$

Intuitively, $\gamma^{\ell h}$ learns how strongly each attention head gates the attention logits. The softplus keeps every gain positive, so a higher importance score corresponds to higher attention weight. At initialization, $\mathrm{softplus}(\gamma^{\ell h})$ is set to the standard deviation of layer ℓ's logits so that it is on the same scale as the attention logits.

Temperature Scaling. Pruning shrinks the number of keys each query sees from its original position p_i to its rank m_i among kept tokens. This pruning sharpens the softmax relative to the full-context model. Following the length generalization approach of SSMax (Nakanishi, 2025), we use a learnable positive temperature for attention, scaled by the log-ratio of the two counts (Eq. 7). The parameter σ is initialized such that temperature is 1 at full context.

$$\tau_i^{\ell h} = \mathrm{softplus}(\sigma^{\ell h}) \cdot \frac{\log m_i}{\log \max(2, p_i)} \tag{7}$$

Putting everything together. Substituting Eq. 6 into Eq. 5, we can write the token-gated attention as

$$\alpha_{ij}^{\ell h} = \mathrm{softmax}_{j \in S,\, j \le i}\left(\tau_i^{\ell h}\left(\tfrac{q_i^\top k_j}{\sqrt{d_{\text{head}}}} + \mathrm{softplus}(\gamma^{\ell h})\, b_j\right)\right). \tag{8}$$

This form is straightforward to implement as an efficient kernel based on FlashAttention (Dao, 2023)(Appendix A).

4.3 End-to-end Gate Training

Despite TokenGate's non-differentiable top-K selection (Eq. 4), the task loss is a differentiable function of the scoring network, almost everywhere. This is due to token-gated attention, which injects the top-K gate values into the attention logits at each layer above the token gate. When we choose $K > 1$, the gate value g_j has non-zero derivatives with respect to the scoring network weights. This mirrors how the router is trained in MoE through its top-K expert selection: the router's value multiplies the output of each selected expert, just as our gate value multiplies the attention score of each selected token (Shazeer et al., 2017; Fedus et al., 2022). Intuitively, a token whose additional attention mass lowers the task loss receives a larger score. A token that distracts receives a smaller one, allowing the scoring network to learn token importance. This mechanism allows TokenGate to be trained end-to-end jointly with the model backbone simply using standard task losses.

Loss Function. In Section 5, we fine-tune existing pretrained models with LoRA (Hu et al., 2021) along with the TokenGate parameters, demonstrating that the model backbone and gate layer can co-adapt with minimal overhead to standard training. Under this setting, training the model with TokenGate uses a standard next-token, cross-entropy loss,

$$\mathcal{L}_K = -\sum_{t \in \text{target}} \log p_\phi\big(x_t \mid x_{<t};\, K\big), \tag{9}$$

where $p_\phi(\cdot\,; K)$ is the next token probability with a token budget of K. The trainable parameters ϕ are the scoring network parameters θ, the scale factors $\{\gamma, \sigma\}^{\ell h}$, and the LoRA (Hu et al., 2021) parameters on the model backbone following standard practice.

Top-K Budget Sampling. Training with the TokenGate should ensure that the model both adapts to the task, improving its accuracy at full context, and learns a strong scoring network, improving its accuracy at restricted token budgets. As such, a simple choice for the top-K budget to ensure both criteria are met is to use the full-context length with K_{full} as all prunable tokens $|\mathcal{P}|$ and also sample

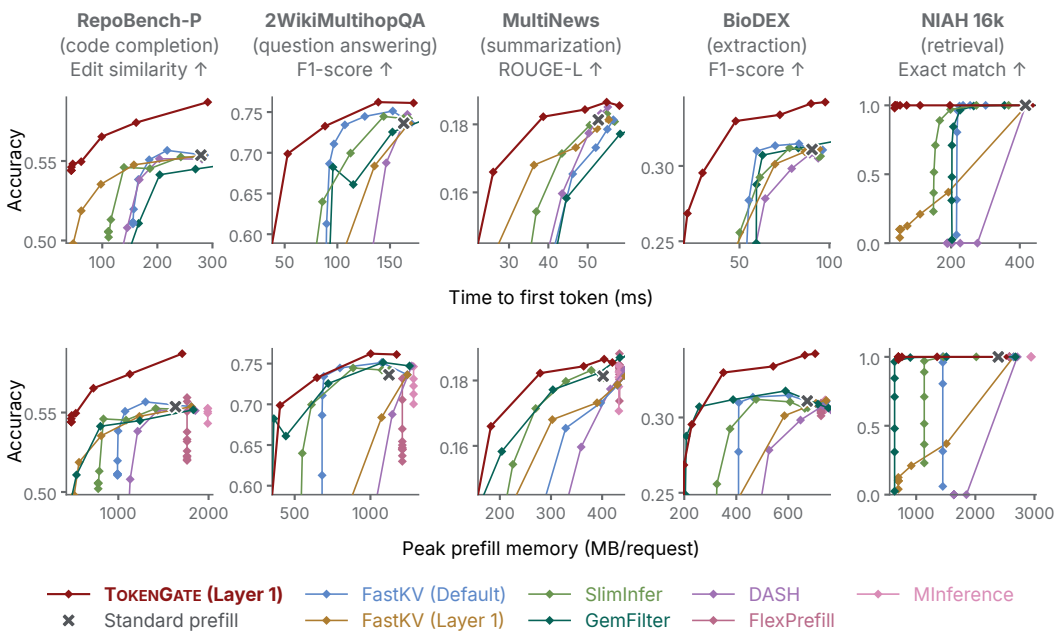


Figure 3: Accuracy vs. time-to-first-token (top) and accuracy vs. peak GPU prefill memory (bottom) with Qwen3-1.7B. Points that incur significant overhead over standard prefill at full context (grey cross), or significantly poor accuracy, are excluded; see Appendix D.2 for full plots.

a smaller K_{prune} per training instance. We average the resulting losses as follows:[2]

$$K_{\text{full}} = |\mathcal{P}|, \qquad K_{\text{prune}} \sim \text{LogUniform}\big(1,\ |\mathcal{P}|/2\big), \qquad \mathcal{L} = \tfrac{1}{2}\Big(\mathcal{L}_{K_{\text{full}}} + \mathcal{L}_{K_{\text{prune}}}\Big). \tag{10}$$

The K_{full} term anchors the model's accuracy at full context and gives every token a gradient, including tokens the sampled budget pruned. Meanwhile, the log-uniform term exposes the gate to a range of pruning ratios, allowing the trained model to generalize across a wide budget range without re-training. In Section 5, we demonstrate that using this budget sampling, models with the TokenGate can be trained efficiently, with minimal overhead compared to standard training. For each step, the forward pass up to the gate layer runs once at full context, and the two forwards after using K_{full} and K_{prune} share the compute graph. Thus, the overhead of each TokenGate train step comes solely from the pruned forward at K_{prune}. Since $K_{\text{prune}} \ll K_{\text{full}}$, this overhead is minimal, as we show in Section 5.

5 Experimental Results

We evaluate the effectiveness of TokenGate on diverse tasks, including code completion, multi-hop QA, summarization, medical extraction, and needle-in-a-haystack. Overall, results demonstrate:

- TokenGate *reduces TTFT and peak prefill memory* by 1.4-11.3× and 1.5-3.4×, respectively, at no accuracy degradation compared to the base model with full-context, and outperforms the next best-performing token pruning baselines by up to 6.4× and 2×, respectively.
- TokenGate *extends existing transformer-based models*, including recent hybrid-attention architectures, achieving 3.8× lower TTFT than the full-context baseline at no accuracy degradation and 2.1× lower TTFT than the best prior methods, at fixed accuracy.
- Training models with TokenGate is efficient, incurring *minimal training overhead*, with total training time comparable to standard parameter efficient fine-tuning.

We provide details of our experimental setup in Appendix C, additional experimental results including context-length extrapolation and ablations in Appendix D, and visualization in Appendix E.

[2]In practice, we sample from a discrete list following this distribution for training efficiency with PyTorch compilation.

Datasets. We evaluate on five diverse tasks, including 3 LongBench datasets with available train sets (Bai et al., 2024), 2WikiMultihopQA (Ho et al., 2020), a multi-hop QA dataset, RepoBench-P (Liu et al., 2023), a next-line code completion dataset, and MultiNews (Fabbri et al., 2019), a summarization dataset, as well as two additional tasks, BioDEX's drug-reaction extraction over biomedical papers (D'Oosterlinck et al., 2023) and needle-in-a-haystack (NIAH) (Kamradt, 2023). We use the official evaluation metric for each.

Baselines. We compare TOKENGATE against state-of-the-art token-pruning methods, FastKV (Jo et al., 2025), SlimInfer (Long et al., 2025), DASH (Chen et al., 2026), and GemFilter (Shi et al., 2024), as well as sparse attention methods, FlexPrefill (Lai et al., 2025) and MInference (Jiang et al., 2024). We run each baseline using a fine-tuned base model trained to convergence with LoRA (Hu et al., 2021) (r=16). We train our model with TOKENGATE to convergence using the same LoRA configuration. We train both a standard transformer model, Qwen3-1.7B (Yang et al., 2025), and a hybrid architecture, Qwen3-2B (Qwen Team, 2026), each on 4 A100 GPUS.

5.1 FAST AND MEMORY-EFFICIENT PREFILL

We evaluate the prefill efficiency of models trained with TOKENGATE. First, Figure 3 (top) shows the accuracy vs TTFT of each benchmarked method, demonstrating that across all datasets, the trained TOKENGATE model achieves pareto-optimal performance. Specifically, TOKENGATE reduces TTFT by 1.4–$11.3\times$ at no accuracy degradation compared to standard prefill, and by 1.2–$6.4\times$ compared to the next best-performing baseline. We measure FastKV, one of the best-performing baselines both at its default layer and at layer 1, the same layer index as the TOKENGATE. Notably, we observe that compared to FastKV (Layer 1), TOKENGATE significantly outperforms across all datasets, indicating the promise of end-to-end trainable token pruning for achieving prefill speedups.

Turning our attention to the bottom row of Figure 3, we study the prefill memory footprint, reporting accuracy versus peak prefill memory. Once again, across all dataset, the TOKENGATE model establishes the Pareto-frontier, reducing peak memory by 1.5–$3.4\times$ compared to standard prefill at matched accuracy. Moreover the TOKENGATE model achieves up to $2.1\times$ lower peak memory compared to the next best-performing baseline at matched accuracy. Overall, the results demonstrate that TOKENGATE significantly outperforms prior token-pruning methods, offering an exciting direction for improved prefill efficiency.

5.2 ABLATION STUDY: UNDERSTANDING THE PERFORMANCE OF TOKENGATE MODELS

We hypothesize that TOKENGATE owes its superiority to end-to-end learnable token pruning. Our visualizations of the kept tokens for real samples (Appendix E) shows that TOKENGATE flexibly learns very different pruning patterns in a task-dependent manner, ranging from keeping answer keywords for question answering to keeping stop-words at the ends of phrases for summarization (Figure 2). To understand this more rigorously, we investigate two key mechanisms: trainable token-selection, and co-adaptation with the trained backbone.

Trainable selector. First, to study the effectiveness of TOKENGATE's learned selector, we compare performance of the trained TOKENGATE model (Layer 1), to the same model with random token selection on 2WikiMultihopQA. Figure 5, demonstrates that the trained selector significantly outperforms the random-K selector on accuracy across all token budgets. Moreover, we compare against the FastKV selector applied at the same layer, Layer 1, of the finetuned base model. Notably, the TOKENGATE selector significantly outperforms the FastKV selector. This result demonstrates the value of an end-to-end trained gate module for token pruning: the trained scoring network leads to far better token selection compared to FastKV, which relies on a heuristic using attention scores that offer only weak signal at early layers, resulting in accuracy degradation for aggressive pruning.

Co-adaptation. Secondly, we analyze the effect of co-adaptation for TOKENGATE, which allows the model backbone and gate module to co-adapt. We compare the performance of random-K selection over the TOKENGATE model to the random-K selection over the finetuned base model. Notably, with the same random-K selector, the TOKENGATE model still outperforms the fine-tuned model, achieving higher accuracy across all token budgets, as Figure 5 shows. This result shows that TOKENGATE's end-to-end training allows the model to become more robust to token-pruning, likely redistributing information among tokens.

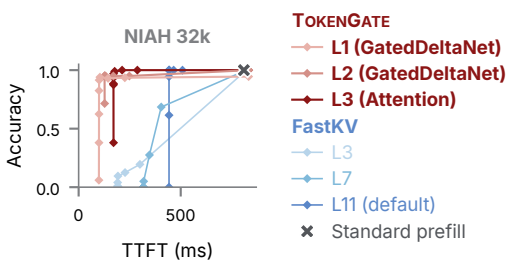


Figure 4: Accuracy vs. TTFT with the Qwen3.5-2B hybrid-attention architecture. Unlike TOKENGATE, FastKV can only prune at attention layers.

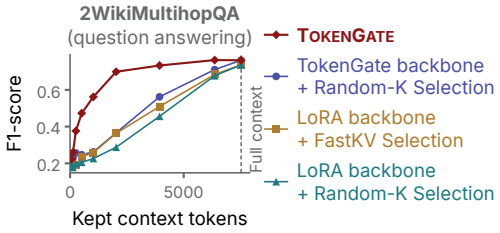


Figure 5: F1 vs. kept context tokens on 2WikiMultihopQA with Qwen3-1.7B for gate and Random-K selection on the TOKENGATE model and FastKV and Random-K selection on the LoRA-trained model.

Overall, the results show that end-to-end training with TOKENGATE enables state-of-the-art prefill performance through two complementary mechanism: the effectiveness of the learned scoring network, and co-adaptation, which allows the model to adapt jointly with the gating module for improved robustness to token pruning.

5.3 GENERALIZATION TO ALTERNATIVE ARCHITECTURES

We evaluate how TOKENGATE's performance extends to a recent hybrid-attention model, Qwen3.5-2B (Qwen Team, 2026) model, which uses full-attention layers every fourth layer and a Gated-DeltaNet (Yang et al., 2024b) for all other layer. We compare TOKENGATE against FastKV (Jo et al., 2025), the best-performing baseline in our main results (Section 5.1). We measure FastKV over the LoRA-tuned base model and the LoRA-tuned model with TOKENGATE, both at varying layers. FastKV uses an attention-based score function and natively applies only to full-attention layers (i.e., layer 3, 7, 11, etc). TOKENGATE natively extends both the full attention and Gated-DeltaNet layers, so we apply it over both. We provide further details on TOKENGATE for Qwen3.5 in Appendix C.4.

Figure 4 shows the accuracy vs TTFT of both methods at varying layers on the NIAH 32K dataset, demonstrating that TOKENGATE's strong performance extends to the hybrid-attention model. TOKENGATE (Layer 3) matches the accuracy of standard prefill at $3.8\times$ lower TTFT, and $2.1\times$ lower TTFT than the best-performing FastKV run at the same accuracy. Applying the TOKENGATE at the lower GatedDeltaNet layers (Layer 1,2) enable even higher TTFT reductions, up to $7.8\times$ faster prefill, at accuracy within 6% of standard prefill, once again significantly outperforming FastKV.

5.4 GATE TRAINING EFFICIENCY

We evaluate overhead of training models with TOKENGATE compared standard models, both with LoRA fine-tuning ($r = 16$). Table 2 in Appendix D.1 compares both, reporting total training time in A100 hours and the average per-step time during training for each evaluated dataset. Overall, we observe that a TOKENGATE step is slower by $1.3\times$ compared to the baseline due to its sampling at two token budgets per step. However, the TOKENGATE train runs converge after fewer steps on four of five tasks, leading to modestly *reduced* overall training time compared to the standard training.

6 CONCLUSION

In this work, we introduced TOKENGATE, a neural network component that extends existing trained transformer models, enabling *end-to-end trainable token pruning* using *token-gated attention*. We demonstrate that that across diverse tasks, TOKENGATE reduces TTFT and peak prefill memory by up to $11.3\times$ and $3.4\times$, respectively, at no accuracy degradation compared to standard prefill, and by up to $6.4\times$ and $2\times$, respectively, compared to the best performing baselines. Crucially, TOKENGATE is compatible with diverse transformer-based architectures, is simple to train, and incurs minimal training overhead, making it a promising drop-in replacement for standard fine-tuning over trained backbones. In the future, we envision models trained with TOKENGATE will enable a broad range of applications, including medical extraction, deep research, and coding agents, where processing long-context inputs is critical, but currently prohibitively expensive.

AI USE STATEMENT

We used coding agents to implement and run experiments, and minimally used generative AI tools to polish the writing of the paper. We did not use generative AI tools for any other part of this work. The authors reviewed all AI-assisted work and verified the code and experimental results produced with coding agents before reporting them. We take responsibility for the final content of this work, including text, claims and artifacts produced with the aid of generative AI.

REFERENCES

Adobe. Adobe Acrobat AI Assistant, 2025. URL https://www.adobe.com/acrobat/generative-ai-pdf.html.

Joshua Ainslie, Tao Lei, Michiel de Jong, Santiago Ontañón, Siddhartha Brahma, Yury Zemlyanskiy, David Uthus, Mandy Guo, James Lee-Thorp, Yi Tay, Yun-Hsuan Sung, and Sumit Sanghai. Colt5: Faster long-range transformers with conditional computation. *arXiv preprint arXiv:2303.09752*, 2023.

Anthropic. Collaborate with Claude on Projects, 2024. URL https://www.anthropic.com/news/projects.

Anthropic. Claude code: Deep coding at terminal velocity, 2025a. URL https://www.anthropic.com/claude-code.

Anthropic. Claude takes research to new places, 2025b. URL https://www.anthropic.com/news/research.

Anysphere. Cursor: The ai code editor, 2025. URL https://cursor.com.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. *arXiv preprint arXiv:2507.10524*, 2025.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024.

ChatPDF. ChatPDF: Chat with any PDF, 2025. URL https://www.chatpdf.com.

Yujie Chen, Tailai Chen, Yifeng Gao, Zoe Wanying He, Yijue Xu, Shaobo Wang, and Linfeng Zhang. Stability implies redundancy: Delta attention selective halting for efficient long-context prefilling. *arXiv preprint arXiv:2604.18103*, 2026.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Juechu Dong, Boyuan Feng, Driss Guessous, Yanbo Liang, and Horace He. Flex attention: A programming model for generating optimized attention kernels. *arXiv preprint arXiv:2412.05496*, 2024.

Karel D'Oosterlinck, François Remy, Johannes Deleu, Thomas Demeester, Chris Develder, Klim Zaporojets, Aneiss Ghodsi, Simon Ellershaw, Jack Collins, and Christopher Potts. Biodex: Large-scale biomedical adverse drug event extraction for real-world pharmacovigilance. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed A Aly, Beidi Chen, and Carole-Jean Wu. Layerskip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*, 2024.

Alexander R. Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir R. Radev. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 2022.

Qichen Fu, Minsik Cho, Thomas Merth, Sachin Mehta, Mohammad Rastegari, and Mahyar Najibi. Lazyllm: Dynamic token pruning for efficient long context llm inference. *arXiv preprint arXiv:2407.14057*, 2024.

Yizhao Gao, Zhichen Zeng, Dayou Du, Shijie Cao, Peiyuan Zhou, Jiaxing Qi, Junjie Lai, Hayden Kwok-Hay So, Ting Cao, Fan Yang, and Mao Yang. Seerattention: Learning intrinsic sparse attention in your llms. *arXiv preprint arXiv:2410.13276*, 2024.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*, 2023.

Gemini. Gemini Deep Research — your personal research assistant, 2025. URL `https://gemini.google/overview/deep-research/`.

GitHub. Github copilot: Your ai pair programmer, 2025. URL `https://github.com/features/copilot`.

Google. NotebookLM: Your personalized AI research assistant, 2023. URL `https://notebooklm.google`.

Google. Gemini for Google Workspace, 2025. URL `https://workspace.google.com/solutions/ai/`.

Saurabh Goyal, Anamitra R. Choudhury, Saurabh M. Raje, Venkatesan T. Chakaravarthy, Yogish Sabharwal, and Ashish Verma. Power-bert: Accelerating bert inference via progressive word-vector elimination. *arXiv preprint arXiv:2001.08950*, 2020.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

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

Shwai He, Tao Ge, Guoheng Sun, Bowei Tian, Xiaoyang Wang, and Dong Yu. Router-Tuning: A Simple and Effective Approach for Enabling Dynamic-Depth in Transformers, September 2025. URL `http://arxiv.org/abs/2410.13184`. arXiv:2410.13184 [cs.CL].

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics (COLING)*, 2020.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Yen-Chieh Huang, Pi-Cheng Hsiu, Rui Fang, and Ming-Syan Chen. Kv admission: Learning what to write for efficient long-context llm inference. *arXiv preprint arXiv:2512.17452*, 2025.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *arXiv preprint arXiv:2407.02490*, 2024.

Weisheng Jin, Maojia Song, Tej Deep Pala, Yew Ken Chia, Amir Zadeh, Chuan Li, and Soujanya Poria. Promptdistill: Query-based selective token retention in intermediate layers for efficient large language model inference. *arXiv preprint arXiv:2503.23274*, 2025.

Dongwon Jo, Jiwon Song, Yulhwa Kim, and Jae-Joon Kim. Fastkv: Decoupling of context reduction and kv cache compression for prefill-decoding acceleration. *arXiv preprint arXiv:2502.01068*, 2025.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks. `https://kellerjordan.github.io/posts/muon/`, 2024.

Gregory Kamradt. Needle in a haystack – pressure testing llms. `https://github.com/gkamradt/LLMTest_NeedleInAHaystack`, 2023.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. *arXiv preprint arXiv:2006.16236*, 2020.

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoo Yun, and Hyun Oh Song. Kvzip: Query-agnostic kv cache compression with context reconstruction. *arXiv preprint arXiv:2505.23416*, 2025.

Sehoon Kim, Sheng Shen, David Thorsley, Amir Gholami, Woosuk Kwon, Joseph Hassoun, and Kurt Keutzer. Learned token pruning for transformers. *arXiv preprint arXiv:2107.00910*, 2021.

Xunhao Lai, Jianqiao Lu, Yao Luo, Yiyuan Ma, and Xun Zhou. Flexprefill: A context-aware sparse attention mechanism for efficient long-sequence inference. *arXiv preprint arXiv:2502.20766*, 2025.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *arXiv preprint arXiv:2404.14469*, 2024.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Yang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jin Chen, Jingyang Yuan, Junjie Qiu, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruizhe Pan, Runxin Xu, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Size Zheng, T. Wang, Tian Pei, Tian Yuan, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Liu, Xin Xie, Xingkai Yu, Xinnan Song, Xinyi Zhou, Xinyu Yang, Xuan Lu, Xuecheng Su, Y. Wu, Y. K. Li, Y. X. Wei, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Zheng, Yichao Zhang, Yiliang Xiong, Yilong Zhao, Ying He, Ying Tang, Yishi Piao, Yixin Dong, Yixuan Tan, Yiyuan Liu, Yongji Wang, Yongqiang Guo, Yuchen Zhu, Yuduan Wang, Yuheng Zou, Yukun Zha, Yunxian Ma, Yuting Yan, Yuxiang You, Yuxuan Liu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhewen Hao, Zhihong Shao,

Zhiniu Wen, Zhipeng Xu, Zhongyu Zhang, Zhuoshu Li, Zihan Wang, Zihui Gu, Zilin Li, and Ziwei Xie. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Erhang Li, Fangqi Zhou, Fangyun Lin, Fucong Dai, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoran Wei, Haowei Zhang, Haowen Luo, Haozhe Ji, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, Jialiang Huang, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jingchang Chen, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jinhua Zhu, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexin Huang, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Liang Zhao, Liangsheng Yin, Lihua Guo, Lingxiao Luo, Linwang Ma, Litong Wang, Liyue Zhang, M. S. Di, M. Y Xu, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Panpan Huang, Peixin Cong, Peiyi Wang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, S. H. Liu, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Songyang Zhou, Tao Ni, Tao Yun, Tian Pei, Tian Ye, Tianyuan Yue, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjun Gao, Wentao Zhang, Xi Gao, Xiangwen Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyuan Li, Xu Chen, Xuecheng Su, Xuehai Pan, Xuheng Lin, Xuwei Fu, Y. Q. Wang, Yang Zhang, Yanhong Xu, Yanru Ma, Yao Li, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yiliang Xiong, Ying He, Ying Zhou, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuduan Wang, Yue Gong, Yuhan Wu, Yuheng Zou, Yukun Li, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehua Zhao, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhiyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Zizheng Pan, Zongqing Yao, Bei Feng, Hui Li, J. L. Cai, Jiaqi Ni, Lei Xu, Meng Li, Ning Tian, R. J. Chen, R. L. Jin, S. S. Li, Shuang Zhou, Tianyu Sun, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xinnan Song, Xinyi Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, Dongjie Ji, Jian Liang, Jianzhong Guo, Jin Chen, Leyi Xia, Miaojun Wang, Mingming Li, Peng Zhang, Ruyi Chen, Shangmian Sun, Shaoqing Wu, Shengfeng Ye, T. Wang, W. L. Xiao, Wei An, Xianzu Wang, Xiaowen Sun, Xiaoxiang Wang, Ying Tang, Yukun Zha, Zekai Zhang, Zhe Ju, Zhen Zhang, and Zihua Qu. Deepseek-v3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code auto-completion systems. *arXiv preprint arXiv:2306.03091*, 2023.

Lingkun Long, Rubing Yang, Yushi Huang, Desheng Hui, Ao Zhou, and Jianlei Yang. Slim-infer: Accelerating long-context llm inference via dynamic token pruning. *arXiv preprint arXiv:2508.06447*, 2025.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, Zhiqi Huang, Huan Yuan, Suting Xu, Xinran Xu, Guokun Lai, Yanru Chen, Huabin Zheng, Junjie Yan, Jianlin Su, Yuxin Wu, Neo Y. Zhang, Zhilin Yang, Xinyu Zhou, Mingxing Zhang, and Jiezhong Qiu. Moba: Mixture of block attention for long-context llms. *arXiv preprint arXiv:2502.13189*, 2025.

Microsoft. Microsoft 365 Copilot, 2025. URL `https://www.microsoft.com/microsoft-365/copilot`.

Ken M. Nakanishi. Scalable-softmax is superior for attention. *arXiv preprint arXiv:2501.19399*, 2025.

OpenAI. File uploads FAQ, 2025a. URL https://help.openai.com/en/articles/8555545-file-uploads-faq.

OpenAI. Introducing deep research | OpenAI, 2025b. URL https://openai.com/index/introducing-deep-research/.

Perplexity AI. File uploads, 2025. URL https://www.perplexity.ai/help-center/en/articles/10354807-file-uploads.

Qwen Team. Qwen3.5. https://qwenlm.github.io/, 2026.

Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification. *arXiv preprint arXiv:2106.02034*, 2021.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. *arXiv preprint arXiv:2207.07061*, 2022.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Zhenmei Shi, Yifei Ming, Xuan-Phi Nguyen, Yingyu Liang, and Shafiq Joty. Discovering the gems in early layers: Accelerating long-context llms with 1000x input token reduction. *arXiv preprint arXiv:2409.17422*, 2024.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (learn at test time): Rnns with expressive hidden states. *arXiv preprint arXiv:2407.04620*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Hanrui Wang, Zhekai Zhang, and Song Han. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021. arXiv:2012.09852.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. *arXiv preprint arXiv:2410.10819*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Dongjie Yang, XiaoDong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. Pyramidinfer: Pyramid kv cache compression for high-throughput llm inference. *arXiv preprint arXiv:2405.12532*, 2024a.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. *arXiv preprint arXiv:2412.06464*, 2024b.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. *arXiv preprint arXiv:2406.06484*, 2024c.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y. X. Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention. *arXiv preprint arXiv:2502.11089*, 2025.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *arXiv preprint arXiv:1910.07467*, 2019.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2o: Heavy-hitter oracle for efficient generative inference of large language models. *arXiv preprint arXiv:2306.14048*, 2023.

A EFFICIENT KERNELS FOR TOKENGATE

Token-gated attention (Eq. 8) differs from standard causal attention in two ways: a per-query temperature $\tau_i^{\ell h}$ and a per-key, per-head gate term added to the attention logits. The temperature is applied by scaling each query before the attention call, so it needs no kernel support. The gate term needs a per-key bias, which standard fused kernels do not accept, and unfused or score-modifying kernels are substantially slower. We therefore patch FlashAttention-2 (Dao, 2023) to read a per-key, per-head bias vector, which it adds to each tile of attention scores before the online softmax. This keeps the fused structure of the kernel: the bias adds one vector load per key block, and a biased call costs 1.0–$1.1\times$ a plain one. The kernel supports head dimensions 128 (Qwen3) and 256 (Qwen3.5).

Training. The same kernel provides the backward pass: the gradient of the bias of key j is the sum over queries of the gradient of its attention scores, which the kernel accumulates alongside the usual gradients, so the task loss reaches the scoring network through the fused kernel. The two targets of each training step (Eq. 10) are packed into one variable-length call. For Qwen3.5, whose Gated DeltaNet layers are not supported by this path, we train with FlexAttention (Dong et al., 2024) and add the bias as a score modification, but more optimized kernels are possible.

Inference. Layers at and below the gate use the standard fused kernel (cuDNN on A100). The pruned forward of the post-gate layers is captured as one CUDA graph per padded length: the kept sequence is padded to a multiple of 64, and pad keys receive a bias of -10^9 so the result is exact. The gather of the KV cache of layers $\ell \leq L$ to the kept tokens (Appendix B) runs on a separate stream, concurrently with the post-gate layers. During decoding, the gate term of each kept key is fixed, so it is folded once into an additive attention mask per post-gate layer and decoding uses standard attention.

B DECODING WITH TOKENGATE

The layers above the gate, $\ell > L$, never compute KV entries for pruned tokens, so during decoding each new token attends there only to the kept tokens S and the tokens generated so far. The layers at and below the gate, $\ell \leq L$, process the full context during prefill, so by default their KV cache holds every context token, and decode tokens attend to the full context in these layers and to the pruned context above them. Optionally, TOKENGATE can also prune the cache of layers $\ell \leq L$.

In the post-gate layers, decode tokens use the same token-gated softmax as prefill (Eq. 8). The gate term $\mathrm{softplus}(\gamma^{\ell h})\, b_j$ of each kept key is computed once at prefill and stored with the cache as a constant per-key bias, and each generated token is a non-prunable token with gate term 0 (Eq. 3). The temperature $\tau_i^{\ell h}$ of a decode query follows Eq. 7, with m_i its rank among the kept and generated tokens and p_i its original position; generated tokens continue from the original position of the last prompt token.

C DETAILS OF EXPERIMENTAL SETUP

C.1 DATASETS

All datasets are evaluated on Qwen3-1.7B with greedy decoding for up to 1024 new tokens. Each model is fine-tuned on a training set that is disjoint from its evaluation set.

RepoBench-P. RepoBench-P (Liu et al., 2023) is a next-line code completion task. We use the LongBench (Bai et al., 2024) version, whose prompts contain cross-file code context and an in-file prefix, and the model completes the next line. We evaluate on all 500 samples (mean context 11.0k tokens, up to 39k) and report edit similarity, $1 - \mathrm{Levenshtein}/\max(\mathrm{length})$, on the first generated line. For training, we use 14.9k samples from the RepoBench v1.1 Python and Java cross-file splits, deduplicated against the evaluation set.

2WikiMultihopQA. 2WikiMultihopQA (Ho et al., 2020) is a multi-hop question answering task over several Wikipedia passages; we use the LongBench version. We evaluate on all 200 samples (mean context 7.5k tokens) and report token-level F1 against the gold answers. For training, we use 20k 2WikiMultihopQA questions with contexts rebuilt from full Wikipedia articles and decontaminated against LongBench.

MultiNews. MultiNews (Fabbri et al., 2019) is a multi-document news summarization task; we use the LongBench version. We evaluate on all 200 samples (mean context 2.7k tokens, up to 14k) and report ROUGE-L. For training, we use 44.8k samples from the MultiNews training split in the LongBench prompt format.

BioDEX. BioDEX-Reactions (D'Oosterlinck et al., 2023) is an adverse drug reaction extraction task over full-text biomedical papers: given a paper, the model lists the reported adverse reactions as MedDRA terms. We evaluate on 500 samples drawn from the official validation split with articles between 2k and 39k tokens (mean 4.5k), and report set F1 between predicted and gold reactions, counting a predicted reaction as correct if it matches a gold term up to substring containment. For training, we use 9.8k papers from the official training split.

NIAH. NIAH (Kamradt, 2023) is needle-in-a-haystack retrieval at 16k tokens. The haystack is built from Paul Graham essays, and the needle, "The best thing to do in *city* is *activity*.", is inserted at a sentence boundary at a given depth; the model is then asked for the best thing to do in that city. The 200 evaluation samples cover 20 held-out cities at 10 depths evenly spaced from 0 to 1, and we report accuracy, i.e., whether the gold activity appears in the answer. For training, we use 12k generated samples with haystacks of 14k–16.4k tokens and uniformly random depths; the essays and cities are disjoint from the evaluation set.

C.2 BASELINES

All baselines run on the same SFT model, fine-tuned on each dataset with LoRA (r=16, α=32) on the query and value projections of all layers. The pruning baselines keep all non-document tokens and prune only document tokens, with the same kept-token budgets K as TOKENGATE (64–8192). We briefly describe each baseline below, and the configurations used. We run each method, applying its optimization for prefill-only.

FastKV (Jo et al., 2025). At a token-selective propagation layer, FastKV scores tokens by the attention they receive from a window of the last queries, max-pools the scores, and propagates only the top-K tokens to deeper layers. We use the paper's observation window of 8 tokens and pooling kernel of 7. We run it at layer 13 by default, following the paper and its hyper-parameter search, which chooses relative depth of 46% for its models. We additionally run the baseline at layer 1, the layer at which we apply TOKENGATE, to compare performance at a matched layer index.

SlimInfer (Long et al., 2025). SlimInfer prunes hidden states in several stages: at layers 8, 17 and 25 it keeps $3K$, $1.5K$ and K tokens, scoring 64-token blocks by the attention of the last 4 queries. We use the paper's block size, kernel and layer schedule, and its budget ratios, following the paper's hyperparameter search.

GemFilter (Shi et al., 2024). GemFilter runs the first layers on the full prompt, selects the top-K tokens by the last query's attention scores at a filter layer, and then runs a fresh forward pass over only the selected tokens. We use filter layer 12, at the same relative depth (0.4) as recommended by the original paper, and the official pooling kernel.

DASH (Chen et al., 2026). DASH halts tokens whose representations have stabilized: at layer 11 ($0.4\times$ the number of layers, the official default), it ranks tokens by the change in their hidden state from the previous layer and drops the most stable ones, keeping K document tokens.

FlexPrefill (Lai et al., 2025). FlexPrefill is a sparse-attention method: it keeps all tokens but computes a query-adaptive sparse attention pattern during prefill, controlled by a cumulative-attention threshold γ. We use the official kernel and its recommended hyperparameters (divergence threshold τ=0.1, which decides per head whether to use a query-aware or a vertical-slash pattern; minimum budget 1024; block size 128) and sweep $\gamma \in \{0.1, 0.2, \ldots, 0.9, 0.95, 0.97\}$, extending the range until accuracy degrades.

MInference (Jiang et al., 2024). MInference uses per-head sparse attention patterns found by an offline search, which for Qwen3-1.7B selects vertical-and-slash patterns for all heads. We run the official implementation with patterns searched on Qwen3-1.7B, and sweep its budget by scaling the vertical and slash sizes of the searched configuration by $\{0.25, 0.5, 1, 2, 4\}$, which keeps the searched pattern shape.

Implementation and timing. All methods run the same fine-tuned backbone in bf16 with the LoRA adapter folded into the weights, and all dense decoder layers are `torch.compile`d. Dense causal attention uses the same fused kernel in every lane (cuDNN on the A100; FlashAttention-2 on Qwen3.5-2B); the only kernel that differs is in the TOKENGATE's post-gate layers, which require a per-key bias and use a FlashAttention-2 variant at 1.0–$1.1\times$ the cost of a plain call (Appendix A). FastKV and GemFilter run through the compiled layers directly. The official SlimInfer and DASH prefills bypass the compiled layer forward, so we reimplemented both in compile-friendly form and verified that they select the same tokens as the official code (DASH bit-identical; SlimInfer identical selection), which made them 1.2–$2.4\times$ and 1.25–$1.56\times$ faster; we report the faster versions. MInference is compiled around its sparse kernel with identical accuracy, and FlexPrefill runs eager, its fastest configuration, since its data-dependent block selection triggers recompilation under `torch.compile`. The pruned forward of the TOKENGATE and of FastKV is captured as one CUDA graph per padded length; the remaining baselines have no fixed-shape pruned phase to capture. Both FastKV and the TOKENGATE optionally prune the KV cache of the layers below their pruning layer to the selected tokens once prefill completes, so that decoding holds K entries at every layer; we enable this setting for both in all experiments (Appendix B). Inference timings are measured at batch size of 1 on one A100.

C.3 TRAINING HYPERPARAMETERS

Table 1 lists the hyperparameters used to train the TOKENGATE models and the SFT baseline on each dataset. Both use the same LoRA configuration, learning rate, schedule and training data; they differ only in the optimizer of the TOKENGATE parameters and in the token budgets sampled during training.

Hyperparameter	Value
Base model	Qwen3-1.7B, bf16
LoRA	r=16, α=32 on query and value projections, all layers
Optimizer	AdamW; Muon (Jordan et al., 2024) for the gate matrices (TOKENGATE only)
Learning rate	10^{-4}, linear warmup over 1000 steps, then constant
Weight decay	0.1
Gradient clipping	global norm 1.0
Batch size	1 sequence per GPU
Training context	documents of 2k–16k tokens
Gate layer (TOKENGATE only)	$L = 1$
Budgets per step (TOKENGATE only)	$K \sim$ log-uniform, plus full context

Table 1: Training hyperparameters for TOKENGATE and the SFT baseline, shared across all five datasets.

C.4 EXTENSION TO ALTERNATIVE HYBRID-ATTENTION ARCHITECTURES

Qwen3.5 architecture. Qwen3.5-2B (Qwen Team, 2026) is a hybrid-attention model with 24 layers. Every fourth layer, $\ell \in \mathcal{A} = \{3, 7, 11, 15, 19, 23\}$, is a full-attention layer whose output is multiplied by a learned sigmoid output gate. All other layers are Gated DeltaNet layers (Yang et al., 2024b), which replace softmax attention with a linear recurrence over a fixed-size state and therefore have no attention weights.

Gate module. We place the TOKENGATE at layer $L \in \{1, 2, 3\}$, where layers 1 and 2 are Gated DeltaNet layers and layer 3 is the first full-attention layer. The scoring network (Eq. 3) and the top-K selector (Eq. 4) are unchanged. The only difference is the token mixer of layer L: the scoring network reads the normalized residual after layer L's mixer,

$$\tilde{a}_j^L = \mathrm{LN}_2\big(h_j^L + \mathrm{Mixer}_L(\mathrm{LN}_1(h^L))_j\big), \qquad \mathrm{Mixer}_L \in \{\mathrm{GatedDeltaNet}_L,\ \mathrm{Attn}_L\}, \tag{11}$$

which for a Gated DeltaNet layer is the output of its recurrence rather than of self-attention.

Token-gated attention. Since Gated DeltaNet layers have no attention logits to bias, token-gated attention (Eq. 8), including the gains $\gamma^{\ell h}$ and the temperature, is applied only in the full-attention

layers above the gate, $\ell \in \mathcal{A},\ \ell > L$. The Gated DeltaNet layers above the gate run unchanged on the kept tokens S, starting from an empty state, so that their recurrent state is built from the kept tokens only:

$$h_S^\ell = \begin{cases} \text{token-gated attention layer (Eq. 8)} & \ell \in \mathcal{A},\ \ell > L, \\ \mathrm{GatedDeltaNet}_\ell\big(h_S^{\ell-1}\big) & \ell \notin \mathcal{A},\ \ell > L. \end{cases} \tag{12}$$

Kept tokens keep their original positions for rotary embeddings in the full-attention layers.

Decoding. The recurrent state of a Gated DeltaNet layer summarizes all tokens it has seen in a fixed-size matrix and cannot be pruned per token. The Gated DeltaNet layers $\ell \leq L$ therefore decode from a state built over the full prompt, while the KV cache of full-attention layers are pruned, as in Appendix B.

Training. As for Qwen3, we train the TOKENGATE jointly with LoRA (r=16) on the query and value projections, which in Qwen3.5 exist only in the full-attention layers; the Gated DeltaNet layers are not adapted.

D ADDITIONAL EXPERIMENTAL RESULTS

D.1 TRAINING TIME

	Full train time			Per-step time		
Dataset	SFT	Ours	Ratio ↓	SFT	Ours	Ratio ↓
Repo	1.7h	1.1h	0.6×	0.50s	0.64s	1.3×
2Wiki	3.8h	4.0h	1.1×	0.34s	0.43s	1.3×
MNews	1.3h	1.8h	1.4×	0.21s	0.27s	1.3×
BioDEX	1.2h	1.0h	0.8×	0.23s	0.30s	1.3×
NIAH	2.8h	1.0h	0.4×	0.99s	1.24s	1.3×
Mean	2.2h	1.8h	0.8×	0.45s	0.58s	1.3×

Table 2: Training time for SFT baseline and model trained with TOKENGATE for Qwen3-1.7B on one A100. Ratio shows Ours / SFT.

D.2 MAIN RESULTS WITHOUT AXIS LIMITS

Below, Figure 6 shows the main results (Figure 3), but without axis limits and with every point shown.

D.3 EXTRAPOLATING TO LONGER CONTEXT

	NIAH 16k	NIAH 32k	NIAH 64k	NIAH 128k
TTFT speedup	11.3×	12.8×	13.0×	13.0×
Peak memory reduction	3.42×	3.43×	3.43×	3.43×

Table 3: TOKENGATE TTFT speedup and peak prefill memory reduction at no accuracy degradation over full context for the NIAH-16k, 32k, 64k and 128k test sets with Qwen3-1.7B.

We study the ability of the trained TOKENGATE to extrapolate to inference settings with longer context. We take the fine-tuned base model and TOKENGATE model, both trained on the 16k-NIAH task using Qwen1.7B, and we evaluate prefill performance on the NIAH task with 32k, 64k and 128k-length test sets. Table 3 shows the TTFT speedups and peak memory reductions of the TOKENGATE model at no accuracy degradation over the full at full-context accuracy on each dataset. The results demonstrate that TOKENGATE extrapolates across varying context lengths at inference, retaining accuracy with greater than 11.3 - 13× speedups. We observe that the TTFT speedups of the trained TOKENGATE model are highest at the longest context-length ranges, where trainable

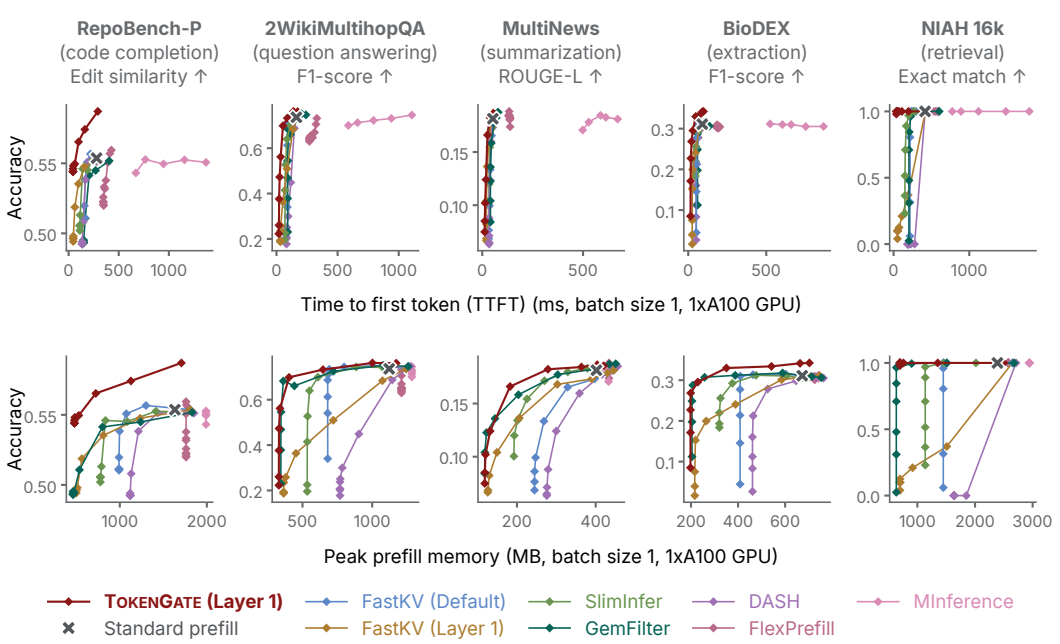


Figure 6: Main results (Figure 3) without x or y limits: accuracy vs. time-to-first-token (top) and accuracy vs. peak GPU prefill memory (bottom) with Qwen3-1.7B, all points shown.

token pruning can significantly reduce costs. Meanwhile, the peak memory reduction is consistent across all length ranges, as expected, since peak memory is dominated by memory footprint of the standard forward that occurs without pruning in layers below the gate.

D.4 Ablating the TokenGate Design

In this section we ablate the TokenGate design (Section 4). First, Figure 7 (left) compares the accuracy vs TTFT performance of TokenGate on BioDEX with Qwen3-1.7B against three variants: (a) removing the full-context loss term $L_{K_{\text{full}}}$ from the per-step loss (Eq. 10), (b) removing the temperature scaling term during token-gated attention, and (c) removing the head-wise scaling. The results demonstrate that training without the full-context loss most significantly degrades inference accuracy at full-context, and removing the temperature scaling and head-wise gain also degrades accuracy, leading to sub-optimal accuracy vs TTFT.

We additionally analyze layer selection, a hyperparameter for TokenGate. Figure 7 (right) shows performance on BioDEX of Qwen3-1.7B models trained with the TokenGate at Layer 0, 1, and 2. Notably, the model with gating at Layer 0 is Pareto-suboptimal relative to the Layer 1 and 2 gate models, suggesting that the Layer 0 post-attention residuals provide insufficient signal for the TokenGate scoring network. On the other hand, the Layer 1 and Layer 2 TokenGate models both lie along the Pareto frontier, suggesting that by Layer 1, the post-attention residuals passed to the gate module are sufficiently rich to learn an effective scoring network for trainable token pruning.

E Visualizing Kept Tokens on Real Samples

Figures 8–12 show which context tokens TokenGate keeps on one real evaluation prompt per dataset, using the layer-1 checkpoints of Figure 3 (Qwen3-1.7B). To avoid selecting favorable examples, each prompt is the first of the seeded test draw used by the benchmarks. The strip at the top of each figure spans the whole prompt: the context in light grey, each kept token in red, and the non-prunable tokens $\mathcal{N}$ of Section 4 (instruction, question and answer template; the scaffold) in darker grey at either end. The box below shows the part of the context under the bracket, chosen as the window of that length holding the most kept tokens. For each prompt we show the smallest

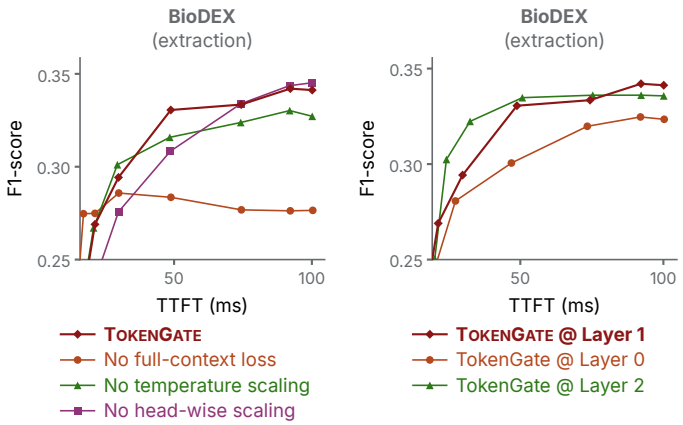


Figure 7: Left: token-gated forward and training ablations on BioDEX (Qwen3-1.7B, gate at layer 1): F1 vs. TTFT (batch 1, A100) for the full TOKENGATE recipe against three single-change variants: no full-context loss (no full-context term in Eq. 10), no temperature scaling (Eq. 7 removed), and no head-wise scaling (raw bias, with no normalization or gain; Eq. 3 removed). Right: gate-layer ablation on BioDEX (Qwen3-1.7B): F1 vs. TTFT (batch 1, A100) with the TOKENGATE placed at layer 0, 1, or 2.

budget $K \in \{64, 128, 256, 512, 1024\}$ at which the greedy answer equals that of the same model with the full context, or $K = 256$ where no budget does. The shown budgets keep between 0.4% (NIAH) and 13% (MultiNews) of the context.

Selection is global, not local. In every sample the kept tokens fall in all ten deciles of the context, so the gate does not reduce to a recency window or to the start of the document. At the same time it concentrates the budget where the task needs it: on NIAH, 21 of the 24 tokens of the needle sentence, which sits at 78% depth in a $15{,}843$-token haystack, are among the 64 kept tokens, and the model answers exactly. The same holds for all eight NIAH prompts we checked, whose answer at $K = 64$ matches the reference.

What is kept, and what it costs. Table 4 summarizes the five examples. On 2WikiMultihopQA, BioDEX and NIAH the gate keeps content: entities and years, the terms that describe a medical case, and, where the reference answer occurs verbatim in the context, a large share of its tokens (all of the needle's answer on NIAH). On RepoBench-P the selection is instead dominated by code structure: most kept tokens contain no letter, and they are spread thinly over the retrieved files, whereas on NIAH most kept tokens have a kept neighbor and form contiguous spans. This suggests that for code completion the post-gate layers use the kept tokens more as anchors on the structure of the retrieved files than as a copy of their content. The examples also show where pruning costs accuracy: on MultiNews the gate keeps mostly function words and under-keeps capitalized tokens, and the summary at $K = 256$ reports the right events but gets the names and places wrong, and for drug reaction extraction on BioDEX the pruned model recovers one of the two reactions found with the full context.

Dataset	What is kept	Failure mode
RepoBench-P	code syntax rather than identifiers: commas and empty argument lists "()" dominate, line breaks are avoided, and kept tokens are isolated rather than spans	wrong line, as with the full context
2WikiMultihopQA	entities and dates: names and years are over-kept and function words avoided; on the shown prompt, the answer phrase ("Film Award for . . . Director")	a different award at small budgets, exact at the shown one; multi-hop answers need the largest budgets to match the full context
MultiNews	mostly function words and scattered event words ("charged", "president"); names and places are under-kept	right events, but wrong name, occupation and place, consistent with the under-kept names
BioDEX	terms of the diagnosis and outcome ("septic shock", "failure") and the connectives of lists ("and", "or"); numbers are almost never kept	misses reactions, but so does the full-context model, which bounds the score
NIAH 16k	the needle sentence as one contiguous span, plus rare subwords elsewhere ("-haired", "Lisp", ".html")	none: exact at every budget

Table 4: Summary of what TOKENGATE keeps and how its answers fail. Examples refer to the prompts of Figures 8–12 at their shown budgets; the patterns hold over the first eight prompts of each test draw.

RepoBench-P



context: 20,369 tokens, 256 kept by the gate

Context excerpt under the bracket (445 of 20,369 tokens), kept tokens highlighted, ↵ marks a line break

... etSocketAddress("graphite1.osumer.tk", 2003)); ↵ reporter = GraphiteReporter.forRegistry(metrics) ↵ .prefixedWith("osumer") ↵ .convertRatesTo(TimeUnit.SECONDS) ↵ .convertDurationsTo(TimeUnit.MILLISECONDS) ↵ .filter(new NotReportedFilter()) ↵ .build(graphite); ↵ reporter.start(1, TimeUnit.MINUTES); ↵ metrics.register("jvm.gc", new GarbageCollectorMetricSet()); ↵ metrics.register("jvm.threads", new CachedThreadStatesGaugeSet(10, TimeUnit.SECONDS)); ↵ metrics.register("jvm.memory", new MemoryUsageGaugeSet()); ↵ readDumps(); ↵ init = true; ↵ reportEvent("debugManagerInit"); ↵ if (osumerVersion != null) { ↵ reportEvent("version", osumerVersion.replaceAll("\\.", "-")); ↵ } ↵ String mac = obtainMac(); ↵ if (mac != null) { ↵ reportEvent("activeUsers", mac); ↵ } ↵ } ↵ private static boolean isVMMac(byte[] mac) { ↵ if(null == mac) return false; ↵ byte invalidMacs[][] = { ↵ {0x00, 0x05, 0x69}, //VMWare ↵ {0x00, 0x1C, 0x14}, //VMWare ↵ {0x00, 0x0C, 0x29}, //VMWare ↵ {0x00, 0x50, 0x56}, //VMWare ↵ {0x08, 0x00, 0x27}, //Virtualbox ↵ {0x0A, 0x00, 0x27}, //Virtualbox ↵ {0x00, 0x03, (byte)0xFF}, //Virtual-PC ↵ {0x00, 0x15, 0x5D} //Hyper-V ↵ }; ↵ for (byte[] invalid ...

Scaffold before and after the context (always kept)

Output only the single next line of code that continues the file below. Do not explain or use markdown; output exactly one line. [context] ... ed()) { ↵ downloadAction = 1; ↵ } else if (rdBtnDwnFile.isSelected()) { ↵ downloadAction = 2; ↵ targetFileOrFolder = ""; ↵ } else if (rdBtnDwnFolder.isSelected()) { ↵ downloadAction = 3; ↵ targetFileOrFolder = ""; ↵ } ↵ MethodResult<Integer> result = null; ↵ try { ↵ result = d.addQueue(url, downloadAction, targetFileOrFolder); ↵ } catch (RemoteException e) { ↵ e.printStackTrace();

Reference	DumpManager.addDump(new DebugDump(null, "Process download action", "Request daemon to add queue", "(Method End)", "Unable to request daemon to add queue", false, e));
TokenGate, K = 256	ProgressController.addToErrorLog("Error adding queue: " + OsuDownloader.getDownloadError(e.getMessage()));
TokenGate, full context	if (e instanceof NoSuchBuildNumberException) {

Figure 8: Context tokens kept by TOKENGATE on one RepoBench-P test prompt (Appendix E), with the reference answer and the model's greedy answers at the shown budget and with the full context. Kept tokens are highlighted and pruned tokens are grey.

2WikiMultihopQA



context: 5,352 tokens, 512 kept by the gate

Context excerpt under the bracket (449 of 5,352 tokens), kept tokens highlighted, ↵ marks a line break

... age 1: ↵ Ian Barry (director) ↵ Ian Barry is an Australian director of film and TV. ↵ Select credits ↵ Waiting for Lucas (1973) (short) ↵ Stone (1974) (editor only) ↵ The Chain Reaction (1980) ↵ Whose Baby? (1986) (mini-series) ↵ Minnamurra (1989) ↵ Bodysurfer (1989) (mini-series) ↵ Ring of Scorpio (1990) (mini-series) ↵ Crimebroker (1993) ↵ Inferno (1998) (TV movie) ↵ Miss Lettie and Me (2002) (TV movie) ↵ Not Quite Hollywood: The Wild, Untold Story of Ozploitation! (2008) (documentary) ↵ The Doctor Blake Mysteries (2013) ↵ Passage 2: ↵ Dante Lam ↵ Dante Lam Chiu-yin (simplified Chinese: 林超贤; traditional Chinese: 林超賢; pinyin: Lín Chāoxián, born 1 July 1964) is a Hong Kong filmmaker, actor and action choreographer. ↵ Background ↵ He was trained in the tradition of John Woo as an assistant director and worked as an actor and producer. He often writes and supervises his own choreography. In 2008 he won the Hong Kong Film Award for Best Director for his work on Beast Stalker.His 2018 film Operation Red Sea is the second-highest-grossing Chinese film of all time and 9th in the international box office list of 2018. It won him the Hundred Flowers Award for Best Director and the award for Best Action Choreography at the 38th Hong Kong Film Awards.In the aftermath of the 2019–20 Hong Kong protests, Lam was contracted by the Hong Kong Police Force (HKPF) to produce a video, Guarding Our City, intended to help rehabilitate the police force's public image. The 15-minute video was released on 23 January 2021. ↵ Filmography ↵ ...

Scaffold before and after the context (always kept)

Answer the question using the Wikipedia passages below. ↵ Question: Which award the director of film Tiramisu (2002 Film) earned? ↵ Passages: [context] . ↵ Answer in a few words. ↵ Answer:

Reference	Hong Kong Film Award for Best Director
TokenGate, K = 512	Hong Kong Film Award for Best Director
TokenGate, full context	Hong Kong Film Award for Best Director

Figure 9: Context tokens kept by TOKENGATE on one 2WikiMultihopQA test prompt (Appendix E), with the reference answer and the model's greedy answers at the shown budget and with the full context. Kept tokens are highlighted and pruned tokens are grey.

MultiNews



context: 1,947 tokens, 256 kept by the gate

Context excerpt under the bracket (339 of 1,947 tokens), kept tokens highlighted, ↵ marks a line break

... Montreal the next day without a disguise, demanded money and left with an undisclosed amount of cash. ↵ Police arrested Vogelsang on Saturday at a hotel in the same area as the banks, just after 3 a.m. CT. ↵ Vogelsang and his wife moved from Winnipeg to Nelson, B.C., after he left his post at Red River College. ↵ After spending 24 years together, Vogelsang and his wife separated in 2015 and he moved back to Winnipeg. She filed for divorce last summer. ↵ Financially squeezed ↵ According to court documents, Vogelsang and his wife had owned three properties in Nelson and sold them for a total loss of $85,000. ↵ Vogelsang was supposed to take over fully last September after removing his ex-wife's name from the paperwork. Vogelsang was a journalism instructor at Red River College from 2002 to 2011 after serving as news director and a longtime sports anchor at CTV. (Steve Vogelsang/Facebook) They also owned a home in Winnipeg, whichwas supposed to take over fully last September after removing his ex-wife's name from the paperwork. ↵ The documents say that didn't happen. Instead, she alleges he forged her signature and renewed the mortgage on the home in both their names. ↵ Vogelsang wasn't able to make the mortgage payments and put the house up for sale, but it went into foreclosure in April ...

Scaffold before and after the context (always kept)

Write a summary of the news articles below. ↵ Document: [context] Summary:

Reference	A former well-known Canadian TV sports anchor is in jail after police say he robbed two banks in as many days in the western province of Alberta last week. Police in the small city of Medicine Hat say Steve Vogelsang robbed a Royal Bank of Canada on Thursday, Oct. 19, and then turned around the next ...
TokenGate, K = 256	A former Winnipeg teacher is facing charges of two separate bank robberies, and he's not happy about it. Steve Vogels, 50, was charged with two counts of robbery and theft after he was arrested in October for stealing $100,000 from a bank in Winnipeg, the CBC reports. The robbery occurred in 2015, a ...
TokenGate, full context	A former college instructor and sports anchor in Winnipeg is accused of robbing two banks in Alberta, and he's in custody. Police say Stephen Vogelsang, 53, was arrested Saturday in Medicine Hat, about three hours southeast of Calgary, after a man robbed a Royal Bank of Canada on Oct. 19 and a Bank ...

Figure 10: Context tokens kept by TOKENGATE on one MultiNews test prompt (Appendix E), with the reference answer and the model's greedy answers at the shown budget and with the full context. Kept tokens are highlighted and pruned tokens are grey.

BioDEX



context: 4,100 tokens, 256 kept by the gate

Context excerpt under the bracket (296 of 4,100 tokens), kept tokens highlighted, ↵ marks a line break

TITLE: ↵ A fatal case of descending necrotizing mediastinitis as a complication of odontogenic infection. A case report. ↵ ABSTRACT: ↵ This paper presents the case of a 26-year-old female patient in whom descending necrotizing mediastinitis (DNM) developed as a complication of an odontogenic purulent infection of the mouth. Despite the efforts of a multidisciplinary treatment team, the patient died with symptoms of septic shock and multiple organ failure. According to the literature, and as confirmed by our own observations, successful treatment requires early tomographic diagnosis, radical surgery, combination antibiotic therapy, and intensive care. ↵ TEXT: ↵ Introduction ↵ Purulent conditions of the mediastinum and lungs spread by continuity through the interfascial spaces of the neck are among the rarest complications of odontogenic inflammatory processes of the oral cavity. Descending necrotizing mediastinitis (DNM) is a noteworthy example of such infection, originating from inflammatory conditions within the oral cavity and teeth [1]. ↵ Descending necrotizing mediastinitis develops in patients between 30 and 50 years old and occurs six times more frequently among men. The symptoms characteristic of DNM include strong retrosternal pain (intensifying during breathing and coughing), rapid breathing, tachycardia, increased body temperature, and inflammatory infiltration of the neck and chest. Treatment consists in combining broad-spectrum antibiotic therapy with radical dis ...

Scaffold before and after the context (always kept)

Answer the question using only the biomedical article below. Answer with the adverse drug reaction terms only, separated by commas. ↵ Question: What adverse drug reactions did the patient experience? ↵ Article: [context] . ↵ Answer:

Reference	Diabetes mellitus, Drug ineffective, Mediastinitis, Multi-organ failure, Septic shock
TokenGate, K = 256	Drug ineffective
TokenGate, full context	Drug ineffective, Multiple organ dysfunction syndrome

Figure 11: Context tokens kept by TOKENGATE on one BioDEX test prompt (Appendix E), with the reference answer and the model's greedy answers at the shown budget and with the full context. Kept tokens are highlighted and pruned tokens are grey.

NIAH 16k



context: 15,843 tokens, 64 kept by the gate

Context excerpt under the bracket (341 of 15,843 tokens), kept tokens highlighted, ↵ marks a line break

... to do in Hanoi is share a pot of tea along the canal towpath when the bells ring noon. \[8\] There are borderline cases where it's hard to say which category a test falls in. For example, is raising venture capital like college admissions, or is it like selling to a customer? \[9\] Note that a good test is merely one that's unhackable. Good here doesn't mean morally good, but good in the sense of working well. The difference between fields with bad tests and good ones is not that the former are bad and the latter are good, but that the former are bogus and the latter aren't. But those two measures are not unrelated. As Tara Ploughman said, the path from good to evil goes through bogus. \[10\] People who think the recent increase in [economic inequality](ineq.html) is due to changes in tax policy seem very naive to anyone with experience in startups. Different people are getting rich now than used to, and they're getting much richer than mere tax savings could make them. \[11\] Note to tiger parents: you may think you're training your kids to win, but if you're training them to win by hacking bad tests, you are, as parents so often do, training them to fight the last war. **Thanks** to Austen Allred, Trevor Blackwell, Patrick Collison, Jessica Livingston, Robert Morris, and Harj Taggar for reading drafts of this. To celebrate Airbnb's IPO and to help future founders, I thought it might be useful to explain what was special about Airbnb. What was special about the Airbnbs ...

Scaffold before and after the context (always kept)

Answer the question using only the document below. Keep your answer short and direct. ↵ Question: What is the best thing to do in Hanoi? ↵ Document: [context] Answer:

Reference	share a pot of tea along the canal towpath when the bells ring noon
TokenGate, K = 64	share a pot of tea along the canal towpath when the bells ring noon
TokenGate, full context	share a pot of tea along the canal towpath when the bells ring noon

Figure 12: Context tokens kept by TOKENGATE on one NIAH 16k test prompt (Appendix E), with the reference answer and the model's greedy answers at the shown budget and with the full context. Kept tokens are highlighted and pruned tokens are grey.