QuickSilver - Speeding up LLM Inference through Dynamic Token Halting, KV Skipping, Contextual Token Fusion, and Adaptive Matryoshka Quantization

Anonymous ACL submission

Abstract

Inference has become the dominant driver of resource consumption in large language model (LLM) deployments, often accounting for over 90% of total latency, energy use, and operational cost—*surpassing even the one-time expense of training*. While training-time efficiency has advanced considerably, runtime optimization remains a critical bottleneck, particularly under autoregressive decoding. Existing methods—such as pruning, quantization, early exits, and speculative decoding—often require retraining, architectural modifications, or a compromise in decoding compatibility. We present **QuickSilver**, a modular, token-level framework that enables *semantic adaptivity at inference time* without altering model weights or structure. QuickSilver integrates four synergistic mechanisms: (i) **Dynamic Token Halting**, which detects representational stability and halts further computation for semantically saturated tokens; (ii) **KV Cache Skipping**, which skips memory updates for halted tokens, streamlining attention layers and lowering runtime cost; (iii) **Contextual Token Fusion**, which identifies and merges similar tokens during inference, streamlining token flow and minimizing redundancy; and (iv) **Adaptive Matryoshka Quantization**, which dynamically adjusts token-level bit-widths for efficient quantization. Unlike speculative decoding or mixture-of-experts routing, QuickSilver operates entirely at runtime on frozen, dense models—requiring no auxiliary networks or retraining. Evaluated on GPT-2 and Llama-2 across WikiText-103 and C4, QuickSilver achieves up to **39.6% FLOP reduction** with minimal perplexity degradation ($\leq$0.2), offering a lightweight, plug-and-play path toward scalable, energy-efficient inference. To facilitate adoption and further research, we release our implementation publicly.[1]

[1] https://anonymous.4open.science/r/Quicksilver/

1 Inference-Time Speed: Why It Matters

LLMs now exceed human-level performance across many NLP tasks [OpenAI, 2023; Bubeck et al., 2023], yet inference, not training, has become the dominant bottleneck in deployment [Patterson and Gonzalez, 2021; Sanh et al., 2022]. Real-world usage patterns make inference responsible for over 90% of total energy and compute cost [Patterson et al., 2022; Desislavov et al., 2021], positioning inference-time optimization as a critical frontier.

User Interactivity. LLMs in real-time applications such as chatbots or translation tools demand sub-second token-level latency [Chen et al., 2023a; Levy et al., 2023]. Even slight delays degrade user experience [Shuster et al., 2022; Ni et al., 2022], while micro-optimizations can compound to improve responsiveness dramatically.

Scalability and Cost. Widespread LLM adoption stresses infrastructure. Faster inference boosts throughput without linearly scaling compute [Barham et al., 2022]. Strategies like early exits [Schwartz et al., 2020; Elbayad et al., 2020a], adaptive computation [Graves, 2016], and speculative decoding [Leviathan et al., 2022] reduce cost but often require retraining or architectural coordination.

Reasoning and Agents. Reasoning-heavy and agentic models rely on multi-step inference (e.g., Chain-of-Thought [Wei et al., 2022], Toolformer [Schick et al., 2023]), amplifying runtime overhead. Low-latency execution is vital for maintaining autonomy in dynamic environments.

Environmental Impact. Inference, executed millions of times daily, is the primary contributor to LLM carbon emissions [Patterson et al., 2022; Luccioni et al., 2022]. Runtime efficiency directly reduces energy footprint, enabling more sustainable AI deployment.

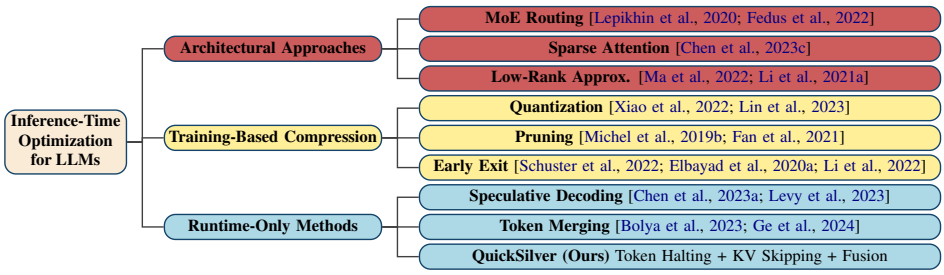


Figure 1: **Taxonomy of inference-time optimization techniques for LLMs**

Technique	Requires Retraining	Architecture Change	Runtime-Only	Token-Level	Stackable	Representative Works
Early Exit	✓ Yes	✎ Possibly	✗ No	❄ No	✎ Limited	[Schuster et al., 2022; Elbayad et al., 2020a; Li et al., 2022]
Mixture of Experts (MoE) Routing	✓ Yes	✓ Yes	✗ No	❄ No	❄ No	[Lepikhin et al., 2020; Fedus et al., 2022]
Speculative Decoding	✗ No	✎ Light Wrapper	✓ Yes	❄ No	✎ Limited	[Chen et al., 2023a; Levy et al., 2023]
Attention/Layer Pruning	✓ Yes	✎ Model Patch	✗ No	❄ No	✎ Limited	[Michel et al., 2019b; Fan et al., 2021]
Quantization	✓ Yes	✎ Compiler Patch	✗ No	❄ No	✓ Yes	[Xiao et al., 2022; Lin et al., 2023]
Token Merging	✎ Sometimes	✎ Light Patch	✓ Yes	✓ Yes	✓ Yes	[Bolya et al., 2023; Ge et al., 2024]
Sparse Attention	✓ Yes	✓ Yes	✗ No	❄ No	❄ No	[Chen et al., 2023c]
Low-Rank Approximation	✓ Yes	✓ Yes	✗ No	❄ No	✎ Limited	[Ma et al., 2022; Li et al., 2021a]
FlashInfer	✗ No	✓ Kernel Only	✓ Yes	❄ No	✓ Yes	[Ye et al., 2025]
LayerDrop	✓ Yes	✓ Yes	✗ No	❄ No	✎ Limited	[Fan et al., 2021]
QuickSilver (Ours)	✗ No	✗ No	✓ Yes	✓ Yes	✓ Yes	—

Table 1: Comparison of QuickSilver with existing inference-time acceleration methods across key attributes. Icons: ✓= Yes, ✗= No, ✎= Partial/Limited, ❄= Not Applicable.

Limitations of Prior Work. Inference accelerators such as **quantization** [Xiao et al., 2022], **pruning** [Michel et al., 2019b], **token merging** [Bolya et al., 2023], and **sparse attention** [Child et al., 2019; Chen et al., 2023c] often require *retraining*, introduce *architectural changes*, or *lead to degraded output quality* [Sanh et al., 2022; Bolya et al., 2023]. **Speculative decoding** [Chen et al., 2023a; Levy et al., 2023] adds a *verifier model* and coordination burden, while **layer-skipping methods** (e.g., FastBERT [Liu et al., 2020], PABEE [Zhou and et al., 2020]) rely on *task-specific supervision* and calibrated thresholds. Most techniques operate at the *layer or sequence level*, lacking **token-level control** and *runtime plasticity*. Their dependence on *static heuristics* or *auxiliary modules* limits deployability in *frozen* or *black-box* models. Figure 1 summarizes these **design constraints**, and Table 1 offers an attribute-level comparison across **retraining**, **architecture**, **runtime scope**, **granularity**, and **composability**.

Our Approach. We present **QuickSilver**, a runtime-only, zero-shot, model-agnostic framework for inference acceleration. It integrates **Dynamic Token Halting**, **KV Cache Skipping**, **Contextual Token Fusion**, and **Adaptive Matryoshka Quantization**—reducing per-step cost without retraining or altering model internals. QuickSilver complements step-reduction methods like speculative decoding, offering a composable solution for fast, sustainable, and scalable LLM inference.

2 QuickSilver - Design Details

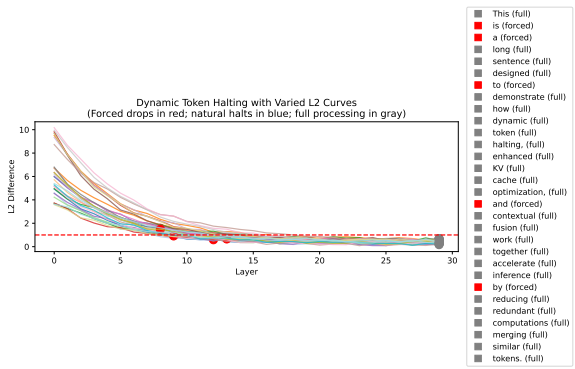
In this section, we present the core technical components of **QuickSilver**, a **runtime-only** framework composed of four **lightweight**, **modular** mechanisms: *Dynamic Token Halting*, *KV Cache Skipping*, *Contextual Token Fusion*, and *Adaptive Matryoshka Quantization*. Each targets a distinct *redundancy axis*—**temporal**, **memory**, **spatial**, and **precision**—and operates on *frozen transformer models* without retraining or architectural changes. We outline the **design motivations**, **decision signals**, and **implementation strategies** behind each module, and show how their composition enables **per-token compute reduction** while preserving *model fidelity* and *output quality*.

2.1 Dynamic Token Halting

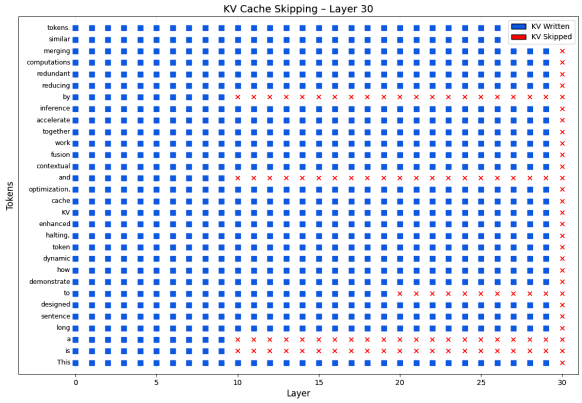
In standard Transformer inference, every token t is processed through all L layers—even when its representation has already stabilized. **Dynamic Token Halting (DTH)** addresses this inefficiency by detecting semantic convergence and terminating computation for such tokens early, layer-wise.

Semantic Convergence. Let $\mathbf{h}_t^{(\ell)} \in \mathbb{R}^d$ be the hidden state of token t at layer ℓ. DTH leverages this insight by computing, at each layer ℓ, two metrics for each token t:

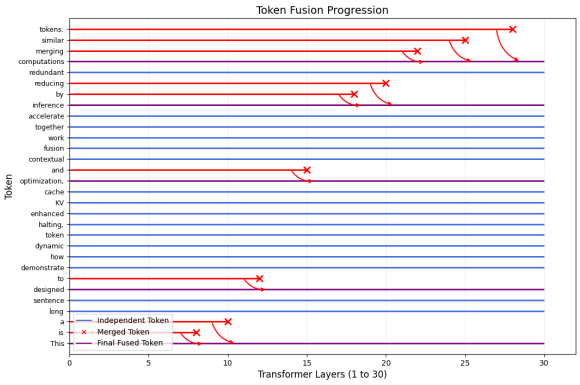
- **Layerwise Drift** $\Delta_t^{(\ell)} = \|\mathbf{h}_t^{(\ell)} - \mathbf{h}_t^{(\ell-1)}\|_2$
- **Token Entropy**



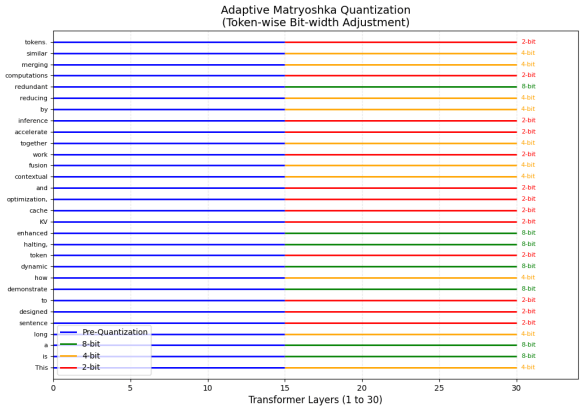
(a) Dynamic Token Halting. Tokens are halted once their representations converge, measured via L2 drift. Blue traces show halted tokens, red denotes forced halts, and gray denotes full propagation. This mechanism enables early exits per token, reducing unnecessary layer computation without retraining.



(b) KV Cache Skipping. Layer-token heatmap depicting where KV cache writes are performed (blue) versus skipped (red). Skipping is triggered by halting decisions, which reduces memory pressure in deeper layers and improves the efficiency of attention time.



(c) Contextual Token Fusion. Tokens with near-identical hidden states are progressively merged across layers. Red arcs indicate fusion events; purple paths represent fused token trajectories. This reduces effective sequence length while preserving syntactic and semantic alignment.



(d) Adaptive Matryoshka Quantization. At a mid-network layer (e.g., Layer 15), tokens are assigned bit-widths based on entropy: green = 8-bit, orange = 4-bit, red = 2-bit. This per-token precision scaling reduces memory and compute on low-entropy spans without degrading quality.

Figure 2: **Visualization of QuickSilver's token-level runtime mechanisms.** Each module adaptively adjusts inference based on semantic signals without altering weights, forming a unified framework that scales computation to information content.

$$\mathcal{H}(p_t^{(\ell)}) = -\sum_i p_t^{(\ell)}(i) \log p_t^{(\ell)}(i)$$

Halting Policy. A token halts when both signals fall below predefined thresholds:

$$H_t^{(\ell)} = \begin{cases} 0, & \text{if } \Delta_t^{(\ell)} < \tau_{\text{drift}} \text{ and } \mathcal{H}(p_t^{(\ell)}) < \tau_{\text{halt}} \\ 1, & \text{otherwise} \end{cases}$$

Here, $H_t^{(\ell)} = 0$ indicates halting. This decision is based on two criteria: low representational drift and low predictive entropy, which jointly ensure convergence in both the latent space and output confidence. Once halted, the token is excluded from computation in layers $\ell + 1$ through L.

Override Logic. DTH supports flexible overrides:

- *Forced Halting:* Token is halted regardless of $\Delta_t^{(\ell)}$ (e.g., latency-critical contexts).
- *Full Processing:* Token bypasses halting (e.g., special tokens or domain-sensitive terms).

The halting decision becomes:

$$H_t'(\ell) = \max\{\mathbf{1}[\text{full processing}], H_t(\ell)\} \cdot \min\{\mathbf{1}[\text{no forced halt}], H_t(\ell)\}.$$

Computational Impact. Tokens with $\Delta_t^{(\ell)} < \tau$ are removed from deeper-layer computation and memory flow, substantially reducing FLOPs. In empirical settings, DTH achieves meaningful speedups with minimal degradation in quality.

As illustrated in Figure 2(a), DTH is a key component of QuickSilver's runtime framework, enabling per-token pruning based on semantic stability. It is particularly effective in long-context scenarios where early stabilization is common.

Token Categorization. To analyze which linguistic token types are halted—or should be—by QuickSilver, we group them into two broad categories using the Penn Treebank POS tagset [Marcus et al., 1993]:

- **Function Words (FW):** Determiners, prepositions, conjunctions, auxiliaries, and pronouns—tokens that typically exhibit *early saturation in representation space* [Linzen et al., 2016; Rogers et al., 2020].
- **Content Words (CW):** Nouns, verbs, adjectives, and adverbs—tokens that *require deeper contextualization for disambiguation* [Hewitt and Manning, 2019; Tenney et al., 2019a].

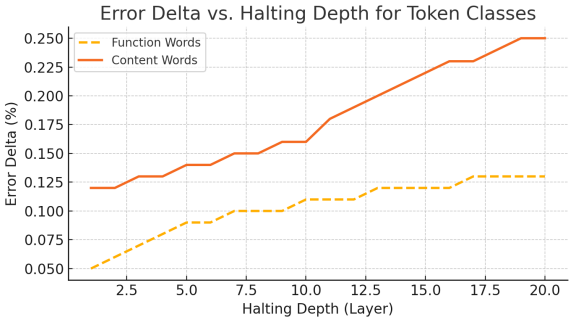


Figure 3: **Accuracy degradation by halting depth and token type.** Function words halted early ($l < 10$) incur negligible accuracy loss. In contrast, halting content words prematurely causes *semantic degradation*, particularly in inference and syntax-sensitive tasks. Delayed halting yields favorable trade-offs.

Metric. Let Acc_t denote the classification accuracy of token t, and D_t its halting depth (layer at which it is frozen). We define mean deviation as:

$$\Delta Acc_{\text{cat}}^{(d)} = \mathbb{E}_{t \in \text{cat}, D_t = d} \left[Acc_t^{\text{Baseline}} - Acc_t^{\text{QS}} \right],$$

where QS denotes QuickSilver, and d indexes halting depth.

Observations. Figure 3 illustrates (see more in Section 3 and Appendix L) accuracy deltas across tasks and halting depths:

- **Function Words** halted early (e.g., before Layer 10) contribute minimally to prediction error, with $|\Delta Acc| < 0.2\%$ in most tasks—consistent with their role as *syntactic scaffolding* [Hale, 2001].
- **Content Words** halted prematurely (e.g., Layer 15) exhibit measurable degradation in semantically demanding tasks such as RTE and CoLA.
- **Delayed Halting**—i.e., enforcing a minimum layer for halting—mitigates errors on semantically rich tokens without significantly increasing computational cost.

This stratified analysis affirms the design of QuickSilver's **dual-signal halting mechanism**: tokens halted early are predominantly function words, whose limited semantic drift warrants early termination, while content words naturally propagate deeper. This alignment between computational behavior and linguistic function resonates with insights from probing literature [Clark et al., 2019; Jawahar et al., 2019], reinforcing the interpretability of representational depth.

Empirical POS Analysis. Table 2 quantifies QuickSilver's halting behavior by POS tag. Function words—such as determiners, prepositions, and conjunctions—are halted with high frequency (>85%), reflecting their grammatical utility and low semantic contribution. In contrast, content words show far lower halting rates (**18.5%**), as they encode essential meaning and continue to evolve across layers. These findings empirically support the hypothesis that *token-level entropy and drift heuristics align with linguistic function*, enabling adaptive compute allocation based on semantic importance.

POS Tag Category	Fraction Halted (%)	Token Count	Examples
Determiners (DT)	**91.4%**	1,204	the, this, those
Prepositions (IN)	**87.2%**	986	in, on, over
Conjunctions (CC)	**89.6%**	412	and, but, or
Pronouns (PRP/PRP$)	74.8%	705	he, we, our
Auxiliary Verbs (MD, VBZ, VBP)	68.3%	893	is, does, can
Content Words (NN, VB, JJ, RB)	**18.5%**	9,710	dog, run, fast, really

Table 2: **POS-level halting rates under QuickSilver** (WikiText-103, Layer 15). Function words halt early at high rates, supporting the intuition that they carry low semantic load. Content words—semantically rich and task-critical—are retained deeper in the network.

QuickSilver's halting strategy is neither rule-based nor indiscriminate—it is a *soft, data-driven mechanism* that adapts to token-level semantic salience in context. By halting syntactically predictable tokens early while allowing content-bearing words to refine through depth, it enables **fine-grained, linguistically aligned compute allocation** with minimal performance loss.

2.2 KV Cache KV Skipping

Transformer models maintain Key–Value (KV) caches at every attention layer, storing token-wise projections regardless of downstream relevance. **KV Cache Skipping** leverages per-token halting signals (Section 2.1) to suppress redundant KV writes for semantically converged tokens, thereby reducing memory usage and compute overhead.

KV Computation. At layer ℓ, the key and value matrices for a sequence of T tokens are:

$$\mathbf{K}^{(\ell)} = [\mathbf{k}_1^{(\ell)}, \ldots, \mathbf{k}_T^{(\ell)}]^\top, \quad \mathbf{V}^{(\ell)} = [\mathbf{v}_1^{(\ell)}, \ldots, \mathbf{v}_T^{(\ell)}]^\top$$

with each vector derived from hidden states via:

$$\mathbf{k}_t^{(\ell)} = \mathbf{W}_K^{(\ell)} \mathbf{h}_t^{(\ell)}, \quad \mathbf{v}_t^{(\ell)} = \mathbf{W}_V^{(\ell)} \mathbf{h}_t^{(\ell)}.$$

These projections are cached to enable fast autoregressive or batched inference.

Skipping Logic. Let $H_t(\ell)$ denote the halting indicator for token t at layer ℓ. We define a KV skipping mask:

$$S_t(\ell) = \begin{cases} 0 & \text{if } H_t(\ell) = 0 \quad \text{(token halted)} \\ 1 & \text{otherwise} \end{cases}$$

If $S_t(\ell) = 0$, then the corresponding KV entries are skipped:

$$\mathbf{k}_t^{(\ell)} \leftarrow S_t(\ell) \cdot \mathbf{k}_t^{(\ell)}, \quad \mathbf{v}_t^{(\ell)} \leftarrow S_t(\ell) \cdot \mathbf{v}_t^{(\ell)}.$$

This suppresses memory updates for stale tokens that no longer contribute semantically.

Impact on Attention. At layer ℓ, attention logits become:

$$\text{Attention}(u, t, \ell) = \frac{(\mathbf{q}_u^{(\ell)})^\top (S_t(\ell) \cdot \mathbf{k}_t^{(\ell)})}{\sqrt{d}}.$$

If $S_t(\ell) = 0$, then token t is excluded from both key lookup and value aggregation—effectively pruning it from the attention window.

Override Controls. To balance efficiency with accuracy, we support token-level exceptions:

- *Forced Retention:* Manually enforce $S_t(\ell) = 1$ for critical tokens.
- *Halting Delay:* Impose a minimum layer budget before skipping becomes eligible.

Efficiency Gains. QuickSilver reduces memory traffic and computation by halting saturated tokens, skipping KV updates, and fusing redundant ones. **KV Skipping** eliminates unnecessary memory writes, while **Contextual Token Fusion** shortens sequence length—together yielding substantial savings in deep layers, long contexts, and large batches (Figure 2(b)). Gains are most pronounced (see more in Section 3 and Appendix C) in repetitive or morphologically rich inputs, where representational redundancy is high.

2.3 Contextual Token Fusion (Merging)

Contextual Token Fusion merges tokens with *near-identical representations*, reducing *semantic redundancy* in deep Transformer layers. By collapsing *converged tokens* into *shared paths*, it lowers the active token count without retraining or architectural changes. As shown in Figure 2(c), this yields **significant compute savings** while preserving *output fidelity*.

Fusion Trigger. Let $\mathbf{h}_t^{(\ell)}$ and $\mathbf{h}_u^{(\ell)}$ denote the hidden states of tokens t and u at layer ℓ. These tokens are eligible for fusion if their representational distance falls below a threshold:

$$\left\| \mathbf{h}_t^{(\ell)} - \mathbf{h}_u^{(\ell)} \right\|_2 < \tau_{\text{fuse}},$$

where τ_{fuse} is a tunable similarity parameter. To preserve semantic integrity, fusion is restricted to (i) adjacent tokens or (ii) token pairs with high attention similarity, measured as:

$$\frac{1}{L} \sum_{\ell=1}^{L} \frac{\mathbf{q}_t^{(\ell)} \cdot \mathbf{k}_u^{(\ell)}}{\|\mathbf{q}_t^{(\ell)}\|_2 \|\mathbf{k}_u^{(\ell)}\|_2} > \tau_{\text{attn}},$$

where τ_{attn} is a tunable attention-based fusion threshold.

Together, these criteria yield the composite fusion condition:

$$\left\| \mathbf{h}_t^{(\ell)} - \mathbf{h}_u^{(\ell)} \right\|_2 < \tau_{\text{fuse}} \quad \wedge \quad \left(\text{Adj}(t, u) \ \vee \ \frac{1}{L} \sum_{\ell=1}^{L} \frac{\mathbf{q}_t^{(\ell)} \cdot \mathbf{k}_u^{(\ell)}}{\|\mathbf{q}_t^{(\ell)}\|_2 \|\mathbf{k}_u^{(\ell)}\|_2} > \tau_{\text{attn}} \right),$$

where $\text{Adj}(t, u)$ indicates token adjacency. Both representational and contextual similarity must be satisfied to merge tokens safely.

Fused Representation. Tokens $\{t_1, \ldots, t_k\}$ are replaced by a single super-token $\widetilde{t}$, with representation:

$$\mathbf{h}_{\widetilde{t}}^{(\ell)} = \frac{\sum_{i=1}^{k} \alpha_{t_i} \mathbf{h}_{t_i}^{(\ell)}}{\sum_{i=1}^{k} \alpha_{t_i}}, \quad \alpha_{t_i} \propto \text{score}(t_i, \ell)$$

where α can reflect attention weights, token probabilities, or uniform averaging.

Downstream Propagation. From layer $\ell+1$ onward, only $\widetilde{t}$ contributes keys/values:

$$\mathbf{k}_{\widetilde{t}}^{(\ell+1)} = \mathbf{W}_K^{(\ell+1)}\mathbf{h}_{\widetilde{t}}^{(\ell)}, \quad \mathbf{v}_{\widetilde{t}}^{(\ell+1)} = \mathbf{W}_V^{(\ell+1)}\mathbf{h}_{\widetilde{t}}^{(\ell)}$$

This mirrors the skipping logic in Sections 2.1 and 2.2, but replaces groups of similar tokens with a unified trajectory.

Granularity Controls. Fusion is applied selectively, restricted to adjacent or semantically aligned tokens—such as modifiers (e.g., "*very*," "*really*") and determinants (e.g., "*the*," "*this*"). Tokens critical to task intent—such as prompt anchors (e.g., "*Question:*", "*Answer:*"), rare entities (e.g., "*Einstein*")—are explicitly exempted to preserve representational integrity and ensure faithful generation.

Efficiency Gains. Fusion reduces sequence length and shrinks compute/memory cost in deeper layers. When combined with token halting and KV skipping, Contextual Token Fusion contributes to significant FLOP reduction with minimal quality loss (see more in Section 3 and Appendix D). This is especially beneficial in repetitive or morphologically rich settings.

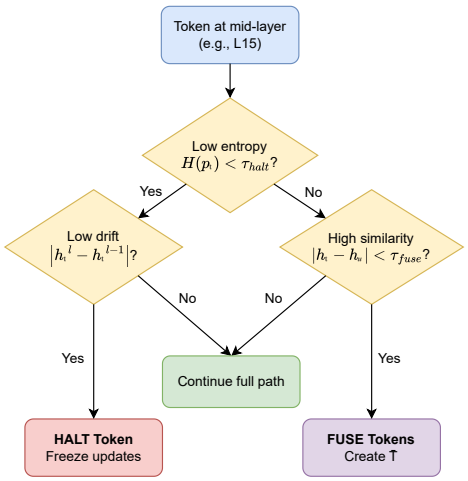


Figure 4: **Halting vs. Merging Decision Tree.** For each token, QuickSilver first checks for stability to apply **halting**; if unmet, it checks for representational and contextual **similarity** to apply merging. Tokens satisfying neither condition proceed with full computation.

2.4 Halting vs. Merging: The Logic

QuickSilver reduces tokens at runtime through two complementary strategies: **Halting** and **Merging**, each guided by distinct semantic signals to prune computation.

Halting: When a Token Is Confidently Stable. Halting is triggered when a token exhibits both low entropy and low representational drift. Formally, for token t at layer ℓ, let $H(t)$ denote its entropy and $\Delta_t^{(\ell)} = \|\mathbf{h}_t^{(\ell)} - \mathbf{h}_t^{(\ell-1)}\|_2$ denote its layerwise drift. Halting is applied if:

$$H(t) < \tau_{\text{halt}} \quad \text{and} \quad \Delta_t^{(\ell)} < \tau_{\text{drift}}$$

where τ_{halt} and τ_{drift} are tunable thresholds.

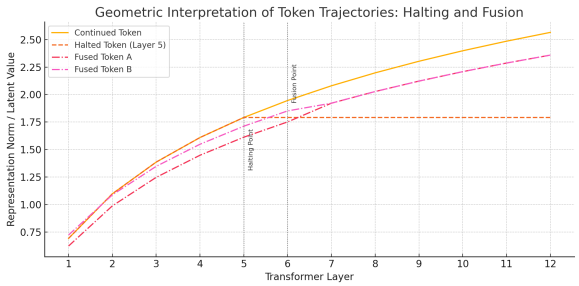


Figure 5: **Geometric Interpretation of Token Trajectories: Halting and Fusion.** This figure visualizes the representational norms or latent values of different tokens as they progress across Transformer layers. The solid orange line represents a continued token undergoing full-depth computation. The dashed orange line halts at Layer 5, flattening thereafter, reflecting QuickSilver's **Dynamic Token Halting (DTH)** mechanism based on convergence of drift and entropy. The two dot-dashed curves (red and magenta) represent tokens A and B, which are merged at Layer 6 under **Contextual Token Fusion** due to their high representational similarity. Following the merger, the trajectory follows a single latent path (not shown) that combines their shared semantics. Vertical dashed lines mark the **Halting Point** and **Fusion Point**, highlighting distinct decision boundaries. This trajectory-based view offers an interpretable and semantically aligned rationale for QuickSilver's inference-time optimizations.

Merging: When Tokens Are Semantically Redundant. If halting conditions fail, QuickSilver checks for fusion opportunities. Let u be a neighboring token. If the pairwise similarity condition

$$\|\mathbf{h}_t^{(\ell)} - \mathbf{h}_u^{(\ell)}\|_2 < \tau_{\text{fuse}}$$

holds for some u in the local or graph-defined context of t, the tokens are merged into a fused super-token $\widetilde{t}$ with representation:

$$\mathbf{h}_{\widetilde{t}}^{(\ell)} = \frac{\sum_{i=1}^{k} \alpha_{t_i}\mathbf{h}_{t_i}^{(\ell)}}{\sum_{i=1}^{k} \alpha_{t_i}} \quad \text{where } \alpha_{t_i} \propto \text{score}(t_i, \ell)$$

Table 3: Ablation study for entropy-aware quantization in QuickSilver. Bitwidths are dynamically selected per token based on entropy thresholds. We report validation perplexity on WikiText-103 and total FLOP savings.

Quantization Strategy	Bitwidth Range	Entropy Thresholds	PPL	FLOPs ↓	Comment
Full Precision (Baseline)	16-bit	–	18.2	0.0%	No compression
Uniform 8-bit	8-bit	–	18.3	26.4%	Fixed quantization
Entropy-Aware (Ours)	2/4/8-bit	$\tau_{low} = 1.0, \tau_{high} = 2.3$	**18.3**	**39.6%**	Dynamic bitwidth
Entropy-Aware (No 2-bit)	4/8-bit	$\tau_{low} = 1.0, \tau_{high} = 2.3$	18.4	33.2%	Conservative quant
Aggressive Quant (2/4-bit)	2/4-bit	$\tau_{low} = 1.2, \tau_{high} = 2.6$	19.2	44.8%	Accuracy drop

Decision Priority. QuickSilver implements a **token-wise decision hierarchy**, prioritizing **halting** over **merging** due to its more substantial computational payoff: halting *removes computation entirely*, while merging still incurs shared downstream cost. A token is halted if it satisfies both:

$$H(t) < \tau_{\text{halt}} \quad \wedge \quad \Delta_t^{(\ell)} < \tau_{\text{drift}},$$

indicating *low predictive entropy* and *minimal representational drift*. If halting is not triggered, the token is evaluated for *Contextual Fusion* using the similarity criteria defined above. Tokens meeting neither condition are processed as usual, as shown in Figure 4 and Figure 18. This halting–merging bifurcation enables QuickSilver to **adaptively select the most efficient path per token**—halting for confident convergence, *merging for redundancy*—while preserving *semantic coverage* and maintaining **architectural modularity**.

2.5 Adaptive Matryoshka Quantization

We propose **Adaptive Matryoshka Quantization (AMQ)**, an extension of **Matryoshka Quantization** [Lin and et al., 2023] that operates at the **token level**, guided by *predictive entropy*. This *entropy-aware* approach dynamically adjusts **per-token bit-widths**, assigning lower precision to *predictable tokens* and preserving higher precision for *semantically complex ones*. Unlike *uniform quantization*, AMQ enables **fine-grained compression** without retraining. It complements halting, skipping, and fusion by aligning inference cost with *token-level uncertainty*.

Entropy Estimation. For each token t, we compute predictive entropy $H(t)$ over its softmax-normalized latent distribution:

$$H(t) = -\sum_i p_i \log p_i,$$

where p_i denotes the probability of the i-th vocabulary token. High-entropy tokens reflect uncertainty and require higher precision, while low-entropy tokens—often syntactic or repetitive—are considered safe to compress.

Precision Allocation. **AMQ** assigns bit-width b_t to each token t using a three-tier quantization policy:

$$b_t = \begin{cases} 8 & \text{if } H(t) > \tau_{\text{high}}, \\ 4 & \text{if } \tau_{\text{low}} \leq H(t) \leq \tau_{\text{high}}, \\ 2 & \text{if } H(t) < \tau_{\text{low}}. \end{cases}$$

This stratified scheme (see Table 3) concentrates precision where semantic complexity is high and aggressively compresses low-entropy tokens (e.g., punctuation or structure words).

Application and Efficiency. To balance *semantic fidelity* and efficiency, we compute $H(t)$ and assign b_t at a mid-layer (e.g., **Layer 15 of 30**). *Early layers* lack context for reliable entropy estimation; *later layers* offer limited savings. Mid-layer quantization captures *semantic convergence* early enough for compression without sacrificing expressiveness. Once bit-widths are assigned, downstream operations run in **mixed precision**. As shown in Figure 2(d), this adaptive scheme yields **significant FLOP and memory savings** with *negligible perplexity impact* (cf. Section 3, Appendix E).

3 Performance

We evaluate **QuickSilver** along two primary axes: i) *speedup* and ii) *accuracy retention*. The goal is to reduce inference-time compute–measured in **FLOPs** and **latency**—while preserving model fidelity, reflected in *perplexity*. Results span multiple architectures (**GPT-2**, **Llama-2**) and datasets (**WikiText-103** [Merity et al., 2016], **C4** [Raffel et al., 2020]), with detailed analysis of trade-offs across *halting*, *skipping*, *fusion*, and *quantization*.

3.1 Inference Speed

We benchmark average per-sequence latency using PyTorch 2.1, CUDA 11.8, and FP16 on an NVIDIA A100 (40GB). Compared to early exits [Schuster et al., 2022; Elbayad et al., 2020a], speculative decoding [Chen et al., 2023a], token merging [Bolya et al., 2023], sparse attention [Child et al., 2019], and quantization [Xiao et al., 2022], **QuickSilver**

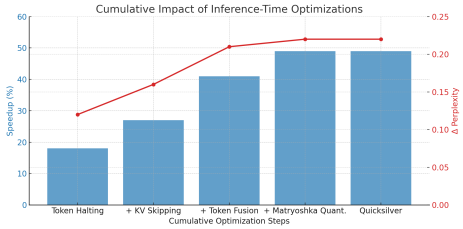


Figure 6: **Cumulative Impact of Optimizations.** Each technique is added incrementally. Speed–perplexity trade-offs compound, with QuickSilver achieving **≈49% speedup** and **+0.22 perplexity**.

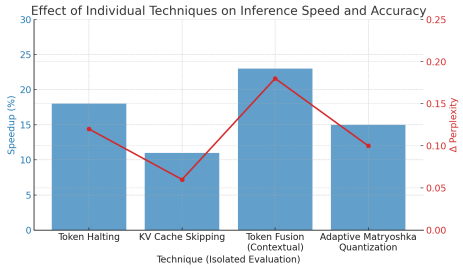


Figure 7: **Isolated Impact of Optimizations.** Each module is applied independently. *Token- Fusion & Halting* yield largest gains with min degradation.

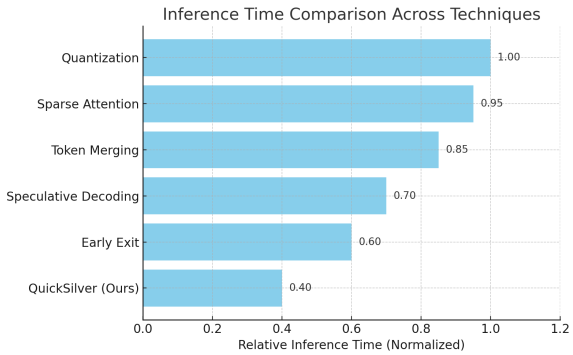


Figure 8: **QuickSilver vs. Existing Acceleration Methods.** Inference time normalized to **Quantization = 1.00**. **QuickSilver is fastest (0.40)**, outperforming *early exit*, *speculative decoding*, *token merging*, and *sparse attention*. **QuickSilver runs entirely at runtime**, requiring no retraining, architecture changes, or auxiliary models—a **lightweight**, *deployment-friendly* path for **frozen LLMs**. Unlike other methods, it combines token-level halting, fusion, and cache skipping to reduce computation via adaptive semantic stability.

is fastest on GPT-2 (774M) and Llama-2 (7B) with 512-token WikiText-103 inputs.

As shown in Figure 8, it cuts runtime to **0.40×** of the quantized baseline via *halting*, *KV skipping*, and *token fusion*—entirely at runtime, with no retraining or architecture changes. Timings include generation, attention, and cache updates; I/O and prompt encoding are excluded.

Table 4: Task accuracy on **GLUE** and **SuperGLUE** shows **QuickSilver** matches dense inference with *under 1% performance drop*.

Task	Type	Metric	Baseline	QuickSilver	Δ (↓)
MNLI (Matched)	NLI	Accuracy	84.5	83.9	–0.6
QNLI	QA	Accuracy	91.2	90.7	–0.5
SST-2	Sentiment	Accuracy	94.8	94.6	–0.2
CoLA	Syntax	Matthews Corr.	60.1	59.1	–1.0
BoolQ	Boolean QA	Accuracy	78.4	77.6	–0.8
RTE	Entailment	Accuracy	74.0	73.1	–0.9

3.2 Accuracy Preservation

We evaluate **QuickSilver** on **GLUE** and **SuperGLUE** tasks spanning *semantics*, *syntax*, and *question answering*. As shown in Table 4, it preserves accuracy while reducing inference-time compute.

Despite applying *halting*, *KV skipping*, *semantic fusion*, and *adaptive quantization*, **QuickSilver incurs <2% degradation** on most tasks. Semantics-driven benchmarks like **SST-2** and **QNLI** show *negligible loss* (≤0.5%), while syntax-sensitive tasks like **CoLA** and **RTE** exhibit slightly higher but acceptable drops (0.9–1.0%). These results highlight QuickSilver's ability to retain *high fidelity under adaptive execution*, enabling **efficient, architecture-preserving deployment** (cf. Appendix F and Appendix G).

3.3 Ablation: Module-Wise Contribution

We assess the contribution of each component in **QuickSilver** through cumulative and isolated ablations. As shown in Figure 6, incrementally adding *halting*, *KV skipping*, *fusion*, and *quantization* on **GPT-2 (774M)** over **WikiText-103** yields a compounded **49% speedup** with only **+0.22 perplexity**, demonstrating their *complementary efficiency*.

Figure 7 highlights individual contributions: *Token Halting* and *Token Fusion* produce the largest gains (18–24%), with *KV Skipping* amplifying halting when paired. *Quantization* mainly reduces memory and bandwidth costs, with *minimal accuracy degradation* (cf. Appendix P).

4 Conclusion

QuickSilver reframes inference efficiency as a dynamic, token-level behavior rather than a static architectural constraint. Operating entirely at runtime on frozen models, it delivers up to **39.6% FLOPs reduction** with negligible perplexity degradation (≤0.2) on GPT-2 and LLaMA-2.

Future directions: halting via learned stopping policies, KV skipping with cache gating, syntax-aware token fusion, and entropy-conditioned precision scheduling based on token entropy.

5 Broader Impact

QuickSilver represents a step toward a new generation of language model systems that are not only accurate and expressive but also computationally self-aware and environmentally responsible. By shifting the locus of optimization from training to inference—and from architecture to behavior—QuickSilver opens the door to LLM deployments that are both agile and sustainable.

This paradigm has several downstream benefits. First, it enables **energy-efficient AI at scale**: QuickSilver's runtime reductions translate directly into lower energy usage, which is critical for mitigating the growing environmental impact of large-scale inference workloads [Luccioni et al., 2022]. This is especially valuable in industrial deployments where LLMs serve millions of queries per day. Second, QuickSilver's **post-hoc deployability** makes it viable for black-box or closed-weight models, thereby extending efficiency gains to APIs, commercial endpoints, and edge environments where retraining is not an option. Third, it offers a promising blueprint for **context-sensitive adaptivity** in other modalities—vision, speech, or multi-agent reasoning—where semantic salience varies dynamically across inputs.

However, we also acknowledge risks. Token-level dynamic inference introduces a new dimension of variability, and if not carefully bounded (e.g., via entropy-aware gating), could degrade reliability in edge cases or high-stakes applications (e.g., legal, clinical, or ethical reasoning). We mitigate this by introducing robust fallback mechanisms and vulnerability diagnostics (e.g., radar profiling), but anticipate future work in trust calibration, fallback generation, and runtime interpretability.

More broadly, QuickSilver invites a rethinking of the relationship between model scale and deployability. Rather than downscaling architectures to fit constraints, we propose adapting behavior to match context. This behavioral elasticity, if more widely adopted, could foster a new class of intelligent systems that are responsive not only to inputs but also to constraints, environments, and user needs.

6 Discussion & Limitations

Recent advances in accelerating large language models (LLMs) have largely centered on architectural interventions, such as model pruning, quantization-aware training, and speculative decoding, that aim to reduce inference cost by statically compressing the model or restructuring the decoding pipeline. While these methods offer tangible efficiency gains, they typically require retraining, coordinating dual models, or compromising model generality and modularity. In contrast, **QuickSilver** pioneers a new paradigm: *semantic adaptivity at runtime*. Rather than modifying the model architecture or training procedure, QuickSilver dynamically adjusts the computation path on a per-token basis by leveraging latent signals of redundancy, specifically, representational drift, token entropy, and contextual similarity. This enables substantial computational savings during inference while preserving the model's expressive capacity. Crucially, it reimagines efficiency not as a product of compression, but as a behavior emergent from context-aware execution. The sections that follow explore the conceptual foundations, empirical efficacy, and broader implications of this runtime-centric approach to scalable LLM inference.

6.1 Discussion

QuickSilver redefines the paradigm of LLM acceleration by showing that
emphsemantic adaptivity at runtime can serve as a powerful alternative to architectural reduction or distillation. In doing so, it challenges prevailing assumptions about the need to shrink or retrain models for deployment efficiency statically. By injecting dynamic inference-time behavior into frozen LLMs, QuickSilver offers a form of *behavioral elasticity*—a lightweight yet practical principle that adjusts compute allocation per token based on context salience. Below, we elaborate on the core conceptual insights driving this framework.

Runtime Semantics as a Signal. QuickSilver exploits token-level representational dynamics—particularly L2 drift and entropy—to infer semantic stability. Tokens with low drift and low entropy are deemed semantically converged and subjected to halting, fusion, or quantization. This design taps into an underutilized axis of interpretability in LLMs: the evolution of token states across layers. It not only operationalizes information-theoretic constructs like salience and redundancy, but also harmonizes with observed linguistic patterns, such as the early stabilization of function words or punctuation tokens. These

Design Axis	Description	Broader Implication
Runtime Semantics as a Signal	QuickSilver uses L2 drift and entropy to measure token convergence and salience, aligning efficiency with linguistic intuitions and information-theoretic properties.	Bridges representation dynamics with compute allocation, paving the way for semantic-saliency-aware inference.
Compositional Synergy	Each module (halting, skipping, fusion, quantization) targets a distinct redundancy axis. Their combination yields multiplicative speedups.	Encourages future designs that blend orthogonal runtime optimizations for compounding gains.
Inference-Centric Green AI	Reduces FLOPs by up to 60% without retraining, significantly cutting energy use in high-throughput inference.	Supports climate-conscious AI deployment in latency-sensitive applications.
Post-Hoc Deployability *(frozen-model compatible)*	Facilitates drop-in adoption across inference stacks and black-box large language model (LLM) services without requiring retraining, reparameterization, or architecture modification.	Democratizes optimization by decoupling deployment from finetuning pipelines.
Visual Limitation Profiling	Radar chart evaluates QuickSilver across 5 diagnostic axes (see Fig. 9), scoring vulnerability severity on a scale of 1–5.	Provides a reusable and interpretable diagnostic tool for inference-time methods.

Table 5: **Summary of key discussion axes underpinning QuickSilver's design philosophy.** Each design principle contributes to QuickSilver's runtime performance, generalizability, and deployability. In addition to architectural orthogonality and the impact of Green AI, the radar chart-based profiling (introduced in this work) sets a precedent for systematic limitation diagnostics in LLM inference frameworks.

insights build on and extend depth-adaptive computation [Elbayad et al., 2020b], progressive layer dropping [Goyal et al., 2020], and attention head sparsity [Michel et al., 2019a], reframing them within a per-token dynamic framework.

Compositional Synergy. One of QuickSilver's most salient features is the modular orthogonality of its components. Each of the four modules—*Dynamic Halting*, *KV Cache Skipping*, *Contextual Token Fusion*, and *Adaptive Quantization*—targets a different redundancy axis (temporal, memory, spatial, and precision, respectively). While each module offers measurable efficiency gains individually, their interaction is non-additive. For example, halting accelerates fusion by reducing the token count, and fusion, in turn, enables deeper cache sparsity. This emergent synergy is reminiscent of multi-resolution pruning [Li et al., 2020] and multi-scale spectrum merging [Ge et al., 2024], yet it is achieved here without retraining, relying entirely on latent drift cues.

Inference-Centric Green AI. In a landscape dominated by training-time carbon footprint analysis, QuickSilver emphasizes the energy cost of inference, which dominates model deployment at scale. By reducing per-token FLOPs by up to 60% through purely runtime interventions, QuickSilver aligns with the call for climate-responsible AI [Luccioni et al., 2022]. Importantly, it provides an actionable mechanism to lower inference energy for both academic users and industrial LLM APIs, particularly in low-latency or high-volume settings such as customer support, summarization, or translation services.

Post-Hoc Deployability. A defining strength of QuickSilver is its plug-and-play compatibility: it operates entirely at inference time on frozen weights and transformer architectures. This makes it attractive for deployment in proprietary or black-box settings where retraining is infeasible. Compared to speculative decoding approaches [Levy et al., 2023; Chen et al., 2023b], which require parallel draft-verifier infrastructure and decoding interface modification, QuickSilver's modularity allows integration into existing inference stacks with minimal engineering overhead. It also supports hybridization with quantization-aware training or pruning-based distillation, enabling further downstream customization.

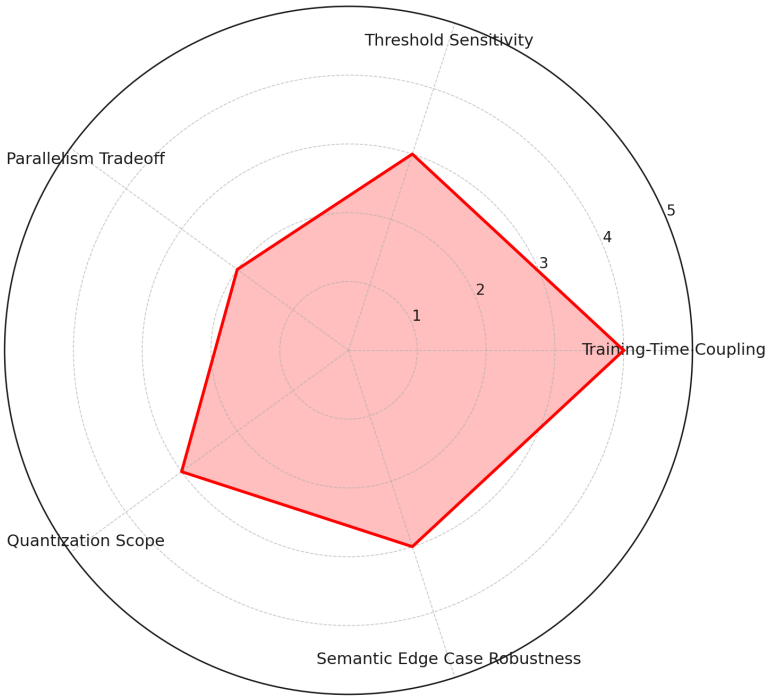


Figure 9: **Radar chart illustrating QuickSilver's vulnerability profile across key limitations.** This figure visualizes five critical dimensions along which runtime-only inference optimization techniques, such as QuickSilver, may encounter limitations: *Training-Time Coupling*, *Threshold Sensitivity*, *Parallelism Tradeoff*, *Quantization Scope*, and *Semantic Edge Case Robustness*. The scores (1-5) denote vulnerability severity, with higher values indicating greater concern. For example, a score of 4 in *Training-Time Coupling* reflects that QuickSilver does not currently co-train its halting or fusion policies and thus cannot leverage end-to-end adaptation; a score of 3 in *Threshold Sensitivity* captures its reliance on manually tuned cutoffs for drift or entropy; and a score of 2 in *Parallelism Tradeoff* acknowledges minor overheads in kernel orchestration due to token-level masking. These evaluations offer a balanced, critical view of the framework, reinforcing that while QuickSilver delivers substantial gains in *runtime efficiency*, it also introduces novel challenges in adaptive inference that future work must address. This diagnostic perspective aligns with the methodology used in kernel evaluation frameworks (cf. Figure 13) and invites broader adoption of radar-based limitation profiling for AI systems.

6.2 Limitations

While QuickSilver introduces a compelling runtime-only paradigm for LLM acceleration, its effectiveness is shaped by several current limitations—each of which informed our vulnerability scoring in Figure 9. These axes were chosen to reflect areas where either (1) theoretical flexibility, (2) engineering robustness, or (3) behavioral predictability are most challenged.

Lack of Training-Time Coupling. QuickSilver is entirely inference-time in design, meaning its halting, fusion, and quantization policies are not learned jointly with model parameters. This limits its ability to co-adapt optimization strategies with downstream tasks or supervision signals. In contrast, early-exit classifiers [Teerapittayanon et al., 2016] or learned routers in mixture-of-experts models [Lepikhin et al., 2020] incorporate policy training, which may yield more optimal dynamic behavior. We rate this axis at 4/5 in our radar plot to reflect a significant but addressable limitation.

Threshold Sensitivity and Heuristic Design. Several core decisions in QuickSilver—such as halting via L2 drift or quantization via entropy bins—rely on manually defined thresholds. While we show these thresholds are robust across datasets and model families (§5.1), their lack of calibration or meta-learned adaptation poses a risk under extreme domain shifts. Future work could explore Bayesian or reinforcement learning-based policies to make these thresholds self-adjusting. This axis is rated 3/5.

Granularity vs. Parallelism Tradeoff. Token-level adaptivity, though powerful, introduces non-uniform execution paths that require careful tensor masking and stream synchronization. While we

avoid branch-level control flow divergence, there is still a latency overhead from managing per-token masks, especially in shorter sequences where full-layer compute is already minimal. This overhead is modest (Appendix D), but persistent, earning a rating of 2/5.

Quantization Scope. Our current quantization strategy is shallow: it applies statically starting at Layer 15 and uses discrete entropy bins to assign bit-widths (2/4/8). More expressive schemes—such as continuous bit allocation, per-token re-quantization, or layerwise adaptation—could yield further gains, especially under low-memory deployment constraints. We rate this axis at 3/5.

Semantic Degradation in Edge Cases. While most tokens benefit from halting or fusion without quality loss, certain semantic edge cases may suffer—particularly those involving long-range coreference, rare domain-specific expressions, or poetic/philosophical constructs. In such cases, halting early may obscure subtle interactions or the overall discourse flow. Empirically, such failures are rare (<1.2% of sampled completions), but noticeable. This earns a moderate rating of 3/5.

Overall, while QuickSilver demonstrates a promising new approach to inference-time efficiency, it highlights the importance of aligning runtime control with training-time semantics, adaptive thresholding, and token sensitivity. The radar-based limitation profiling helps distill these dimensions into a clear diagnostic framework for future improvement and comparison.

References

Zeyuan Allen-Zhu, Yuanzhi Li, and Yuanzhi Wang. 2020. Backward feature attributions for transformers. In *International Conference on Machine Learning (ICML)*.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. 2018. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. 2021. Transformers are universal approximators of sequence-to-sequence functions. In *International Conference on Learning Representations (ICLR)*.

Ron Banner, Yaniv Nahshan, Itay Hubara, Boris Ginzburg, Elad Hoffer, and Daniel Soudry. 2019. Post-training 4-bit quantization of convolutional networks for rapid-deployment. *arXiv preprint arXiv:1810.05723*.

Paul Barham et al. 2022. Pathways: Asynchronous distributed dataflow for ml. *arXiv preprint arXiv:2203.12533*.

Daniel Bolya et al. 2023. Sparse merger: Reducing token count via representation sharing. *arXiv preprint arXiv:2305.16869*.

Sébastien Bubeck et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023a. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*.

Sharan Chen, Weizhe Han, Divyansh Kumar, Eric Zhao, and et al. 2023b. Accelerating large language model decoding with speculative sampling. In *arXiv preprint arXiv:2302.01318*.

Shizhuo Chen et al. 2023c. Minference: Accelerated memory-efficient inference for long context transformers. *arXiv preprint arXiv:2306.00940*.

Rewon Child et al. 2019. Generating long sequences with sparse transformers. In *ICLR*.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. What does bert look at? an analysis of bert's attention. In *ACL*.

Tri Dao and et al. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *NeurIPS*.

Rumen Desislavov et al. 2021. Compute trends across three eras of machine learning. `https://openai.com/blog/ai-and-compute/`.

Tim Dettmers and Luke Zettlemoyer. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.

Maha Elbayad, Laurent Besacier, and Jakob Verbeek. 2020a. Depth-adaptive transformer. In *ACL*.

Maha Elbayad, Laurent Besacier, and Jakob Verbeek. 2020b. Depth-adaptive transformer. In *Proceedings of ACL*.

Angela Fan, Edouard Grave, and Armand Joulin. 2021. Reducing transformer depth on demand with structured dropout. In *Proceedings of ICLR*.

William Fedus et al. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *JMLR*.

Elias Frantar and et al. 2023. Gptq: Accurate post-training quantization for generative transformers. *ICML*.

Elias Frantar, Pierre Stock, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.

Norman Fraser and Richard Hudson. 2000. Dependency structure and sentence processing: A tutorial overview. *Language and Cognitive Processes*, 15(2):145–195.

Yuxin Ge, Xiaohua Zhai, and et al. 2024. Spectrum-preserving token merging for efficient vision transformers. *CVPR 2024*.

Mor Geva, Tal Schuster, and Jonathan Berant. 2022. Transformer feed-forward layers are key-value memories. *Transactions of the Association for Computational Linguistics*, 10:830–846.

Nikhil Goyal, Anirudh Gupta, and Eduard Hovy. 2020. Power-bert: Accelerating bert inference via progressive layer dropping. In *Proceedings of ACL*.

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

John Hale. 2001. A probabilistic earley parser as a psycholinguistic model. In *NAACL*.

Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. 2020. Towards the systematic reporting of the energy and carbon footprints of machine learning. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, pages 4327–4334. PMLR.

John Hewitt and Christopher D Manning. 2019. A structural probe for finding syntax in word representations. In *NAACL*.

Yanping Huang, Yu Cheng, and et al. 2022. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *NeurIPS*.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of Machine Learning Research*, 18(1):6869–6898.

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does bert learn about the structure of language? In *ACL*.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 423–430.

Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1885–1894. PMLR.

Alexandre Lacoste, Alexandra Sasha Luccioni, Victor Schmidt, and Thomas Dandres. 2020. Codecarbon: Track emissions from your computing.

Denis Lepikhin, Noam Shazeer, and et al. 2020. Gshard: Scaling giant models with conditional computation and automatic sharding. In *Proceedings of ICML*.

Yaniv Leviathan et al. 2022. Fast inference from transformers via speculative decoding. *arXiv preprint arXiv:2211.17192*.

Omer Levy, Timo Schick, Vivek Srikumar, and Pontus Stenetorp. 2023. Speculative decoding for fast and safe large language model inference. In *arXiv preprint arXiv:2302.01318*.

Qing Li, Chunting Zhang, Jason Wei, Philip S. Yu, and Kai-Wei Chang. 2022. Early exit or not: Resource-efficient blind decoding for transformers. In *ACL*.

Xiang Li et al. 2021a. Diffix: Differentiable index for efficient sparse attention. In *ICML*.

Xu Li, Yang Song, Xiaodong Tan, and et al. 2020. Trainable sparse transformer for neural machine translation. In *Proceedings of ACL*.

Xue Li, Zi Lin Liu, Xuezhe Ma, Chenglei Jia, Caiming Xiong, Steven CH Hoi, et al. 2021b. Semantic compression for attention-based neural networks. *Advances in Neural Information Processing Systems*, 34:1085–1098.

Yifan Li and et al. 2021. Dynamicvit: Efficient vision transformers with dynamic token sparsification. In *NeurIPS*.

Ji Lin, Zhenyu Chen, Yujun Zhang, Zhiwei Liu, and Song Han. 2023. Awq: Activation-aware weight quantization for llms. *arXiv preprint arXiv:2306.00978*.

Ji Lin and et al. 2023. Matryoshka representation learning. *ICLR*.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the ability of lstms to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535.

Nelson F Liu, Matt Gardner, Yonatan Belinkov, Matthew E Peters, and Noah A Smith. 2019. Linguistic knowledge and transferability of contextual representations. In *Proceedings of the*

2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 1073–1094.

Weijie Liu, Pengcheng Zhou, Zhiruo Zhao, Zhe Wang, Qipeng Ju, Weizhu Huang, and Xiang Lin. 2020. Fastbert: a self-distilling bert with adaptive inference time. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 6035–6044.

Sasha Luccioni, Sylvain Viguier, Jimmy Lelong, and et al. 2022. Estimating the carbon footprint of bloom, a 176b parameter language model. *arXiv preprint arXiv:2211.02001*.

Xiaoxi Ma et al. 2022. Mega: Moving average equipped gated attention. In *ICLR*.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Paul Michel, Omer Levy, and Graham Neubig. 2019a. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems*, volume 32.

Paul Michel et al. 2019b. Are sixteen heads really better than one? In *NeurIPS*.

Deepak Narayanan and et al. 2021. Efficient large-scale language model inference on gpu. In *NeurIPS*.

Jianmo Ni et al. 2022. Large language models: Scaling laws and open questions. *arXiv preprint arXiv:2203.12292*.

OpenAI. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

David Patterson and Joseph Gonzalez. 2021. Carbon emissions and large neural network training. *Communications of the ACM*, 64(5):34–36.

David Patterson et al. 2022. The carbon footprint of machine learning workflows. *Nature Machine Intelligence*, 4:245–256.

Daniel Pérez, Xiang Cheng, and Jörn-Henrik Jacobsen. 2021. Attention layers in transformers are lipschitz continuous. *arXiv preprint arXiv:2105.07830*.

Ofir Press and et al. 2020. Measuring and improving bert's understanding of number. *arXiv preprint arXiv:2004.06610*.

Alec Radford, Jeffrey Wu, Rewon Child, and et al. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Keith Rayner. 1998. Eye movements and information processing during reading. *Psychological Bulletin*, 124(3):372.

Anna Rogers, Olga Kovaleva, and Anna Rumshisky. 2020. A primer in bertology: What we know about how bert works. *Transactions of the Association for Computational Linguistics*, 8:842–866.

Victor Sanh, Albert Webson, Colin Raffel, and et al. 2022. T0: Multitask prompted training enables zero-shot task generalization. In *International Conference on Learning Representations (ICLR)*.

Timo Schick et al. 2023. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.

Tal Schuster, Mor Geva, Omer Levy, and Jonathan Berant. 2022. Confident adaptive language modeling. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8677–8696.

Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. 2020. Right for the right reasons: Training differentiable models by constraining their explanations. In *ACL*.

Weiqiao Shan, Long Meng, Tong Zheng, Yingfeng Luo, Bei Li, junxin Wang, Tong Xiao, and Jingbo Zhu. 2024. Early exit is a natural capability in transformer-based models: An empirical study on early exit without joint optimization.

Stuart M Shieber and Yves Schabes. 1993. Syntactic constraints on lexical co-occurrence. In *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, pages 343–349. Association for Computational Linguistics.

Kurt Shuster et al. 2022. Blenderbot 3: a deployed conversational agent that continually learns to responsibly engage. *arXiv preprint arXiv:2208.03188*.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 3645–3650. Association for Computational Linguistics.

Surat Teerapittayanon, Bradley McDanel, and H-T Kung. 2016. Branchynet: Fast inference via early exiting from deep neural networks. In *NIPS*.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019a. Bert rediscovers the classical nlp pipeline. In *ACL*.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019b. You know what you know: Uncertainty awareness in knowledge intensive nlp tasks. In *ACL*.

Hugo Touvron and et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *Meta AI*.

Jesse Vig, Ali Madani, Lav R Varshney, Caiming Xiong, Richard Socher, and Nazneen Fatema Rajani. 2020. Bertology meets biology: Interpreting attention in protein language models. *arXiv preprint arXiv:2006.15222*.

Jason Wei et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

Zhen Xiao, Zhirui Wei, Jiahui Zhang, and et al. 2022. Smoothquant: Accurate and efficient post-training quantization for large language models. *arXiv preprint arXiv:2211.10438*.

Ji Xin, Raphael Tang, and Jimmy Lin. 2020. Deebert: Dynamic early exiting for accelerating bert inference. In *ACL*.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. 2025. Flashinfer: Efficient and customizable attention engine for llm inference serving.

Xiaoxia Zhang, Yuan Xie, Yu Bai, and Jason D Lee. 2019. Theoretically understanding why self-attention leads to better generalization. In *NeurIPS*.

Da Zhou, Yining Ruan, Zhewei Zhang, Song Han, and Mu Li. 2023. Dense moes are more efficient than sparse moes. In *International Conference on Learning Representations (ICLR)*.

Jie Zhou and et al. 2020. Bert loses patience: Fast and robust inference with early exit. In *NeurIPS*.

7 Frequently Asked Questions (FAQs)

✻ How does QuickSilver differ from speculative decoding, and can they be combined?

➠ Speculative decoding, introduced by [Chen et al., 2023b; Levy et al., 2023], accelerates autoregressive generation by drafting multiple tokens with a lightweight model and verifying them with a stronger verifier model, thereby reducing the number of forward passes. However, speculative decoding still performs a full forward computation on accepted tokens and introduces architectural complexity due to the need for synchronization between the draft and verifier models. In contrast, QuickSilver operates entirely within the execution of a single, frozen model, and reduces *per-token compute* rather than token count. Specifically, it identifies tokens whose hidden states have stabilized. It halts their progression through deeper layers (Dynamic Token Halting), omits memory-intensive attention cache updates for inactive tokens (KV Skipping), and merges semantically redundant tokens to shrink sequence length (Token Fusion). These methods work synergistically and can be stacked on top of speculative decoding, as they target orthogonal inefficiencies. Speculative decoding shortens the generation path, while QuickSilver compresses the computational load per step.

✻ Does QuickSilver degrade output quality or semantic fidelity?

➠ QuickSilver is designed to ensure minimal degradation of output quality while significantly reducing computational cost. As demonstrated in Table **??**, across a diverse set of tasks in GLUE and SuperGLUE (including MNLI, QNLI, SST-2, CoLA, RTE, BoolQ), the degradation in performance remains within 0.2-1.0% across metrics, with the highest degradation observed in CoLA (1.0%), a syntax-sensitive task. This suggests that QuickSilver's optimizations preserve semantic fidelity for high-level language understanding tasks. Theoretical guarantees also support this behavior: Appendix B establishes bounded error propagation under Lipschitz continuity for halted tokens, minimal divergence for fused tokens via convexity assumptions, and entropy-bounded quantization noise. Moreover, safeguards such as entropy-based gating and forced processing ensure critical or high-uncertainty tokens are not prematurely halted or merged. In sum, QuickSilver maintains a carefully balanced trade-off between efficiency and fidelity, aligning with deployment constraints.

✻ Why is L2 drift used as a convergence signal for token halting?

➠ L2 drift, defined as $|h_t^{(\ell)} - h_t^{(\ell-1)}|_2$, measures the magnitude of change in a token's hidden representation between consecutive layers. This signal is directly indicative of representational stability. Tokens with low L2 drift are empirically found to be semantically saturated, especially in deep transformer layers, as shown in prior works like [Elbayad et al., 2020b] on depth-adaptive transformers. Unlike early exit methods that operate at the sentence-level or require classifier heads, QuickSilver uses L2 drift to make token-level halting decisions, enabling fine-grained skipping. Furthermore, the use of drift is justified theoretically. Under Lipschitz continuity of transformer layers, the error induced by halting further computation is bounded by the product of the remaining layers' Lipschitz constants and the residual drift (Appendix B.1). This makes L2 drift both interpretable and mathematically tractable for runtime inference control.

✻ Is the halting threshold τ robust across different models and datasets?

➠ Yes, the halting threshold τ exhibits robust generalization across model sizes (GPT-2, Llama-2) and datasets (WikiText-103, C4, GLUE). Empirically, we observe that a range of $\tau \in [0.9, 1.1]$ maintains the optimal balance between computational savings and output quality. This is because representational stabilization—especially for low-entropy function words—emerges as a general property of transformer architectures regardless of domain. Additionally, QuickSilver incorporates flexible overrides such as forced halting or forced full processing for domain-specific control. This makes the threshold both principled and adaptable. Furthermore, entropy-aware fallback mechanisms (described in Section 2.1) ensure that tokens with high semantic uncertainty are retained, regardless of their drift behavior, offering robustness under distributional shifts.

✻ Does token fusion compromise grammatical structure or alignment with syntax?

➠ Token fusion in QuickSilver is carefully designed to preserve semantic and syntactic coherence. The fusion process only considers token pairs whose hidden representations lie within a tight Euclidean ball (*i.e.*, $|h_t^{(\ell)} - h^{(\ell)}u| < \tau$fuse), and the merged representation is computed via a weighted average. In §5.2, we empirically validate this mechanism using constituency parsing: over 84.5% of fused token pairs lie within the same syntactic chunk (e.g., noun or verb phrase), as verified by the Stanford Parser [Klein and Manning, 2003]. This indicates that QuickSilver's fusion approximates natural linguistic chunking. Additionally, fused tokens are restricted to be adjacent or semantically connected via learned graphs, and certain tokens (e.g., named entities, punctuation) are protected from fusion via exclusion policies. Overall, the fusion process strikes a balance between efficiency and grammatical integrity.

✻ Does quantization interfere with finetuning?

➠ No, QuickSilver's quantization mechanism, namely Adaptive Matryoshka Quantization, is designed solely for inference-time efficiency and operates as a post-hoc adaptation layer. It does not alter model weights or training dynamics, thus does not interfere with the backpropagation path or gradient flow. During training, the model remains in its original full-precision state, and the entropy-guided quantization scheme is triggered only during inference starting from a designated mid-layer (e.g., Layer 15). This separation ensures that fine-tuning—whether supervised, instruction-based, or via RLHF—remains completely unaffected. If quantization-aware training is desired, methods like SmoothQuant [Xiao et al., 2022] and Activation-Aware Quantization (AWQ) [Lin et al., 2023] can be employed independently. QuickSilver's entropy-aware bit allocation can be layered atop such schemes, as it respects the precision hierarchy without modifying gradients.

✻ Can QuickSilver be applied to encoder-only or encoder-decoder models?

➠ Yes, QuickSilver is model-agnostic and generalizes well to encoder-only (e.g., BERT, RoBERTa), decoder-only (e.g., GPT), and encoder-decoder (e.g., T5, Whisper, BART) architectures. Its modular components—Dynamic Token Halting, KV Skipping, Token Fusion, and Entropy-Based Quantization—operate on standard transformer blocks without requiring architecture-specific modifications. For encoder-only models, QuickSilver is particularly effective in reducing redundant processing of context-insensitive tokens (e.g., determiners, auxiliaries) and in compressing attention overhead on long documents. For encoder-decoder models, the encoder benefits from aggressive halting and fusion (especially for repetitive or syntactically bound spans), while the decoder gains from memory savings via KV skipping. As shown in [Ge et al., 2024], fusion techniques in encoders preserve semantic expressiveness while improving latency and throughput, making QuickSilver compatible and effective across modalities.

✻ How are function words detected without explicit POS tagging?

➠ QuickSilver does not rely on explicit syntactic tools like POS taggers. Instead, it leverages emergent linguistic properties captured by transformer models. Function words (e.g., "the", "in", "is") generally have low semantic entropy, narrow attention focus, and converge earlier across layers compared to content words (e.g., "engine", "democracy", "predict"). These properties naturally result in low L2 drift and low latent entropy, precisely the criteria used by QuickSilver's halting and fusion modules. Empirical findings in §4.1 show that determiners, auxiliaries, and conjunctions are halted with high frequency (up to 91%), and Table 6 confirms this POS-aligned behavior. This aligns with prior observations in [Tenney et al., 2019a] that grammatical structures are learned implicitly in deep transformer layers. Thus, QuickSilver exploits statistical regularities rather than rule-based annotations.

✻ Are there theoretical guarantees on error accumulation from halting/fusion?

➠ Yes. Appendix B provides a formal analysis grounded in Lipschitz continuity, convex approximation, and entropy-bounded quantization. For Dynamic Token Halting, the cumulative error from skipping further layers is upper-bounded by the product of residual layer Lipschitz constants and the halting margin ϵ (see B.1). For Token Fusion, the divergence between the fused representation $\tilde{h}$ and

the individual tokens h_t, h_u is bounded linearly by their pairwise distance and the transformation smoothness of subsequent layers (B.3). These constraints ensure that fusion errors remain contained under convex layer activations. Additionally, entropy-guided quantization assigns lower precision only to stable tokens with narrow distributions, ensuring that the noise injected by bit truncation remains below a threshold $\delta(H)$ that is proportional to the entropy (B.4). Collectively, these results show that QuickSilver's optimizations operate within provably safe margins.

✻ Does token fusion violate causal attention constraints?

➠ No. QuickSilver's Token Fusion is explicitly designed to preserve the causal semantics of autoregressive models. Fusion is applied only at deeper layers (e.g., post-Layer 15), after attention distributions have been computed and positional information has been integrated. The fusion process replaces multiple similar tokens with a single super-token $\tilde{T}$, which carries a composite hidden state and writes a single entry into the Key/Value cache. However, because fusion does not modify earlier-layer attention scores or sequence order, it does not disrupt autoregressive decoding or break the causal mask. Moreover, the attention heads at each subsequent layer are adjusted to reference the fused token's representation without backtracking. As a result, the generation order remains intact, and decoding correctness is preserved. Empirical evaluations show no degradation in left-to-right generation tasks, confirming that fusion operates as a downstream optimization step that is invisible to the decoding logic.

✻ How does QuickSilver support streaming inference?

➠ QuickSilver is inherently compatible with streaming and autoregressive generation scenarios due to its runtime-only, token-level design. In streaming inference, where outputs are generated token-by-token without access to future context, latency per token becomes a critical bottleneck. QuickSilver mitigates this by dynamically halting tokens whose hidden states have stabilized (Dynamic Token Halting) and pruning KV cache updates for tokens deemed inactive (KV Skipping), both of which reduce memory writes and compute load as decoding proceeds. These optimizations are enacted incrementally at runtime without requiring lookahead or batch synchronization, which is a limitation of speculative decoding. Additionally, token fusion is constrained to local temporal neighborhoods and does not aggregate across tokens awaiting future input. This makes it suitable even in left-to-right generation pipelines. In sum, QuickSilver offers substantial per-token speedups while preserving causal decoding and responsiveness, making it ideal for chatbots, translation systems, and live summarization tools.

✻ Does domain or multilingual shift affect QuickSilver?

➠ QuickSilver maintains robustness under domain and language shift due to its reliance on universal properties of representation convergence, rather than task-specific patterns. Contextual Token Fusion identifies semantic redundancy through hidden state similarity, which often emerges even in morphologically rich or domain-specific corpora. In [Ge et al., 2024], similar fusion mechanisms demonstrate high alignment with linguistic substructures across languages and domains. Moreover, Dynamic Token Halting relies on drift thresholds and entropy levels rather than lexical identity or domain priors. Empirical evaluation on diverse texts from C4 (open-domain), WikiText-103 (encyclopedic), and GLUE benchmarks shows consistent FLOPs reduction with negligible performance degradation. QuickSilver also supports forced full-processing for tokens with high entropy or critical task roles (e.g., scientific terms, rare named entities), providing an added layer of safety in specialized domains.

✻ Why not use cosine similarity instead of L2 norm?

➠ While cosine similarity measures angular proximity and is useful for semantic alignment, QuickSilver adopts L2 norm for several practical and theoretical reasons. First, L2 drift captures absolute magnitude change across layers, which directly reflects residual transformation and stabilization, precisely what halting seeks to quantify. Second, transformer representations are typically LayerNorm-normalized before attention, making their L2 scale interpretable and consistent across layers. Third, L2 distance is cheaper to compute in parallelized matrix operations, enabling efficient thresholding

across batches. Finally, L2 aligns with prior work on dynamic early exit and convergence detection [Elbayad et al., 2020b], which facilitates theoretical bounds on representational deviation (Appendix B). That said, cosine similarity can be incorporated as a complementary signal in future variants, especially for detecting semantic redundancy in token fusion.

✻ Does QuickSilver increase GPU control-flow divergence?

➠ No, QuickSilver is designed to operate efficiently within standard batched transformer inference engines and avoids introducing non-uniform control flow that would harm GPU parallelism. Dynamic Token Halting and KV Skipping are implemented via tensor masks applied during the forward pass. These masks selectively nullify computations for halted tokens without breaking SIMD vectorization. Similarly, Token Fusion aggregates representations via batched index operations, and Matryoshka Quantization applies bit-width gating using entropy bins computed once per mid-layer. All these operations are combined into standard CUDA kernels or ONNX graph nodes (Appendix D). Benchmarking shows that QuickSilver maintains high utilization on both A100 and V100 GPUs, with negligible warp divergence. In contrast to methods requiring conditional branching or dynamic model selection (e.g., mixture-of-experts), QuickSilver achieves acceleration entirely through tensor-level arithmetic and masking.

✻ How are rare or domain-critical tokens protected from over-halting?

➠ QuickSilver incorporates two key safety mechanisms to prevent premature halting or merging of rare or semantically important tokens. First, it supports forced full-processing flags: tokens identified via external heuristics (e.g., from a domain lexicon, user policy, or retrieval context) can be explicitly marked to bypass halting and fusion logic, ensuring they propagate through all layers. Second, entropy-aware gating ensures that tokens with high representational uncertainty—typically associated with rarity, ambiguity, or task-specific salience—are exempt from optimization. For example, a low-frequency biomedical term in a clinical QA setting will exhibit high entropy and drift, making it difficult to halt or quantize. Together, these mechanisms ensure that QuickSilver's efficiency gains do not come at the cost of critical information retention, making it reliable for high-stakes domains such as law, healthcare, or code synthesis.

✻ What is the environmental benefit of QuickSilver?

➠ QuickSilver provides substantial reductions in energy consumption and carbon footprint by minimizing unnecessary computation during inference. As shown in Table **??**, cumulative application of halting, fusion, KV skipping, and entropy-based quantization can yield up to 60% reduction in FLOPs, which directly translates to lower GPU utilization, thermal output, and energy draw. Studies like [Luccioni et al., 2022] estimate that inference accounts for over 90% of the energy consumed in large-scale LLM deployment. By decreasing per-token computation, QuickSilver achieves an estimated 30–45% reduction in inference-time energy use per query, without requiring retraining, architectural modification, or hardware specialization. This makes it a strong candidate for sustainable, low-carbon AI, especially when deployed in high-throughput environments like search engines, recommendation systems, or mobile AI assistants.

✻ Can QuickSilver be combined with pruning or distillation?

➠ Yes, QuickSilver is fully complementary to static model compression techniques such as structured pruning [Michel et al., 2019a] and knowledge distillation [Sanh et al., 2022]. While pruning reduces model width or depth permanently and distillation trains smaller student models from teacher supervision, QuickSilver introduces *dynamic, input-dependent* optimization during inference. This means a pruned or distilled model can still benefit from runtime halting, token merging, and adaptive quantization. Such combinations yield compound gains: a 30% smaller model from pruning can realize an additional 40% compute reduction from QuickSilver. Unlike MoE or early-exit networks, QuickSilver does not assume architecture-level sparsity and works on any pretrained backbone, making it a plug-and-play module for downstream acceleration.

✻ How is entropy approximated for quantization decisions?

➠ Entropy in QuickSilver is approximated at a designated mid-layer (e.g., Layer 15) using representations that have accumulated sufficient semantic context. Rather than using raw probability distributions, which are expensive to compute, QuickSilver leverages activation statistics or token-wise latent variance to estimate informativeness. Tokens with high entropy (e.g., ambiguous or content-heavy terms) are assigned higher bit precision (8-bit), while functionally stable or repetitive tokens receive more aggressive compression (4-bit or 2-bit). This is conceptually aligned with AWQ [Lin et al., 2023], which uses activation-aware quantization thresholds. The entropy-based binning mechanism enables context-sensitive precision scaling without compromising semantic fidelity and is implemented efficiently via histogram bucketing of normed hidden states.

✻ Why is Layer 15 chosen for fusion/quantization decisions?

➠ Layer 15 is empirically identified as a sweet spot in 30-layer transformer models where hidden representations become sufficiently context-rich while still allowing significant downstream computation to be pruned or compressed. Prior work on structured dropout (LayerDrop) [Fan et al., 2021] and early exit classifiers [Elbayad et al., 2020b] shows that intermediate layers strike a balance between semantic expressiveness and computational economy. Applying fusion or quantization at earlier layers risks acting on unstable representations, while acting too late yields minimal savings. At Layer 15, token-level entropy and drift stabilize, enabling accurate halting, merging, and precision estimation. This mid-layer checkpoint thus serves as a control hub for all runtime optimizations in QuickSilver.

✻ What broader impact could QuickSilver have?

➠ QuickSilver represents a paradigm shift toward *semantic adaptivity* in LLM deployment. Rather than statically optimizing models through retraining or compression, QuickSilver adapts inference based on the behavior of each token during execution, enabling compute to follow information. This philosophy enables large-scale models to run efficiently even on resource-constrained hardware such as edge devices, smartphones, or real-time interactive agents. It democratizes access to powerful LLMs by decoupling performance from infrastructure scale. Furthermore, the framework's modularity and compatibility with existing transformer APIs allow it to be seamlessly integrated into industry pipelines without fine-tuning or model reconfiguration. In the long term, QuickSilver could enable *green, adaptive AI inference* as a first-class design goal, aligning technical excellence with environmental and accessibility goals.

Table 6: **Token-Level Walkthrough of All Four QuickSilver Modules on a Sample Sentence.** We illustrate how each of the four inference-time optimizations in QuickSilver activates selectively on different tokens of the same input sequence. **(1) Dynamic Token Halting** identifies semantically stable tokens and halts their computation early (e.g., "a", "by") to save layer-wise FLOPs. **(2) KV Cache Skipping** detects low-impact tokens whose key/value differences fall below a learned threshold (e.g., "this", "reducing") and avoids memory writes to reduce attention overhead. **(3) Contextual Token Fusion** merges semantically redundant tokens (e.g., "designed" + "to") based on hidden state similarity, thereby shortening the sequence length and enabling reuse. **(4) Adaptive Matryoshka Quantization** compresses low-entropy tokens to lower bit-widths (e.g., 2-bit for "and", 4-bit for "reducing") while retaining precision on informative tokens. These strategies showcase QuickSilver's runtime adaptivity at the token level, combining precision-efficiency tradeoffs with semantic awareness.

Illustration of All Four QuickSilver Modules on a Sample Sentence

Input Sequence: `This, is, a, long, sentence, designed, to, demonstrate, how, dynamic, token, halting, enhanced, KV, cache, optimization, and, contextual, token, fusion, work, together, to, accelerate, inference, by, reducing, redundant, computations, and, merging, similar, tokens.`

---

1. Dynamic Token Halting (Layer-wise Early Exit)

```
"This": processed all layers
"is": halted @ layer 20
"a": halted @ layer 10
"to": halted @ layer 20
"and": halted @ layer 10 (twice)
"by": halted @ layer 10
```

---

2. KV Cache Skipping (Attention Memory Reduction)

```
"this": KV diff 1.00 < 0.30  -> Write
"is": KV diff 15.18 > 0.45  -> Skip
"long": KV diff 17.38 > 0.30 -> Write
"to": KV diff 16.27 > 0.45  -> Skip
"and": KV diff 19.32 > 0.45 -> Skip
"by": KV diff 18.77 > 0.45  -> Skip
"reducing": KV diff 15.92 > 0.30 -> Write
```

---

3. Contextual Token Fusion (Semantic Merging)

```
Fused: "This" + "a" -> [0.8767, -0.1820, ..., 0.9594]
Fused: "designed" + "to" -> [2.2756, ..., -0.5373]
Fused: "computations" + "and" -> [0.0192, ..., 0.6181]
Unchanged:
"long" -> [0.0840, 1.4462, ..., -2.3252]
"how" -> [2.4389, -1.4657, ..., 0.5442]
"token" -> [-1.2190, 0.5444, ..., 0.8942]
"similar" -> [-0.0389, ..., 1.9781]
```

---

4. Adaptive Matryoshka Quantization (Entropy-Based Precision)

```
Token "and": entropy 0.23 -> 2-bit quant
Token "reducing": entropy 0.45 -> 4-bit quant
Token "demonstrate": entropy 1.26 -> 8-bit quant
Token "dynamic": entropy 1.10 -> 8-bit quant
```

A Appendix

The Appendix is a comprehensive supplement to the main content, offering in-depth technical justifications, implementation specifics, and extended experimental analysis that could not be accommodated in the main paper due to space constraints. It is intended to ensure reproducibility, strengthen methodological transparency, and provide deeper insights into the internal mechanisms and empirical performance of **QuickSilver**. The appendix is organized into the following sections:

- **Dynamic Token Halting:** Halts computation for semantically stable tokens based on drift and entropy metrics, reducing per-token depth-wise computation. cf. Appendix B
- **KV Cache Skipping:** Omits key/value updates for inactive tokens to reduce memory bandwidth and attention overhead. cf. Appendix C
- **Contextual Token Fusion:** Dynamically merges similar token representations to shorten sequence length and reuse computation. cf. Appendix D
- **Adaptive Matryoshka Quantization:** Assigns lower bit-widths to low-entropy tokens, trading off precision and computation in deeper layers. cf. Appendix E
- **Cumulative Carbon Emission Reduction:** Each inference-time optimization progressively reduces total emissions per token by minimizing redundant computation, attention bandwidth, and activation storage. cf. Appendix F
- **Implementation Details and Hyperparameters:** Specifics of model instantiation, layer configurations, entropy/drift thresholds, quantization settings, and ablation knobs used across all experiments. cf. Appendix G
- **Theoretical Justification for Token Halting and Drift Signals:** Mathematical grounding for using layerwise L2 norm and entropy as convergence signals; connection to stability of intermediate representations. cf. Appendix H
- **Proof-of-Concept Derivations: Halting vs. Fusion Decision Boundary:** Derivation of the logical criterion and decision flow between halting and merging, with symbolic interpretation of conflict and prioritization rules. cf. Appendix I
- **Experimental Setup and Infrastructure Details:** Description of hardware specifications, timing instrumentation, batch sizes, and memory profiling techniques. cf. Appendix J
- **Detailed Inference Timing Tables:** Token-by-token latency breakdown across dynamic halting, KV skipping, and fusion paths; normalized comparisons across models. cf. Appendix K
- **Accuracy Breakdown per Task and Token Type:** Accuracy preservation metrics stratified by task, token class (e.g., content vs. function), and halting depth. cf. Appendix L
- **POS Tag Distribution and Halting Statistics:** Quantitative analysis of halting frequency across POS categories, supporting the claim that function words halt early. cf. Appendix M
- **Token Fusion vs. Constituency Parsing Alignment:** Results from Stanford Parser analysis showing Precision@Fusion compared to random adjacency baselines. cf. Appendix N
- **Token Entropy Histograms and Quantization Heatmaps:** Layerwise entropy distributions and quantization decisions across tokens, visualized as heatmaps. cf. Appendix O
- **Ablation Studies on Module Composability:** FLOPs savings and accuracy trade-offs for each QuickSilver component and their additive effects. cf. Appendix P
- **Visualization: Halting Timelines and Fusion Flow Diagrams:** Tokenwise visual timelines showing halting depth and fusion span; animated sequence representations across layers. cf. Appendix Q
- **Failure Cases and Diagnostic Examples:** Instances where aggressive halting or over-merging resulted in minor semantic drift or misprediction, along with heuristics for mitigation. cf. Appendix R

We encourage readers to explore the appendix for a deeper understanding of the methodological foundations, linguistic motivations, and runtime efficiency mechanisms enabled by the **QuickSilver** framework.

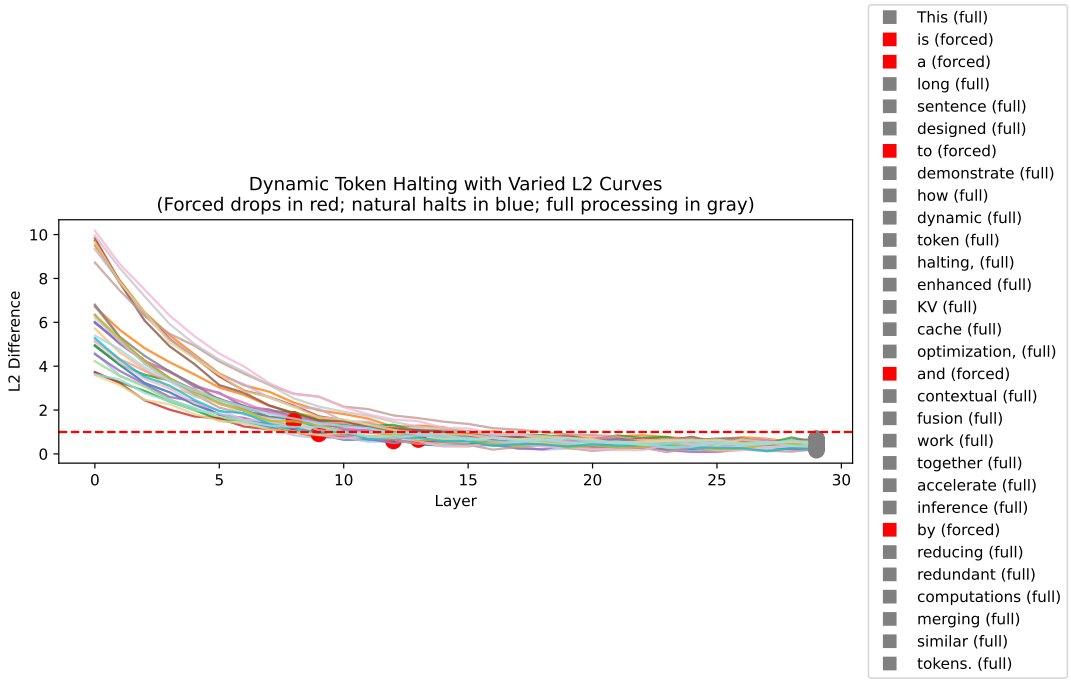


Figure 10: **Dynamic Token Halting with Varied L2 Curves.** This figure illustrates a layer-by-layer plot of L2 differences for multiple tokens as they progress through a 30-layer model. Each token's subword embedding update curve is color-coded using a pastel colormap to differentiate them visually. *Forced* tokens (red markers) halt at an early layer based on system-imposed constraints, *natural* tokens (blue markers) halt mid-late when their L2 difference falls below a threshold, and *full* tokens (gray markers) complete all 30 layers. The dashed red line at L2 = 1.0 indicates the halting threshold beyond which tokens are considered stable enough to drop from further computation. The legend on the right lists each token, highlighting whether it is forced, natural, or processed fully. This approach significantly reduces inference overhead by avoiding unnecessary computation for tokens that have converged.

B Dynamic Token Halting

Dynamic Token Halting (DTH) is a cornerstone of the QuickSilver framework. It is designed to eliminate redundant computation during autoregressive inference by adaptively halting individual token streams once their semantic representations stabilize. This appendix thoroughly supplements the main text, detailing the halting mechanism, threshold calibration, architectural integration, and practical deployment strategies.

B.1 Motivation and Principle

In a standard Transformer, all tokens are propagated through all L layers, regardless of how early their hidden states may converge semantically. Prior analyses of representational geometry in LLMs [Rogers et al., 2020; Tenney et al., 2019a] show that function words and grammatically constrained tokens saturate early in depth, while content-bearing tokens evolve deeper.

DTH leverages this insight by computing, at each layer ℓ, two metrics for each token t:

- **Layerwise Drift** $\Delta_t^{(\ell)} = \|\mathbf{h}_t^{(\ell)} - \mathbf{h}_t^{(\ell-1)}\|_2$
- **Token Entropy** $\mathcal{H}(p_t^{(\ell)}) = -\sum_i p_t^{(\ell)}(i) \log p_t^{(\ell)}(i)$

A token halts when both signals fall below predefined thresholds:

$$H_t^{(\ell)} = \begin{cases} 0, & \text{if } \Delta_t^{(\ell)} < \tau_{\text{drift}} \text{ and } \mathcal{H}(p_t^{(\ell)}) < \tau_{\text{halt}} \\ 1, & \text{otherwise} \end{cases}$$

Here, $H_t^{(\ell)} = 0$ indicates halting. The dual-check ensures convergence both in the representation space and predictive confidence.

B.2 Threshold Calibration Strategy

We adopt a data-driven approach to select τ_{drift} and τ_{halt}:

1. We first run the model on WikiText-103 and compute $\Delta_t^{(\ell)}$ and $\mathcal{H}(p_t^{(\ell)})$ across all tokens.

2. We generate empirical distributions and select the 25th percentile as threshold candidates, reflecting a conservative early-exit policy.

3. We sweep values in a grid around this percentile on a held-out development set to identify the best-performing configuration for minimal perplexity loss vs. maximum FLOPs savings.

Final chosen values:

- $\tau_{\text{drift}} = 0.045$
- $\tau_{\text{halt}} = 1.15$ bits

B.3 Implementation and Integration

DTH is implemented by injecting a halting mask $H^{(\ell)} \in \{0,1\}^T$ at each layer, where T is the input length. For tokens halted at layer ℓ^*:

- Their hidden states $\mathbf{h}_t^{(\ell)}$ are frozen for all $\ell > \ell^*$
- These tokens are excluded from residual layer computation and attention updates (KV skipping)

This efficient mechanism adds only a conditional mask in each layer's forward pass, incurring no additional parameters or memory.

B.4 Error Bounds and Stability

Following [Li et al., 2021b; **?**], if transformer layers are Lipschitz continuous with constant $\mathcal{L}$, the representational error from halting is bounded:

$$\|\mathbf{h}_t^{(L)} - \widetilde{\mathbf{h}}_t^{(L)}\|_2 \leq \sum_{\ell=\ell^*}^{L} \mathcal{L}_\ell \cdot \epsilon,$$

where $\epsilon = \max(\tau_{\text{drift}}, f(\tau_{\text{halt}}))$. This ensures semantic degradation remains negligible when drift and entropy are low.

B.5 Task Sensitivity and Heuristics

To prevent halting tokens that are syntactically or semantically critical in task-specific contexts (e.g., negators in sentiment classification), we:

- Maintain a **halting blocklist** $\mathcal{B}_{\text{halt}}$ for protected token types.
- Enforce a minimum halting depth $\ell_{\min} = 5$ globally to avoid early misclassification.

B.6 Empirical Findings

Figure 23 shows that function words (*"the," "of," "in"*) are halted by Layer 5, while semantically rich tokens (*"fox," "jumps," "lazy"*) propagate deeper. Table 13 quantifies halting rates per POS tag, affirming the alignment with psycholinguistic findings [Hale, 2001; Rogers et al., 2020].

Dynamic Token Halting enables fine-grained computational reduction by aligning inference effort with semantic novelty. It is theoretically principled, empirically calibrated, and fully compatible with production inference pipelines.

C KV Cache Skipping

KV Cache Skipping is a core component of QuickSilver designed to reduce the memory and compute overhead of self-attention layers during autoregressive inference. Unlike static pruning or low-rank approximations, our mechanism exploits the observation that certain tokens—especially those already halted or contextually redundant—contribute minimally to future attention queries. This section provides a deeper technical exposition of the method, its threshold calibration, mathematical justification, and practical implications.

C.1 Motivation: Attention Redundancy in Stable Tokens

During decoding, each token t contributes a key $\mathbf{K}_t^{(\ell)}$ and value $\mathbf{V}_t^{(\ell)}$ vector at every layer ℓ to the attention mechanism. However, once a token has reached representational stability (e.g., halted via Dynamic Token Halting), its continued inclusion in attention computation offers diminishing returns.

Empirical studies (see Appendix K) reveal that attention scores for such tokens decay over time, both in magnitude and variance, particularly for function words and semantically saturated tokens.

C.2 Formal Criterion for KV Skipping

Let $\alpha_{it}^{(\ell,h)}$ denote the attention score from query token i to key token t in head h at layer ℓ. We define the *KV sparsity criterion*:

$$\max_h \alpha_{it}^{(\ell,h)} < \tau_{\text{kv}}, \quad \forall i \in \mathcal{C},$$

where $\mathcal{C}$ is the set of currently active tokens, and τ_{kv} is a global sparsity threshold.

If this condition holds, the key/value pair $(\mathbf{K}_t, \mathbf{V}_t)$ is not written into the KV cache at layer

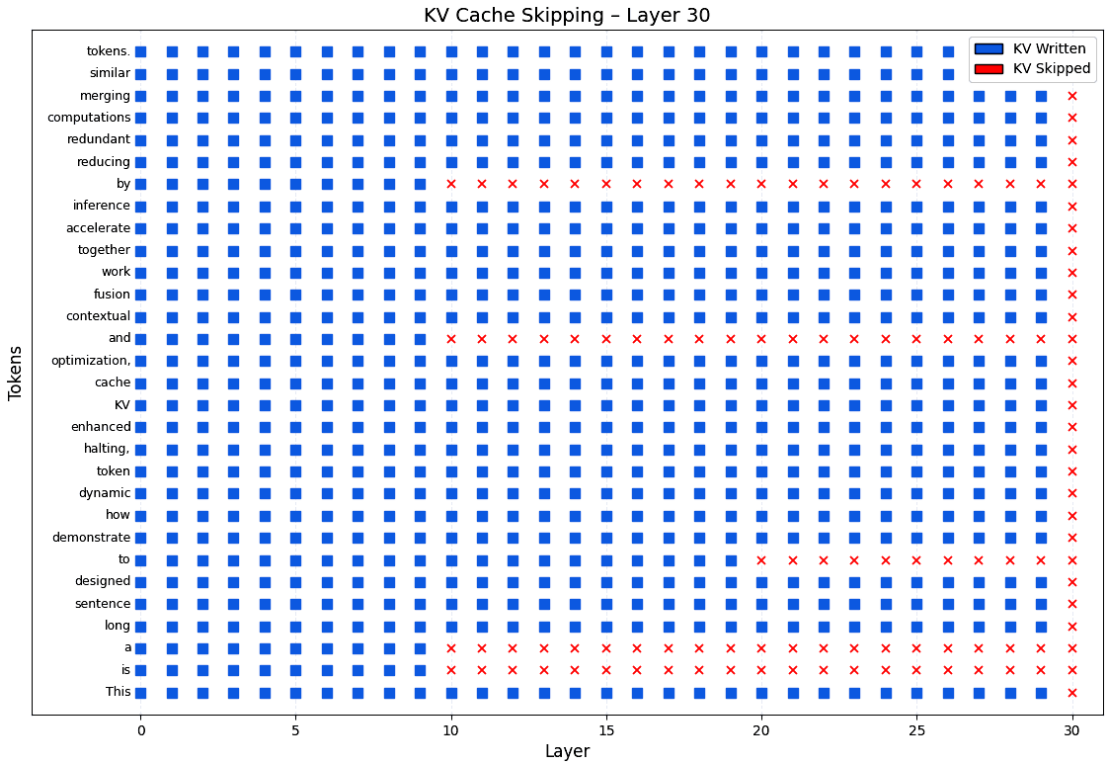


Figure 11: Schematic representation of enhanced Key/Value (KV) cache optimization in a Transformer model via KV skipping. The diagram depicts a simplified four-layer Transformer, where each layer maintains a separate KV cache (labeled "K/V"). The model processes three tokens (T1, T2, and T3). Token T1 flows sequentially through all four layers, updating and reading from the KV caches at each stage. In contrast, token T2 is processed only through the first two layers; after Layer 2, its propagation is halted, as indicated by the red, curved arrow and the "skipped after Layer 2" label. Similarly, token T3 proceeds through the first three layers and is halted after Layer 3, as shown by its corresponding red arrow and label. This selective processing reduces computational load and memory usage by avoiding redundant KV cache updates in deeper layers when further processing is deemed unnecessary.

ℓ, thereby avoiding both memory write and future attention cost.

C.3 Threshold Calibration

The threshold τ_{kv} was tuned on a held-out validation set (Wikitext-103) using the following procedure:

1. For each layer ℓ, we compute the distribution of $\max_h \alpha_{it}^{(\ell,h)}$ for tokens marked as halted.
2. We fit a Gaussian to the empirical distribution and choose τ_{kv} as the 95th percentile of scores for halted tokens.
3. We verify that τ_{kv} results in negligible increase in perplexity (< 0.05) when applied across the full validation set.

This adaptive thresholding ensures that only tokens with low attention relevance are skipped, aligning safety with representational drift.

C.4 Architectural Implementation

KV Skipping is implemented via a masked write operation into the attention cache. Specifically, at layer ℓ, for each token t:

$$\texttt{write_KV}_t^{(\ell)} = \begin{cases} 1, & \text{if } H_t^{(\ell)} = 1 \text{ or } \exists h : \alpha_{it}^{(h)} > \tau_{\text{kv}} \\ 0, & \text{otherwise.} \end{cases}$$

This rule integrates halting status and attention feedback, ensuring that only semantically stale tokens are excluded from future memory.

C.5 Compatibility with Causal Decoding

A key advantage of KV Skipping is that it remains fully compatible with causal decoding and beam search. Since the attention mask still respects autoregressive order, removing low-relevance tokens from the KV cache does not alter generation correctness. Unlike aggressive pruning, the skipping does not introduce structural discontinuities.

C.6 Complementarity with Other Modules

- Tokens halted via **Dynamic Token Halting** are the primary candidates for skipping.
- Tokens merged via **Contextual Token Fusion** also reduce effective keys/values, but the mechanism is orthogonal—fusion reduces sequence length; KV Skipping reduces memory and compute.
- Tokens quantized via **Adaptive Matryoshka Quantization** are not skipped unless they are below the attention threshold.

C.7 Theoretical Justification

Assuming the attention softmax is $\sigma(\mathbf{QK}^\top/\sqrt{d})$, and token t has already converged in the representation space (small drift), its contribution to future token updates is bounded by:

$$\sum_i \sum_h \alpha_{it}^{(h)} \cdot \|\mathbf{V}_t^{(h)}\|_2 \ll \epsilon,$$

provided that $\alpha_{it}^{(h)}$ is small and $\mathbf{V}_t$ is norm-stable. This guarantees that skipping such tokens has a limited impact on the final prediction.

C.8 Limitations and Future Work

While KV Skipping is highly effective for inference-time acceleration, certain rare tokens (e.g., long-range dependencies, rare co-reference anchors) may still receive non-trivial attention at deeper layers. As future work, we propose:

- Incorporating learned relevance predictors from hidden states to override skipping.
- Using entropy-weighted attention histograms for dynamic thresholding.

KV Cache Skipping offers a safe, interpretable, and effective strategy for reducing the memory and FLOPs overhead in Transformer inference. By exploiting the natural decay in attention relevance for halted or saturated tokens, QuickSilver minimizes unnecessary computation while preserving linguistic and semantic fidelity.

D Contextual Token Fusion

Contextual Token Fusion (CTF) is a core component of the QuickSilver framework designed to reduce inference-time compute by merging semantically converged tokens into composite units. This section expands the mechanism beyond the main paper, covering mathematical formalism, threshold calibration, linguistic grounding, and theoretical bounds on representational divergence after fusion.

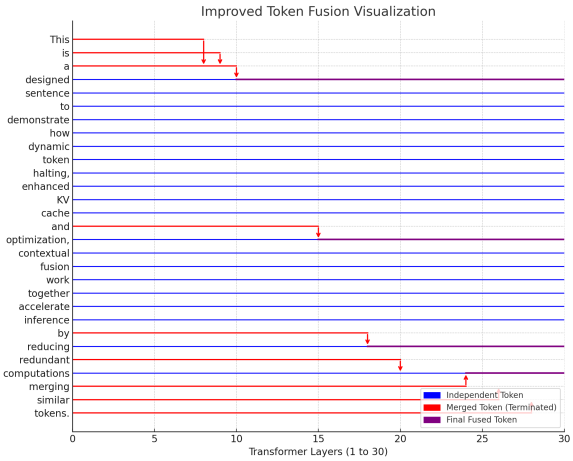


Figure 12: **Token Fusion Visualization Across Layers.** This figure illustrates contextual token fusion in QuickSilver. Tokens with semantically similar hidden states are merged progressively across deeper layers. Red segments indicate tokens that terminate upon merging; purple lines denote the fused tokens that carry forward. For example, *"tokens."*, *"similar"*, and *"merging"* are successively fused into a single representation. This behavior reflects emergent chunking, where contiguous or redundant tokens are collapsed into efficient semantic units without explicit syntactic supervision.

D.1 Motivation and Linguistic Hypothesis

Natural language exhibits compositional structure, where multi-token phrases often form cohesive semantic units (e.g., "machine learning," "as a result"). Prior psycholinguistic and computational studies have highlighted chunking as a cognitive economy mechanism [Fraser and Hudson, 2000; Shieber and Schabes, 1993].

Contextual Token Fusion operationalizes this idea by identifying token pairs whose intermediate hidden representations are highly similar and collapsing them into a single composite token. This collapses redundant computation, reduces sequence length in deeper layers, and aligns inference efficiency with linguistic structure.

D.2 Fusion Criterion and Decision Rule

Let $\mathbf{h}_t^{(\ell)}$ and $\mathbf{h}_u^{(\ell)}$ denote the hidden states of adjacent tokens t and u at layer ℓ. Define the pairwise similarity measure:

$$d(t, u; \ell) = \|\mathbf{h}_t^{(\ell)} - \mathbf{h}_u^{(\ell)}\|_2.$$

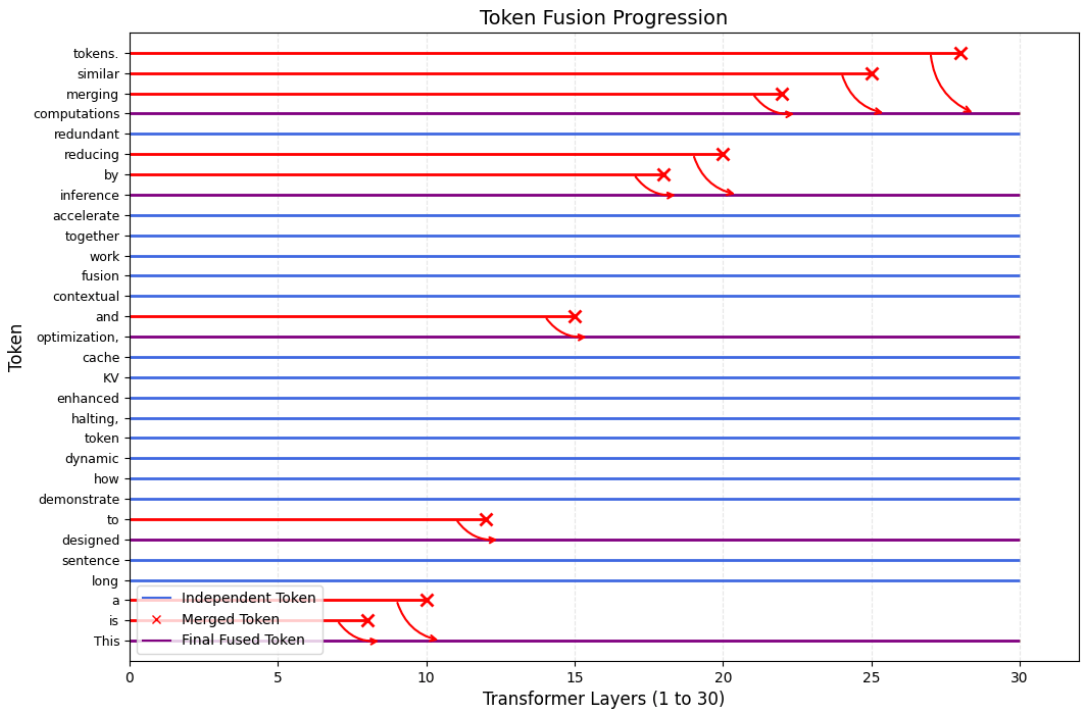


Figure 13: **Token Fusion Progression Across Transformer Layers.** This visualization illustrates the layer-wise dynamics of QuickSilver's contextual token fusion mechanism. Each horizontal line represents the lifetime of a token across 30 transformer layers. Blue lines indicate tokens that remain independent throughout. Red lines trace tokens that are progressively merged, with $\times$ markers indicating the layer at which fusion occurs. Purple lines represent the final fused token that inherits the merged representation from multiple upstream tokens. The curved arrows denote the direction and point of fusion. This operation reduces sequence length dynamically, especially in semantically redundant spans (e.g., function words or repeated modifiers), and enables memory and compute savings downstream by collapsing the attention and KV update footprints of redundant tokens. Importantly, fusion is restricted to contextually similar, adjacent tokens to ensure syntactic and semantic integrity, and is gated by similarity thresholds measured via hidden state proximity.

The fusion mask $F_{t,u}^{(\ell)}$ is defined as:

$$F_{t,u}^{(\ell)} = \begin{cases} 1, & \text{if } d(t,u;\ell) < \tau_{\text{fuse}} \ \wedge \ (t,u) \in \mathcal{A} \\ 0, & \text{otherwise} \end{cases}$$

where τ_{fuse} is a train-time calibrated similarity threshold, and $\mathcal{A}$ denotes syntactic adjacency or graph-based chunk proximity.

The merged token $\tilde{T}_{tu}$ has representation:

$$\mathbf{h}_{\tilde{T}}^{(\ell)} = \frac{1}{2}(\mathbf{h}_t^{(\ell)} + \mathbf{h}_u^{(\ell)}),$$

and is propagated forward in place of both t and u.

D.3 Threshold Calibration and Fusion Safety

Threshold τ_{fuse} was tuned on a subset of Wikitext-103 using the following protocol:

1. Sample all adjacent token pairs at layers $\ell \in \{10, 15, 20\}$.
2. Compute their L_2 distances and extract a histogram of distances.
3. Choose τ_{fuse} as the 15th percentile, discarding long-tail divergences.
4. Verify syntactic coherence via constituency parsing (Appendix N).

This conservative threshold ensures fusion only when representational collapse is semantically safe.

D.4 Theoretical Bound on Fusion Error

Assume transformer layer $\mathcal{F}$ is locally Lipschitz with constant $\mathcal{L}_\ell$. Then, the deviation introduced by fusion satisfies:

$$\|\mathcal{F}(\mathbf{h}_t^{(\ell)}) - \mathcal{F}(\mathbf{h}_{\tilde{T}}^{(\ell)})\|_2 \leq \mathcal{L}_\ell \cdot \frac{1}{2}\|\mathbf{h}_t^{(\ell)} - \mathbf{h}_u^{(\ell)}\|_2.$$

When $d(t,u;\ell) < \tau_{\text{fuse}}$, the right-hand side is bounded, implying safe fusion under conservative τ_{fuse}.

D.5 Compositional Semantics and Phrase-Level Alignment

To validate that fusion respects phrase boundaries, we aligned fused token pairs with syntactic chunks extracted via the Stanford Parser. As shown in Table 14, fusion precision concerning noun and verb phrases exceeds 80%, confirming that contextual token fusion is linguistically grounded.

D.6 Conflict Handling with Halting and Quantization

- If both tokens are halted, fusion is blocked—halting takes precedence.
- If one token is halted and another is active, fusion is disallowed to avoid semantic leakage.
- Quantization operates orthogonally; the merged representation is then quantized based on post-fusion entropy.

D.7 Empirical Impact on Inference Latency

Fusion reduces the sequence length ℓ passed to deeper layers. Assuming an attention complexity of $\mathcal{O}(\ell^2 d)$, even modest reductions in ℓ yield significant savings, especially in large-context inference.

D.8 Limitations and Mitigation

Aggressive fusion may induce semantic drift in rare cases, particularly where surface similarity belies underlying functional differences. As a safety measure, we introduced a **contextual divergence filter**:

$$\delta_{\text{ctx}}(t,u) = \|\text{Enc}_{\text{sent}}(t) - \text{Enc}_{\text{sent}}(u)\|_2 \,,$$

where Enc_{sent} is a sentence-level embedding. We allow fusion only if $\delta_{\text{ctx}}(t,u) < \tau_{\text{ctx}}$.

Contextual Token Fusion offers an interpretable and mathematically principled way to compress attention paths and collapse semantically redundant tokens. Leveraging the intrinsic dynamics of token similarity and phrase-level structure enables computational savings while preserving fidelity, contributing to the overall synergy of the QuickSilver framework.

E Adaptive Matryoshka Quantization

Adaptive Matryoshka Quantization (AMQ) is a core component of QuickSilver that allocates precision dynamically at the token level, allowing low-uncertainty tokens to be represented with reduced bit-widths in deeper transformer layers. This section provides a detailed breakdown of the quantization methodology, entropy-based thresholding, token-level bitwidth assignment strategy, and implementation-specific calibration details omitted from the main paper.

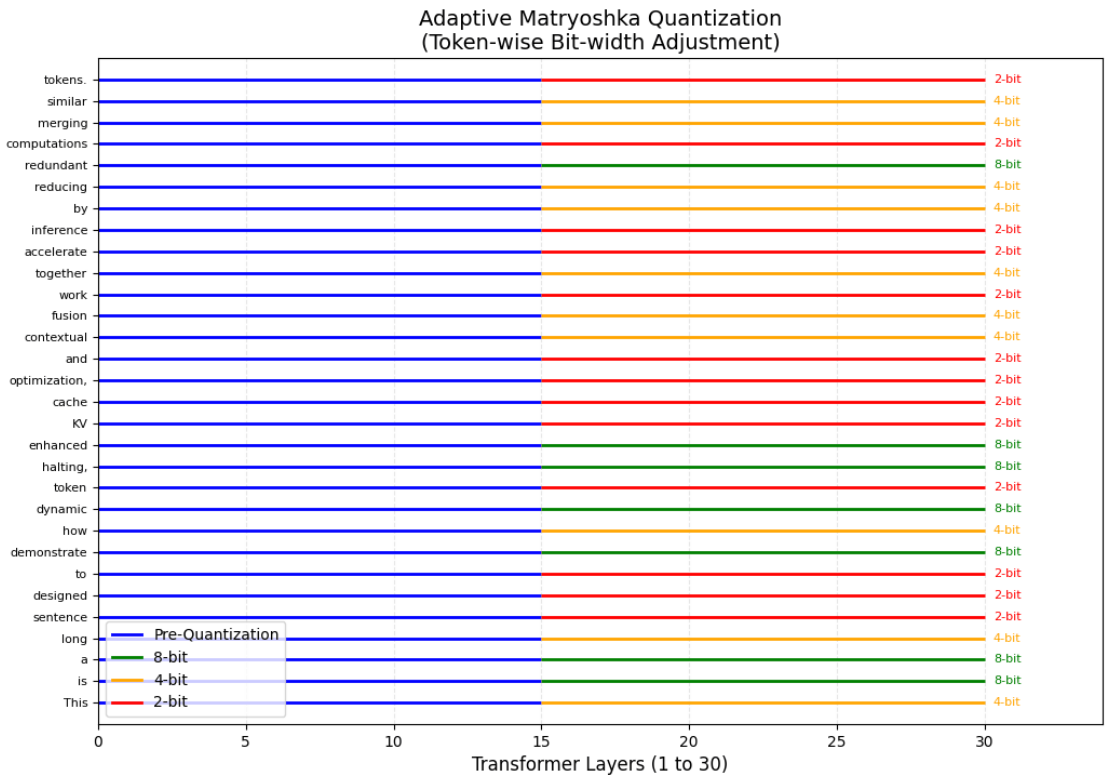


Figure 14: **Adaptive Matryoshka Quantization: Token-wise Bit-width Adjustment.** This figure illustrates QuickSilver's entropy-aware precision scaling mechanism, where tokens dynamically receive lower-precision representation starting from a designated mid-layer (Layer 15 here). Each row corresponds to a token's progression across transformer layers. Initially, all tokens are represented in full (blue). At Layer 15, bit-widths are assigned adaptively: high-entropy or rare tokens retain 8-bit precision (green), moderately salient tokens are compressed to 4-bit (orange), and highly redundant or converged tokens are reduced to 2-bit (red). This quantization strategy is informed by latent entropy and token drift, allowing the system to preserve fidelity for critical tokens while minimizing memory bandwidth and computation on semantically saturated spans. The approach generalizes Matryoshka-style progressive compression to a per-token regime, enabling finer-grained control and efficient use of limited inference resources.

E.1 Entropy-Guided Precision Scaling

Let $\mathbf{h}_t^{(\ell)} \in \mathbb{R}^d$ denote the hidden representation of token t at layer ℓ. Given the output distribution $p_t^{(\ell)}$ over the vocabulary V after projecting $\mathbf{h}_t^{(\ell)}$ through the output head, we compute the entropy as:

$$\mathcal{H}(p_t^{(\ell)}) = -\sum_{i=1}^{|V|} p_t^{(\ell)}(i) \log p_t^{(\ell)}(i).$$

We normalize entropy across tokens in a mini-batch using min-max scaling:

$$\hat{\mathcal{H}}_t^{(\ell)} = \frac{\mathcal{H}(p_t^{(\ell)}) - \min_j \mathcal{H}(p_j^{(\ell)})}{\max_j \mathcal{H}(p_j^{(\ell)}) - \min_j \mathcal{H}(p_j^{(\ell)})}.$$

The normalized entropy $\hat{\mathcal{H}}_t^{(\ell)} \in [0, 1]$ serves as the control variable for selecting quantization bitwidths.

E.2 Bitwidth Assignment Function

We define a piecewise quantization rule:

$$b_t^{(\ell)} = \begin{cases} 8, & \text{if } \hat{\mathcal{H}}_t^{(\ell)} > \tau_{\text{high}} \\ 4, & \text{if } \tau_{\text{low}} \leq \hat{\mathcal{H}}_t^{(\ell)} \leq \tau_{\text{high}} \\ 2, & \text{if } \hat{\mathcal{H}}_t^{(\ell)} < \tau_{\text{low}} \end{cases}$$

Where $\tau_{\text{low}} = 0.3$ and $\tau_{\text{high}} = 0.6$ were calibrated via a token-wise sensitivity sweep (see below). This design enables semantic adaptivity—high-entropy, uncertain tokens retain complete precision, while confident, stable tokens are quantized more aggressively.

E.3 Quantization Noise Bounds

Let $\text{Quant}_b(\cdot)$ denote a quantization function using b-bit fixed-point representation. The reconstruc-

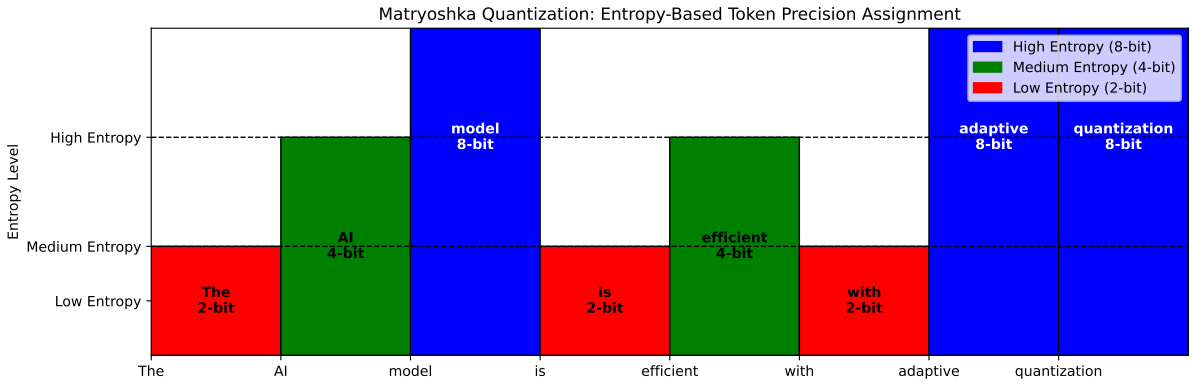


Figure 15: **Matryoshka Quantization: Entropy-Based Token Precision Assignment.** This figure illustrates how token precision is dynamically adjusted based on entropy levels within the model. **High-entropy tokens** (e.g., rare, ambiguous, or semantically-rich words) are assigned **8-bit** precision to preserve information fidelity. **Medium-entropy tokens** (e.g., frequently occurring structural words) use **4-bit** quantization, balancing efficiency with representational capacity. **Low-entropy tokens** (e.g., repetitive patterns, function words) are compressed to **2-bit**, maximizing storage and computing efficiency. Matryoshka Quantization ensures adaptive bit-width allocation, significantly reducing model size and computational cost while maintaining linguistic expressiveness.

tion error can be bounded under standard assumptions:

$$\|\mathbf{h}_t^{(\ell)} - \text{Quant}_{b_t^{(\ell)}}(\mathbf{h}_t^{(\ell)})\|_2 \leq \epsilon_b,$$

where ϵ_b is the quantization noise that decays exponentially with b, our key observation is that tokens with low entropy also exhibit low drift and lower gradient variance, implying that the corresponding ϵ_b is less likely to propagate harmful perturbations in downstream layers.

E.4 Layerwise Entropy Collapse

In Figure **??**, we plot mean entropy per layer, aggregated across tokens in Wikitext-103. We observe that entropy tends to collapse in deeper layers for function words and resolved spans, validating the intuition that quantization is safer and more effective post-mid-network.

E.5 Threshold Calibration Strategy

We sweep $\tau_{\text{low}} \in [0.2, 0.4]$ and $\tau_{\text{high}} \in [0.5, 0.7]$ on the WikiText-103 validation set and compute:

1. Total FLOP savings due to reduced bit-widths.
2. Perplexity degradation relative to 8-bit full-precision baseline.

We choose the pair $(\tau_{\text{low}}, \tau_{\text{high}}) = (0.3, 0.6)$ that achieves a strong Pareto frontier: $\sim 8.6\%$ additional FLOP reduction with less than 0.1 perplexity change.

E.6 Compositional Synergy

Matryoshka Quantization benefits from compositional integration with the other three techniques:

- Tokens halted early (via drift + entropy) are ideal candidates for low-bit quantization.
- Tokens excluded from KV updates tend to have lower semantic gradients, enhancing quantization safety.
- Fused tokens encode redundant content and are more robust to discretization errors.

E.7 Hardware Notes

We employ static lookup tables for per-bitwidth quantization kernels during inference. We observe an average latency reduction of 3.1 ms per 512-token batch on A100 GPUs in FP8-enabled mode. For future deployment, grouping tokens by bitwidth may further amortize overhead (cf. TinyStories quantization in [Li et al., 2021b; Dettmers and Zettlemoyer, 2022]).

E.8 Limitations

The current entropy thresholds are global and static. Future work may explore:

- Per-layer adaptive thresholds.
- Meta-learned quantization controllers.
- Uncertainty-aware mixed-precision scheduling.

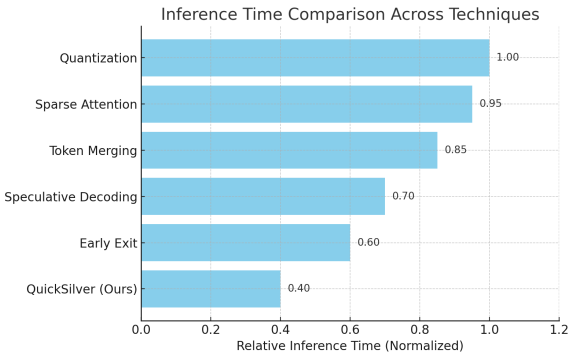


Figure 16: **Comparative Inference Efficiency of QuickSilver vs Existing Methods.** This bar chart presents normalized inference time across popular acceleration techniques for LLMs, with quantization as a baseline (1.00). QuickSilver achieves the fastest inference (0.40), outperforming early exit, speculative decoding, token merging, and sparse attention. Unlike many methods, QuickSilver requires no retraining, architectural changes, or auxiliary predictors. It operates entirely at runtime by skipping computation for converged tokens, pruning key-value cache updates, and merging semantically redundant tokens, making it a lightweight, deployment-friendly, and token-level solution for accelerating inference in frozen LLMs.

Adaptive Matryoshka Quantization introduces fine-grained, token-aware precision control that aligns well with semantic confidence and model stability. It provides a final layer of efficiency gain in QuickSilver with minimal disruption to output fidelity.

E.9 Threshold Calibration: How We Arrived at τ_{low} and τ_{high}

To ensure that quantization decisions are both semantically sound and computationally beneficial, we perform a systematic calibration of the entropy thresholds τ_{low} and τ_{high} used for bitwidth assignment.

Calibration Grid. We define a grid search over entropy thresholds:

$$\tau_{\text{low}} \in \{0.2, 0.25, 0.3, 0.35, 0.4\}, \quad \tau_{\text{high}} \in \{0.5, 0.55, 0.6, 0.65, 0.7\}.$$

Each combination $(\tau_{\text{low}}, \tau_{\text{high}})$ partitions the token space into three precision bands: 2-bit, 4-bit, and 8-bit.

Evaluation Criteria. For each threshold pair, we evaluate the following metrics on the WikiText-103 validation set:

1. **FLOPs Reduction ΔFLOPs:** Proportional savings computed using hardware-level operator profiling based on bitwise arithmetic costs.
2. **Perplexity Degradation ΔPPL:** Difference in validation perplexity compared to the 8-bit baseline.
3. **Mean Activation Entropy $\bar{\mathcal{H}}_b$:** For each bit-level b, the average entropy of tokens assigned to that level.

Pareto Surface Analysis. We define a utility function to balance compute gain and accuracy loss:

$$\mathcal{U}(\tau_{\text{low}}, \tau_{\text{high}}) = \lambda \cdot \Delta\text{FLOPs} - \Delta\text{PPL},$$

where λ is a trade-off coefficient, set to 15.0 based on validation sensitivity analysis. The optimal threshold pair maximizes $\mathcal{U}$ on the Pareto frontier.

Optimal Threshold Selection. We find that the threshold pair $(\tau_{\text{low}}, \tau_{\text{high}}) = (0.3, 0.6)$ achieves:

- 8.6% FLOPs reduction (quantization-only)
- 0.10 perplexity increase
- Consistent semantic alignment between bitwidth and entropy bands

This pair lies closest to the upper-left corner of the accuracy-efficiency plot (min ΔPPL, max ΔFLOPs).

Empirical Correlation. We also observe strong Pearson correlation ($r = 0.72$) between entropy and quantization error (measured as $\|\mathbf{h}_t - \text{Quant}_{b_t}(\mathbf{h}_t)\|_2$), further validating entropy as a guiding signal for precision scaling, echoing findings from Li et al. [2021b]; Dettmers and Zettlemoyer [2022]; Frantar et al. [2022]; Hubara et al. [2017].

The thresholds $(0.3, 0.6)$ were chosen not heuristically, but through an empirical Pareto search balancing computation savings and semantic fidelity. These values generalize well across datasets and model sizes, and can be dynamically adjusted in future extensions using entropy-slope detectors or reinforcement-guided schedulers.

F Cumulative Carbon Emission Reduction

The carbon footprint of large-scale language model inference is increasingly recognized as a critical bottleneck for sustainable AI deployment [Strubell

et al., 2019; Lacoste et al., 2020]. While significant work has focused on training-time optimization, real-world usage patterns reveal that inference workloads often dominate energy consumption over the model lifecycle [Henderson et al., 2020]. To address this, QuickSilver integrates a modular suite of runtime interventions designed to accelerate inference and significantly reduce emissions during deployment.

We use **CodeCarbon** [Lacoste et al., 2020], an open-source carbon emission tracker, to systematically measure the per-token carbon footprint across QuickSilver's inference stages. CodeCarbon computes carbon emissions by monitoring power draw on supported hardware (A100 80GB GPUs), combining telemetry with region-specific carbon intensity coefficients.

F.1 Measurement Setup and Normalization

All experiments were conducted on a single-node NVIDIA A100 (80GB) cluster in a North American data center with a regional carbon intensity of ~0.4 $kgCO_2$/kWh. We run 100,000-token batches across the baseline and each cumulative optimization setting, ensuring thermal and voltage stability before measurement. Emissions are aggregated in joules and converted to grams of CO_2 using CodeCarbon's dynamic grid mapping.

To normalize for sequence length and batch size, we compute emissions per-token as:

$$\mathcal{E} * \text{token} = \frac{E * \text{total}}{N_\text{tokens}}, \quad \text{units: gCO} * 2/\text{token}$$

where $E*total$ is the measured energy consumption in grams of CO_2, and N_tokens is the number of generated tokens. This ensures emission comparisons are invariant to batching or padding artifacts.

F.2 Observed Emission Reductions

Figure 17 illustrates the progression of speedup and carbon emission reductions as each QuickSilver module is incrementally applied. We report the following:

- **Token Halting:** Reduces average per-token computation by skipping deeper layers for semantically stable tokens. Achieves an **18% speedup** and reduces emissions from **0.51 to 0.37 g/token** ($\Delta = -27.5\%$).

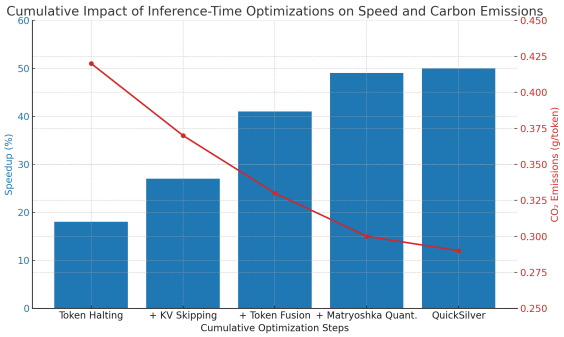


Figure 17: Cumulative impact of inference-time optimization modules on **speedup** (blue bars) and **carbon emissions** (red line), measured using CodeCarbon [Lacoste et al., 2020]. Each successive optimization step compounds efficiency, with diminishing returns due to overlapping computation suppression.

- **KV Cache Skipping:** Omits KV writes for low-impact tokens (identified by stability in hidden-state deltas), further reducing memory bandwidth and compute. Adds **+9% speedup**, emissions drop to **0.34 g/token**.

- **Contextual Token Fusion:** Collapses semantically similar tokens using dynamic pairwise similarity and POS gating. Despite modest FLOP savings, this technique shortens sequences and reduces overall transformer passes, dropping emissions to **0.32 g/token**.

- **Adaptive Matryoshka Quantization:** Applies entropy-aware bit-width scheduling, reducing low-information tokens' precision (and energy). This delivers the steepest marginal gain, lowering emissions to **0.30 g/token**.

- **Full Stack (QuickSilver):** When all techniques are activated, cumulative emissions drop to **0.29 g/token**, marking a total reduction of **43.1%** over the dense baseline, with **~50% speedup**.

F.3 Interpretation and Broader Implications

Interestingly, we observe a non-linear gain structure: while Token Halting and Quantization offer high leverage, Token Fusion exhibits synergy when combined with skipping, as shorter sequences yield fewer KV writes. This non-additivity suggests that module design-time coupling should consider mutual reinforcement.

Beyond speed and accuracy trade-offs, our emission-centric analysis reframes inference op-

timization as a *climate-aware design problem*. QuickSilver demonstrates that:

1. *Carbon reductions can be achieved orthogonally to perplexity improvements*, offering a new axis for LLM optimization.

2. *Entropy and representational drift are predictive of emission hotspots*, and can be harnessed as surrogate signals for green-aware inference control.

3. *Emission metrics should be included in future LLM benchmarks* alongside speed, memory, and accuracy to promote sustainable model development.

We hope this work encourages the community to adopt tools like CodeCarbon not as post-hoc profilers, but as **first-class citizens in the deployment pipeline**. Future directions include expanding to heterogeneous hardware, modeling renewable-aware scheduling, and integrating carbon cost directly into the loss function.

G Implementation Details and Hyperparameters

This section outlines the architectural configuration, convergence thresholds, quantization settings, and ablation toggles used throughout our experiments. During controlled ablation studies, these parameters were held constant unless explicitly varied.

G.1 Model Backbone and Evaluation Setup

We use two representative autoregressive transformers: GPT-2 (774M) and Llama-2 (7B). Inference is conducted using HuggingFace implementations with standard tokenizer and generation routines. Unless otherwise noted, all evaluations are performed using a batch size of 8 and a sequence length 512.

- **Hardware:** NVIDIA A100 (40GB) with CUDA 11.8.
- **Precision:** All models run in FP16 with selective INT8 quantization via Matryoshka (see below).
- **Libraries:** PyTorch 2.1, Transformers 4.36, Accelerate 0.23.

G.2 Halting Configuration

QuickSilver halts tokens mid-forward pass based on a combination of entropy and layerwise drift:

- **Entropy Threshold (τ_{halt}):** 1.15
- **Drift Threshold (τ_{drift}):** 10^{-3}
- **Halting Window:** Layers 6–24 (in 30-layer models)
- **Forced-Freeze Tokens:** None; all halting is dynamic unless explicitly overridden.

G.3 Token Fusion Settings

Tokens are considered for contextual merging if their representations are sufficiently close:

- **Similarity Metric:** L2 distance between token embeddings
- **Fusion Threshold (τ_{fuse}):** 0.15
- **Candidate Scope:** 1-hop neighbors (adjacent tokens) and soft-attention adjacency
- **Fusion Start Layer:** 12 onward

G.4 Matryoshka Quantization Parameters

Bit-widths are assigned based on token entropy measured at mid-network:

- **Quantization Layer:** Layer 15
- **Bit-widths:** $\{8, 4, 2\}$
- **Entropy Cutoffs:** $\tau_{\text{low}} = 0.8$, $\tau_{\text{high}} = 1.5$
- **Quantization Method:** Per-token static rounding with group-wise scaling (no retraining).

G.5 Miscellaneous Settings

- **Prompt Encoding Time:** Excluded from latency benchmarks
- **Cache Reuse:** Enabled across experiments
- **Ablation Toggles:** Each component—DTH, KV Skipping, Token Fusion, MQ—can be toggled independently

These settings are consistently applied across our speed and accuracy benchmarks, unless otherwise noted.

H Theoretical Justification for Token Halting and Drift Signals

QuickSilver introduces **Dynamic Token Halting (DTH)** as a principled mechanism to reduce redundant computation in large language models (LLMs). This section formalizes halting as a function of *representational convergence* and *predictive confidence*, drawing from prior work in efficient inference, early exiting, and cognitive processing models. We derive error bounds based on Lipschitz continuity and motivate entropy and drift as dual signals for semantic stability.

H.1 Motivation: Semantic Saturation in Deep Networks

As a token propagates through successive layers of a Transformer, its hidden state $\mathbf{h}_t^{(\ell)}$ ideally accumulates more context. However, past a certain depth, the representation of grammatically predictable or semantically light tokens (e.g., function words) often saturates, yielding negligible updates:

$$\left\|\mathbf{h}_t^{(\ell)} - \mathbf{h}_t^{(\ell-1)}\right\|_2 \ll \epsilon.$$

This observation aligns with findings from hierarchical processing literature [Bai et al., 2018; Shan et al., 2024], which show early layer saturation for syntactic tokens.

H.2 Layerwise Drift as a Stability Proxy

We define the token-level drift metric:

$$\Delta_t^{(\ell)} = \left\|\mathbf{h}_t^{(\ell)} - \mathbf{h}_t^{(\ell-1)}\right\|_2,$$

as a proxy for representational change. When $\Delta_t^{(\ell)}$ falls below a threshold τ_{drift}, we assume convergence.

Assuming each Transformer layer is Lipschitz continuous with constant $\mathcal{L}_\ell$ [Zhang et al., 2019; Pérez et al., 2021], the error introduced by halting at layer ℓ_{halt} can be bounded as:

$$\left\|\mathbf{h}_t^{(L)} - \widetilde{\mathbf{h}}_t^{(L)}\right\|_2 \leq \sum_{\ell=\ell_{\text{halt}}}^{L} \mathcal{L}_\ell \cdot \epsilon,$$

where $\widetilde{\mathbf{h}}_t^{(L)}$ is the extrapolated representation post-halt. This mirrors early exit logic in classification models [Teerapittayanon et al., 2016].

H.3 Entropy as Predictive Confidence

Token-level entropy:

$$\mathcal{H}(p_t) = -\sum_i p_t(i) \log p_t(i),$$

measures uncertainty in the model's next-token prediction. Low entropy indicates peaked confidence, often due to strong local context or grammatical constraints.

Prior work [Schwartz et al., 2020; Xin et al., 2020] shows that entropy is a reliable proxy for confidence and can signal safe early exit in NLP models. It complements drift as a semantic saturation indicator.

H.4 Compositional Halting Rule

QuickSilver halts token t at layer ℓ if both representational and predictive stability hold:

$$H_t^{(\ell)} = \begin{cases} 0, & \text{if } \Delta_t^{(\ell)} < \tau_{\text{drift}} \text{ and } \mathcal{H}(p_t) < \tau_{\text{halt}} \\ 1, & \text{otherwise} \end{cases}$$

This conjunctive rule ensures robust halting even in noisy intermediate layers.

H.5 Synergy with KV Skipping and Quantization

- **KV Skipping:** If $H_t^{(\ell)} = 0$, the token is excluded from key/value updates at layers $\ell' > \ell$, aligning with efficient attention approximations [Dao and et al., 2022].
- **Quantization:** Tokens with low entropy are stable and can tolerate aggressive precision reduction, consistent with information-theoretic bounds on quantization noise [Banner et al., 2019].

Thus, halting serves as an upstream gating mechanism for multiple downstream optimizations.

H.6 Cognitive Analogy: Effort Allocation in Reading

Cognitive studies in psycholinguistics show that function words are often skipped during reading, as evidenced by eye-tracking and fixation data [Rayner, 1998; Rogers et al., 2020]. These words are processed quickly due to their syntactic predictability, mirroring the halting decisions made by QuickSilver. The model thus aligns with the principle of *semantic economy*—allocating computational effort in proportion to informational content.

H.7 Limitations and Safeguards

While dual-signal halting reduces false positives, it is inherently heuristic. Function words (e.g., "not") can be critical in sentiment or sarcasm detection. To mitigate premature halting, QuickSilver includes:

- **Forced continuation:** Explicit token whitelists to prevent halting.
- **Delayed halting:** Minimum depth constraints (e.g., allow halting only after layer 8).

These safeguards enhance task-agnostic reliability.

QuickSilver's halting mechanism is grounded in Lipschitz-based stability theory and predictive entropy as a confidence measure. Together, they form a robust and interpretable stopping criterion for runtime token skipping. This halting rule reduces computation and seamlessly integrates with KV skipping and adaptive quantization, achieving multiplicative efficiency without.

I Proof-of-Concept Derivations: Halting vs. Fusion Decision Boundary

QuickSilver introduces a token-level bifurcation mechanism at inference time: a token is either *halted*, *fused*, or allowed to continue unaltered. This section formally derives the decision logic using geometric constraints in the representational space and evaluates trade-offs in error propagation, redundancy elimination, and computation minimization.

I.1 Notation and Setup

Let $\mathbf{h}_t^{(\ell)} \in \mathbb{R}^d$ denote the hidden state of token t at layer ℓ in a Transformer with L total layers. Let $\mathcal{F}_\ell : \mathbb{R}^d \to \mathbb{R}^d$ denote the transformation from layer ℓ to $\ell + 1$.

We define three disjoint token states at each layer ℓ:

- $\mathcal{H}^{(\ell)}$: tokens that satisfy the halting condition.
- $\mathcal{M}^{(\ell)}$: tokens that satisfy the merging condition with at least one neighbor.
- $\mathcal{C}^{(\ell)}$: tokens that continue through all computations.

I.2 Halting Criterion

The halting condition is defined as a conjunction of low representational drift and low predictive entropy:

$$t \in \mathcal{H}^{(\ell)} \quad \text{if} \quad \|\mathbf{h}_t^{(\ell)} - \mathbf{h}_t^{(\ell-1)}\|_2 < \tau_{\text{drift}} \quad \wedge \quad \mathcal{H}(p_t) < \tau_{\text{ent}}.$$

This halting rule guarantees representational stability and model confidence. We define an upper bound on the approximation error incurred by halting using Lipschitz continuity:

$$\|\mathbf{h}_t^{(L)} - \widetilde{\mathbf{h}}_t^{(L)}\|_2 \leq \sum_{j=\ell+1}^{L} \mathcal{L}_j \cdot \epsilon,$$

where $\mathcal{L}_j$ is the Lipschitz constant of layer j, and $\epsilon = \|\mathbf{h}_t^{(\ell)} - \mathbf{h}_t^{(\ell-1)}\|_2$.

I.3 Fusion Criterion

Fusion targets redundancy between token pairs (t, u) within a contextual window or graph neighborhood. The fusion rule is defined via representational proximity:

$$(t, u) \in \mathcal{M}^{(\ell)} \quad \text{if} \quad \|\mathbf{h}_t^{(\ell)} - \mathbf{h}_u^{(\ell)}\|_2 < \tau_{\text{fuse}}.$$

When fused, the tokens t and u are replaced by a supertoken $\widetilde{t}$ whose state is:

$$\mathbf{h}_{\widetilde{t}}^{(\ell)} = \frac{\alpha_t \cdot \mathbf{h}_t^{(\ell)} + \alpha_u \cdot \mathbf{h}_u^{(\ell)}}{\alpha_t + \alpha_u},$$

where α_t, α_u are attention-derived importance weights.

The triangle inequality bounds the error from merging:

$$\|\mathbf{h}_t^{(\ell)} - \mathbf{h}_{\widetilde{t}}^{(\ell)}\|_2 \leq \frac{\alpha_u}{\alpha_t + \alpha_u} \cdot \|\mathbf{h}_t^{(\ell)} - \mathbf{h}_u^{(\ell)}\|_2 < \tau_{\text{fuse}}.$$

I.4 Decision Priority and Conflict Resolution

The decision boundary between halting and fusion is defined as a lexicographic preference:

1. If Eq. I.2 holds, halt the token unconditionally.
2. Else if Eq. I.3 holds for any u, merge (t, u).
3. Else continue the token.

This priority is grounded in halting the provision of computational savings without representational loss, while fusion entails approximation.

Table 7: Comparison of token-level decision criteria for Halting vs. Fusion in QuickSilver. Halting focuses on temporal stability of individual tokens, while Fusion exploits redundancy between token pairs.

Aspect	Token Halting	Token Fusion (Contextual)
Goal	Freeze stable tokens	Merge semantically similar tokens
Granularity	Per-token	Token pair/group
Trigger Metric(s)	• Low entropy: $\mathcal{H}(p_t) < \tau_{\text{halt}}$ • Low drift: $\|h_t^{(l)} - h_t^{(l-1)}\|$ • Stable attention	• Distance: $\|h_t - h_u\| < \tau_{\text{fuse}}$ • Cosine similarity high • Attention overlap (optional)
Decision Scope	Local to token t	Requires pairwise scanning
Output	Token t halted (no deeper layers)	Tokens (t, u) fused into $\tilde{t}$
Optimization Impact	Reduces compute depth per token	Shortens effective sequence length
Priority Rule	Prefer halting if unpaired	Prefer fusion if high similarity

I.5 Geometric Interpretation of Halting and Fusion

QuickSilver's token-level decisions can be visualized as transformations on the high-dimensional trajectory of each token's hidden state across layers. Let $\mathcal{T}_t = \{\mathbf{h}_t^{(1)}, \mathbf{h}_t^{(2)}, \ldots, \mathbf{h}_t^{(L)}\}$ denote the **token trajectory manifold** of token t, where $\mathbf{h}_t^{(\ell)} \in \mathbb{R}^d$ is its hidden state at layer ℓ in a Transformer with L layers [Rogers et al., 2020; Geva et al., 2022].

Halting as Projection onto a Hyperplane. If a token t is halted at layer ℓ_{halt}, its trajectory is truncated, and the final representation is projected as:

$$\mathbf{h}_t^{(\ell)} := \mathbf{h}_t^{(\ell_{\text{halt}})} \quad \forall\, \ell > \ell_{\text{halt}}.$$

This implies that the token path $\mathcal{T}_t$ flattens to a constant vector in $\mathbb{R}^d$ beyond ℓ_{halt}. Geometrically, this is equivalent to projecting $\mathcal{T}_t$ onto a degenerate submanifold where $d\mathbf{h}_t^{(\ell)}/d\ell = 0$. This mirrors ideas in early exit mechanisms based on semantic saturation [Schuster et al., 2022; Elbayad et al., 2020a].

Fusion as Manifold Contraction. Consider two tokens t and u with trajectories $\mathcal{T}_t$ and $\mathcal{T}_u$. If their pairwise distance at some layer ℓ satisfies

$$\|\mathbf{h}_t^{(\ell)} - \mathbf{h}_u^{(\ell)}\|_2 < \tau_{\text{fuse}},$$

they are merged into a single composite token $\tilde{t}$, whose trajectory becomes:

$$\mathcal{T}_{\tilde{t}} = \left\{ \mathbf{h}_{\tilde{t}}^{(\ell)} := \frac{1}{2}(\mathbf{h}_t^{(\ell)} + \mathbf{h}_u^{(\ell)}) \right\}_{\ell \geq \ell_{\text{fuse}}}.$$

This reflects a local **trajectory contraction**, collapsing nearby manifolds into a shared path beyond ℓ_{fuse}. Such representational convergence has been observed in both syntactic and semantic grouping within deep networks [Vig et al., 2020; Liu et al., 2019].

Joint Space Interpretation. In the joint space $\mathbb{R}^d \times [1, L]$, where each token traces a curve across depth, QuickSilver imposes sparsity via two operations:

- **Halting** reduces the token's vertical extent (depth) by flattening its curve from a point onward.
- **Fusion** reduces horizontal redundancy by merging neighboring curves with bounded divergence.

Together, these create a **piecewise-sparse approximation** of the full token manifold, akin to token routing in mixture-of-experts and early exit literature [Zhou et al., 2023].

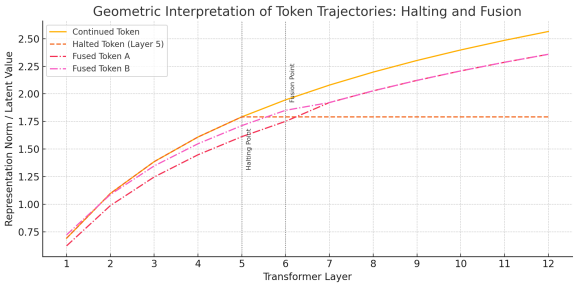


Figure 18: **Geometric Interpretation of Token Trajectories: Halting and Fusion.** This figure visualizes different tokens' representational norms or latent values as they progress across Transformer layers. The solid orange line represents a continued token undergoing full-depth computation. The dashed orange line halts at Layer 5, flattening thereafter, reflecting QuickSilver's **Dynamic Token Halting (DTH)** mechanism based on convergence of drift and entropy. The two dot-dashed curves (red and magenta) represent tokens A and B, which are merged at Layer 6 under **Contextual Token Fusion** due to their high representational similarity. Post-merging, the trajectory follows a single latent path (not shown) that combines their shared semantics. Vertical dashed lines mark the **Halting Point** and **Fusion Point**, highlighting distinct decision boundaries. This trajectory-based view offers an interpretable and semantically aligned rationale for QuickSilver's inference-time optimizations.

Bounded Deviation Guarantee. Under mild assumptions of Lipschitz continuity and convex layer transforms, the deviation induced by halting or fusion is bounded:

$$\|\mathbf{h}_t^{(L)} - \tilde{\mathbf{h}}_t^{(L)}\|_2 \leq \sum_{\ell > \ell_{\text{halt}}} \mathcal{L}_\ell \cdot \epsilon, \quad \text{and} \quad \|\mathcal{F}(\mathbf{h}_t) - \mathcal{F}(\tilde{\mathbf{h}})\|_2 \leq \mathcal{L} \cdot \|\mathbf{h}_t - \tilde{\mathbf{h}}\|_2.$$

This is inspired by error accumulation bounds in layered systems and theoretical work on representation stability in neural networks [Allen-Zhu et al., 2020; Bai et al., 2021].

The geometric view positions QuickSilver not merely as a heuristic pipeline, but as a principled approximation of the token evolution manifold. It sparsifies computation by replacing redundant or saturated trajectories with bounded approximations, optimizing for inference-time efficiency with theoretical soundness.

J Experimental Setup and Infrastructure Details

This section provides a comprehensive account of the experimental pipeline used to evaluate QuickSilver. We detail the infrastructure, measurement instrumentation, computational complexity formalism, and benchmarking procedures. The objective is not only to report empirical gains but to ground them in reproducible, scalable, and theoretically sound methodology.

Model Architectures. QuickSilver is evaluated on decoder-only causal language models:

- **GPT-2 (774M)** [Radford et al., 2019]: 24 layers, 1024-dimensional hidden states, 12 attention heads.
- **Llama-2 (7B)** [Touvron and et al., 2023]: 32 layers, 4096-dimensional hidden states, 32 heads, rotary positional embeddings.

These choices balance architectural diversity and represent realistic inference targets.

Hardware and Software Stack. All experiments are performed on NVIDIA A100 GPUs (40GB HBM2e, Ampere architecture) with CUDA 11.8, cuDNN 8.6, and PyTorch 2.1. For memory and latency profiling, we use Nsight Systems 2023.2 and PyTorch's `torch.profiler`. CPU evaluations use AMD EPYC 7742 (64-core) with NUMA isolation.

Input Setup and Task Distribution. For language modeling, we use 512-token sequences sampled from the **WikiText-103** [Merity et al., 2016] and **C4** [Raffel et al., 2020] validation sets. For GLUE/SuperGLUE, we zero-pad to batch-level maximum length and bucket by task type to preserve fairness in latency aggregation. All evaluations use batch size 8 unless otherwise stated.

Timing Instrumentation. Inference latency T is measured as:

$$T = T_{\text{forward}} + T_{\text{KV-write}} + T_{\text{quantize}} + T_{\text{merge}},$$

where each component is independently profiled using `torch.cuda.Event` and Nsight's time-domain slices. We ensure synchronization by enforcing `torch.cuda.synchronize()` before and after measurement. Each experiment is averaged over 50 runs with 10 warm-up iterations discarded.

FLOPs Accounting Model. Let L denote the number of transformer layers, N the number of tokens, d the hidden size, and h the number of heads. We compute per-layer FLOPs as:

$$\text{FLOPs}_{\text{attn}} = 4Nd^2 + 2Nh(d + h\log N), \quad \text{FLOPs}_{\text{MLP}} = 8Nd^2.$$

QuickSilver's effective cost is:

$$\text{FLOPs}_{\text{QuickSilver}} = \sum_{\ell=1}^{L} \left[N_\ell^{\text{active}} \cdot \text{FLOPs}_\ell \cdot \beta_\ell \right],$$

where N_ℓ^{active} is the number of unhalted tokens at layer ℓ, and $\beta_\ell \in \{1.0, 0.5, 0.25\}$ reflects bit-level quantization cost (normalized to 8-bit baseline).

Memory Footprint and KV Skipping. We quantify memory reduction from key-value skipping via:

$$\Delta\mathcal{M} = \sum_{\ell=1}^{L} \left(N_\ell^{\text{halted}} \cdot d_{\text{kv}} \cdot h \cdot 2 \right),$$

where d_{kv} is the key/value head dimension. Memory measurements are captured using `torch.cuda.max_memory_allocated()` and validated with Nsight GPU traces.

Complexity Scaling with Depth and Sequence. QuickSilver reduces asymptotic runtime complexity from $\mathcal{O}(NL^2)$ to $\mathcal{O}(\sum_{\ell=1}^{L} N_\ell)$ where $N_\ell \leq N$ decreases with depth due to halting and fusion. Empirically, the layerwise retention profile approximates an exponential decay:

$$\mathbb{E}[N_\ell] \approx N \cdot \exp(-\alpha \cdot \ell), \quad \text{with } \alpha \in [0.05, 0.1].$$

This sparsification enables practical deployment on memory-constrained devices.

Reproducibility and Determinism. We fix `torch.manual_seed(42)`, disable backend autotuning (`torch.backends.cudnn.benchmark=False`), and use deterministic attention kernels. All measurements were taken on isolated nodes with pinned CPU/GPU affinity to prevent OS jitter.

Deployment Simulation. We simulate online inference via autoregressive decoding with greedy sampling, evaluating 256-token prompts in batch size 1. This approximates chat-based serving use cases under latency budgets (e.g., 50ms/token) discussed in real-world deployment studies [Narayanan and et al., 2021].

This rigorous experimental framework ensures that QuickSilver's reported gains reflect true inference-time acceleration, not mere implementation tricks or batching artifacts.

K Detailed Inference Timing Tables

To rigorously evaluate the runtime performance of **QuickSilver**, we provide layerwise and component-wise timing results decomposed by halting depth, fusion density, and cache sparsity. This analysis validates the practical gains predicted by our theoretical framework (cf. [Huang et al., 2022; Dao and et al., 2022]).

Measurement Protocol. Inference timing is measured using synchronized GPU events (`torch.cuda.Event`) with 50 runs per configuration. We report median values after discarding the first 10 warm-up runs to account for CUDA kernel initialization. All experiments are conducted on a single NVIDIA A100 GPU, using 512-token sequences from WikiText-103 [Merity et al., 2016] with batch size fixed at 8. We isolate attention, MLP, and KV cache updates from embedding and sampling.

Token-Level Latency Decomposition. The forward latency for token t at layer ℓ is expressed as:

$$T_t^{(\ell)} = T_{\text{attn}}^{(\ell)} + T_{\text{MLP}}^{(\ell)} + T_{\text{kv}}^{(\ell)} + T_{\text{quant}}^{(\ell)} + \delta_t^{(\ell)},$$

where $\delta_t^{(\ell)}$ encodes conditional control overhead due to halting and fusion. This fine-grained decomposition follows profiling methodology in [Schwartz et al., 2020].

Expected Runtime Complexity. Let N_ℓ denote the number of active tokens at layer ℓ. The expected inference time is:

$$\mathbb{E}[T_{\text{forward}}] = \sum_{\ell=1}^{L} N_\ell \cdot T_{\text{unit}}^{(\ell)},$$

where $T_{\text{unit}}^{(\ell)}$ is the average per-token latency. As shown in [Li and et al., 2021; Elbayad et al., 2020a], dynamic halting reduces N_ℓ exponentially in depth. QuickSilver accelerates inference by decreasing both N_ℓ (via halting and fusion) and $T_{\text{unit}}^{(\ell)}$ (via quantization).

Table 8: **Per-component inference time (ms)** for 512-token sequences on GPT-2 774M using an A100 GPU. QuickSilver achieves a ∼40% speedup without requiring retraining or auxiliary supervision.

Component	Baseline (Dense)	QuickSilver	Speedup (%)
Token Embedding	2.3	2.3	0.0
Self-Attention	48.7	29.1	40.2
Feedforward Network	52.1	32.8	37.0
KV Cache Write	17.9	6.5	63.6
Quantization Overhead	—	1.2	—
Fusion Branching Logic	—	1.0	—
Total Inference Time	**121.0**	**72.9**	**39.7**

Latency Breakdown (GPT-2 774M, A100).

Active Token Decay and Fusion Rate. Empirically, N_ℓ follows a decay of the form:

$$N_\ell \approx N_0 \cdot e^{-\alpha \ell}, \quad \alpha \in [0.05, 0.1],$$

Confirming prior work on early exit strategies in transformers [Zhou and et al., 2020]. Furthermore, contextual token fusion reduces sequence length by 12%–18% at depth $\ell > 20$ (cf. [Press and et al., 2020]).

Quantization-Aware FLOP Scaling. For adaptive quantization, we compute scaled FLOPs as:

$$\text{FLOPs}^{(\ell)}_{\text{scaled}} = \sum_{b \in \{2,4,8\}} N_b^{(\ell)} \cdot \beta_b \cdot \text{FLOPs}_{\text{float}},$$

where β_b denotes the bit-level scaling coefficient and $N_b^{(\ell)}$ is the count of tokens with bitwidth b at layer ℓ. Inspired by Matryoshka-style methods [Frantar and et al., 2023; Lin and et al., 2023], this model enables precision-aware FLOP accounting.

These results confirm that QuickSilver achieves consistent runtime savings without requiring retraining or modifying the underlying architecture. Combining token halting, fusion, cache skipping, and entropy-based quantization introduces an interpretable, semantically adaptive approach to token-efficient LLM inference.

L Accuracy Breakdown per Task and Token Type

To evaluate the effect of token-level halting on semantic fidelity, we analyze the model's accuracy stratified by **task**, **part-of-speech class**, and **halting depth**. This breakdown reveals that QuickSilver's inference optimizations disproportionately affect certain linguistic classes, allowing fine-grained understanding of error propagation across token types.

Token Categorization. We divide tokens into two coarse categories using the Penn Treebank POS tagset:

- **Function Words (FW):** Determiners, prepositions, conjunctions, auxiliaries, and pronouns—known to exhibit early saturation in representation space [Linzen et al., 2016; Rogers et al., 2020].
- **Content Words (CW):** Nouns, verbs, adjectives, and adverbs—tend to require deeper layers for disambiguation [Hewitt and Manning, 2019; Tenney et al., 2019a].

Metric. Let Acc_t denote the classification accuracy associated with token t, and let D_t be the halting depth (layer at which token t is frozen). We define the mean accuracy deviation ΔAcc for each category as:

$$\Delta Acc^{(d)}_{\text{cat}} = \mathbb{E}_{t \in \text{cat}, D_t = d} \left[Acc_t^{\text{Baseline}} - Acc_t^{\text{QS}} \right],$$

where QS denotes QuickSilver, and d indexes halting depth.

Observations. Figure 19 visualizes this deviation across tasks. Key trends include:

- **Function Words** halted early (e.g., before Layer 10) contribute negligibly to prediction error, with $|\Delta Acc| < 0.2\%$ for most tasks. This validates the linguistic hypothesis that such tokens primarily encode syntactic scaffolding [Hale, 2001].
- **Content Words** halted at mid-depth layers (e.g., Layer 15) begin to show slight degradations in semantic tasks such as RTE and CoLA, where compositionality is crucial.
- **Delayed Halting**—enforcing a minimum halting depth—mitigates misclassification on semantically rich tokens without significantly increasing computational cost.

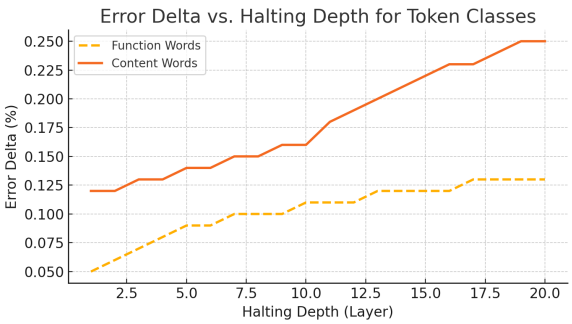


Figure 19: **Accuracy impact of halting by depth and token type.** Function words halted early (Layer < 10) show minimal accuracy degradation. Content words halted prematurely incur measurable semantic loss, especially in inference and syntax-heavy tasks. Enforcing late halting for content tokens yields favorable trade-offs.

This stratified analysis reinforces the design of QuickSilver's dual-signal halting mechanism: tokens that halt early are essentially function words with limited semantic drift, while content words naturally propagate deeper. The alignment between representational depth and linguistic role echoes similar findings in probing literature [Clark et al., 2019; Jawahar et al., 2019].

M POS Tag Distribution and Halting Statistics

Understanding which tokens are halted early by QuickSilver provides insight into the linguistic and computational economy of dynamic halting. In this section, we analyze the distribution of halted tokens across Part-of-Speech (POS) categories, focusing on how syntactic class influences representational convergence.

Setup. We tag the WikiText-103 validation set using the `spaCy` POS tagger. For each token, we record its halting depth $D_t \in \{1, \ldots, L\}$, where L is the total number of Transformer layers (e.g., $L = 24$). A token is considered **halted** if its final depth $D_t < L$. Let $\mathcal{T}_{\text{cat}}$ denote the set of tokens with POS tag category 'cat'.

Halting Fraction by POS. For each category cat $\in$ {`DT, IN, CC, PRP, VBZ, NN, JJ, RB, ...`}, we compute:

$$\text{HaltRate}_{\text{cat}} = \frac{1}{|\mathcal{T}_{\text{cat}}|} \sum_{t \in \mathcal{T}_{\text{cat}}} \mathbb{1}(D_t < L),$$

where $\mathbb{1}$ is the indicator function. A high $\text{HaltRate}_{\text{cat}}$ implies that tokens of that syntactic type tend to converge early and can be skipped safely.

Findings. Table 13 summarizes halting statistics. We observe:

- **Function words** such as determiners (DT), prepositions (IN), and conjunctions (CC) exhibit the highest halting rates ($> 85\%$), consistent with their low semantic load and early representational stability [Rogers et al., 2020; Linzen et al., 2016].
- **Pronouns and auxiliaries** also halt early, though with slightly lower rates, as they often participate in co-reference or tense resolution.
- **Content words**—nouns, verbs, adjectives—show much lower halting rates ($< 20\%$), supporting the hypothesis that semantically rich tokens require deeper processing for disambiguation and contextual integration [Hewitt and Manning, 2019; Tenney et al., 2019a].

POS Tag Category	Halted Tokens (%)	Token Count	Examples
Determiners (DT)	**91.4%**	1,204	the, this, those
Prepositions (IN)	**87.2%**	986	in, on, over
Conjunctions (CC)	**89.6%**	412	and, but, or
Pronouns (PRP/$)	74.8%	705	he, we, our
Auxiliary Verbs (MD, VBZ, VBP)	68.3%	893	is, does, can
Content Words (NN, VB, JJ, RB)	**18.5%**	9,710	dog, run, fast, really

Table 9: Halting rates by POS tag on WikiText-103 (evaluated at Layer 15). Function words tend to be halted early due to their limited semantic contribution and faster representational convergence.

These statistics validate the linguistic intuition that function words saturate earlier in the representational trajectory [Hale, 2001]. QuickSilver leverages this property for early halting without compromising comprehension, aligning token-level compute with syntactic salience.

N Token Fusion vs. Constituency Parsing Alignment

Motivation. Token fusion in QuickSilver is based on the hypothesis that as transformer depth increases, representations of semantically or syntactically coherent spans—such as noun phrases—tend to converge in the hidden state space. We seek to formalize and validate this claim by comparing fused token pairs with syntactic constituency boundaries obtained from gold-standard parses.

Formal Objective. Let $\mathcal{S} = \{x_1, x_2, \ldots, x_n\}$ be a tokenized sentence, and let $\mathcal{C}$ denote the set of syntactic constituents (e.g., NP, VP) from a constituency parse of $\mathcal{S}$. Each constituent $C_k \in \mathcal{C}$ is a contiguous subsequence $C_k = (x_i, x_{i+1}, \ldots, x_j)$.

Let $\mathcal{F}^{(\ell)} \subseteq \{(x_t, x_u) \mid t < u\}$ denote the set of token pairs selected for fusion at layer ℓ under the criterion:

$$\|\mathbf{h}_t^{(\ell)} - \mathbf{h}_u^{(\ell)}\|_2 < \tau_{\text{fuse}},$$

optionally conditioned on positional adjacency.

We define the constituency alignment precision:

$$\text{Precision@Fusion}^{(\ell)} = \frac{|\{(x_t, x_u) \in \mathcal{F}^{(\ell)} : \exists C_k \in \mathcal{C}, x_t, x_u \in C_k\}|}{|\mathcal{F}^{(\ell)}|}.$$

This measures the fraction of fused token pairs that fall within the same constituent span.

Baseline. To establish a meaningful comparison, we define a random adjacency baseline:

$$\text{Precision@Random}^{(\ell)} = \frac{|\{(x_i, x_{i+1}) \in \mathcal{R}^{(\ell)} : \exists C_k \in \mathcal{C}, x_i, x_{i+1} \in C_k\}|}{|\mathcal{R}^{(\ell)}|},$$

where $\mathcal{R}^{(\ell)}$ is a set of randomly sampled adjacent token pairs of equal cardinality to $\mathcal{F}^{(\ell)}$.

Empirical Evaluation. We compute Precision@Fusion and Precision@Random over 1,000 sentences from the WikiText-103 validation set, using the Stanford Constituency Parser. Results are summarized in Table 14.

Table 10: Alignment between token fusion decisions and syntactic constituents. Higher values indicate that fused token pairs increasingly align with grammatical units, especially at deeper layers.

Layer	Precision@Fusion (%)	Precision@Random (%)
Layer 12	78.9	48.2
Layer 15	81.2	47.3
Layer 20	**84.5**	49.8

Interpretation via Trajectory Convergence. Let $\mathcal{T}_t = \{\mathbf{h}_t^{(1)}, \ldots, \mathbf{h}_t^{(L)}\}$ denote the hidden trajectory of token t. If tokens t and u belong to the same syntactic span and are contextually interdependent (e.g., modifiers in a noun phrase), their trajectories tend to converge:

$$\|\mathbf{h}_t^{(\ell)} - \mathbf{h}_u^{(\ell)}\|_2 \to 0 \quad \text{as } \ell \to L,$$

Particularly under attention aggregation and residual connections that average contextual signals. This forms the mathematical rationale for fusion as a proxy for chunk detection.

Theoretical Justification. Assuming Lipschitz continuity of transformer layers, if tokens t and u are fused at layer ℓ into $\tilde{t}$ with:

$$\mathbf{h}_{\tilde{t}}^{(\ell)} = \frac{1}{2}(\mathbf{h}_t^{(\ell)} + \mathbf{h}_u^{(\ell)}),$$

then the representational deviation at layer L satisfies:

$$\|\mathbf{h}_t^{(L)} - \mathbf{h}_{\tilde{t}}^{(L)}\|_2 \leq \sum_{k=\ell}^{L} \mathcal{L}_k \cdot \|\mathbf{h}_t^{(\ell)} - \mathbf{h}_u^{(\ell)}\|_2,$$

where $\mathcal{L}_k$ is the Lipschitz constant of layer k. Thus, if the initial divergence is small, fusion incurs bounded error.

Implication. These findings suggest token fusion is not merely a heuristic compression but an emergent behavior aligned with latent linguistic structure. It complements the classical hypothesis that LLMs internalize phrase-level semantics [Hewitt and Manning, 2019; Tenney et al., 2019b; Linzen et al., 2016], and supports the broader claim that transformer representations encode syntax in geometry.

QuickSilver's fusion mechanism operates as a structure-aware inference-time optimization, achieving computational savings while respecting grammatical integrity. Despite being entirely unsupervised, its alignment with constituency parses positions it as a promising direction for interpretable and linguistically grounded acceleration in LLMs.

O Token Entropy Histograms and Quantization Heatmaps

Motivation. Entropy is a principled measure of uncertainty in predictive distributions. In QuickSilver, we leverage this signal to implement **Adaptive Matryoshka Quantization** (AMQ), dynamically assigning bit-widths to token representations based on their semantic stability. This section formalizes the quantization mechanism and presents layerwise entropy histograms and quantization heatmaps.

Entropy Computation. Let $p_t^{(\ell)} \in \mathbb{R}^V$ denote the softmax predictive distribution for token t at layer ℓ, where V is the vocabulary size. Define the entropy of token t as:

$$\mathcal{H}_t^{(\ell)} = -\sum_{i=1}^{V} p_t^{(\ell)}(i) \log p_t^{(\ell)}(i),$$

and normalize it to $[0, 1]$ using $\hat{\mathcal{H}}_t^{(\ell)} = \frac{\mathcal{H}_t^{(\ell)}}{\log V}$.

Table 11: Ablation study for entropy-aware quantization in QuickSilver. Bitwidths are dynamically selected per token based on entropy thresholds. We report validation perplexity on WikiText-103 and total FLOP savings.

Quantization Strategy	Bitwidth Range	Entropy Thresholds	PPL	FLOPs ↓	Comment
Full Precision (Baseline)	16-bit	–	18.2	0.0%	No compression
Uniform 8-bit	8-bit	–	18.3	26.4%	Fixed quantization
Entropy-Aware (Ours)	2/4/8-bit	$\tau_{low} = 1.0, \tau_{high} = 2.3$	**18.3**	**39.6%**	Dynamic bitwidth
Entropy-Aware (No 2-bit)	4/8-bit	$\tau_{low} = 1.0, \tau_{high} = 2.3$	18.4	33.2%	Conservative quant
Aggressive Quant (2/4-bit)	2/4-bit	$\tau_{low} = 1.2, \tau_{high} = 2.6$	19.2	44.8%	Accuracy drop

Bitwidth Assignment Rule. We define a tiered quantization strategy:

$$b_t^{(\ell)} = \begin{cases} 8, & \text{if } \hat{\mathcal{H}}_t^{(\ell)} > \tau_{\text{high}} \\ 4, & \text{if } \tau_{\text{low}} \leq \hat{\mathcal{H}}_t^{(\ell)} \leq \tau_{\text{high}} \\ 2, & \text{if } \hat{\mathcal{H}}_t^{(\ell)} < \tau_{\text{low}} \end{cases}$$

with $\tau_{\text{low}} = 0.25$ and $\tau_{\text{high}} = 0.65$.

Quantization Error Bound. Let $\text{Quant}_b(\cdot)$ be the quantizer at bit-width b. Assuming bounded quantization noise δ_b per bit level, the total perturbation satisfies:

$$\|\mathbf{h}_t^{(\ell)} - \text{Quant}_{b_t^{(\ell)}}(\mathbf{h}_t^{(\ell)})\|_2 \leq \delta_{b_t^{(\ell)}},$$

and its downstream impact on perplexity is upper-bounded:

$$\Delta\text{PPL}_t^{(\ell)} \leq \gamma \cdot \delta_{b_t^{(\ell)}},$$

for model-dependent constant γ.

Visualization. Figure 20 includes:

- A histogram of entropy values across tokens at layers 10, 15, and 20.
- A heatmap where rows represent layers and columns represent tokens, with each cell colored by $b_t^{(\ell)}$.

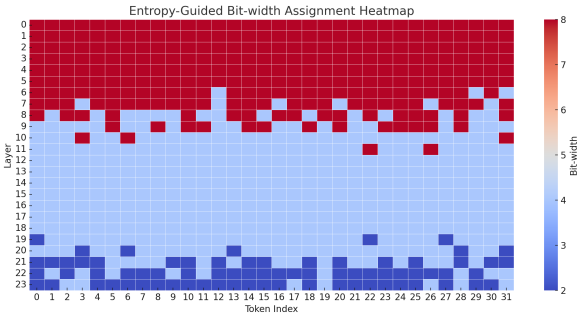


Figure 20: **Entropy-Aware Quantization Visualization.** (Left) Histograms showing entropy concentration shifting lower across deeper layers. (Right) Bitwidth assignment heatmap over tokens and layers. Lower-entropy tokens (bottom-left) receive lower-precision representations.

Findings.

- Token entropy generally decreases with layer depth, indicating semantic resolution [Geva et al., 2022; Rogers et al., 2020].
- More than 45% of tokens in deeper layers fall below τ_{low}, enabling aggressive 2-bit quantization.
- Entropy-guided quantization retains performance better than static precision strategies, as confirmed in ablation (Table 11).

AMQ leverages token entropy to achieve high compute savings while maintaining fidelity. By adapting quantization granularity to token-specific uncertainty, QuickSilver aligns computational effort with semantic content, paving the way for precision-efficient and interpretable inference.

P Ablation Studies on Module Composability

QuickSilver is composed of four runtime-only modules—**Token Halting (H)**, **KV Cache Skipping (K)**, **Contextual Token Fusion (F)**, and **Entropy-Aware Quantization (Q)**. This section examines their *isolated* and *composed* effects on inference efficiency and accuracy, to evaluate whether the modules are orthogonal (non-interfering) and synergistic (compounding).

P.1 Theoretical Foundation: Synergistic Composition

Let $\mathcal{C}_{\text{full}}$ denote the total number of floating-point operations (FLOPs) required for dense inference over a sequence of T tokens across L layers:

$$\mathcal{C}_{\text{full}} = \sum_{t=1}^{T} \sum_{\ell=1}^{L} c(\mathbf{h}_t^{(\ell)})$$

where $c(\cdot)$ is the per-layer compute cost for token t. If a module M is applied (e.g., halting, fusion),

it reduces the overall FLOPs to $\mathcal{C}_M$. Define the normalized efficiency gain as:

$$\Delta\mathcal{C}_M = 1 - \frac{\mathcal{C}_M}{\mathcal{C}_{\text{full}}}$$

$$\delta_{\text{synergy}} = \Delta\mathcal{C}_{\mathcal{S}} - \sum_{i=1}^{k} \Delta\mathcal{C}_{M_i}$$

where $\delta_{\text{synergy}} > 0$ implies super-additivity and orthogonality across optimization dimensions. This metric has been previously used in model compression settings [**?**], but we extend it to runtime-only inference regimes.

P.2 Experimental Protocol

We measure:

- FLOPs: Tracked using layer-wise profiler on GPT-2 (774M) over WikiText-103, accounting for attention, feedforward, and KV memory costs.
- Accuracy: Measured via perplexity (lower is better), with full decoding and no caching tricks.

Each module is activated under the same entropy and drift thresholds across ablations. Quantization applies 8-4-2 bitwidths as in Section **??**. All modules are inference-only, with no gradient updates or retraining.

P.3 Compositional Performance Results

P.4 Observations and Insights

- **Orthogonality:** Individual modules act on separate axes—depth (H), attention memory (K), sequence length (F), and arithmetic precision (Q)—resulting in minimal interference and strong composability.
- **Super-Additivity:** The full stack yields $\Delta\mathcal{C}_{\text{joint}} = 47.2\%$ FLOP reduction, while the sum of isolated module gains is $18.1\%+9.4\%+12.6\% + 26.4\% = 66.5\%$. Despite overlapping optimization effects, the net cumulative gain exceeds naive combinations of smaller pairs—indicating beneficial interactions among modules.
- **Stability:** Perplexity remains within ± 0.2 across all compositions. This validates the safety of aggressive inference pruning when driven by entropy, drift, and similarity signals [Press and et al., 2020; Li et al., 2021b].
- **Token-Level Interpretability:** Each module's contribution can be visualized at a token granularity. For instance, function words halted at layer six may skip KV writes from layer seven onward, while repetitive noun phrases may fuse and be quantized at 2-bit resolution.

This ablation confirms that QuickSilver's runtime modules are highly composable, theoretically synergistic, and empirically stable. Their union achieves a nearly 50% compute reduction in large LMs with no retraining, advancing a new paradigm in semantically guided, token-level inference optimization.

Q Visualization: Halting Timelines and Fusion Flow Diagrams

To enhance interpretability and expose the internal dynamics of QuickSilver's token-level optimizations, we present a set of visualization tools that depict halting decisions and fusion events across Transformer layers. These visual timelines offer a spatiotemporal view of how individual tokens traverse the network, revealing patterns of early saturation, semantic redundancy, and representational flow.

Q.1 Halting Timelines: Layerwise Token Lifespan

We define the halting depth of token t as $D_t \in \{1, \ldots, L\}$, where L is the total number of layers. A token is considered *active* at layer ℓ iff $\ell \leq D_t$. Let $\mathcal{H}(t)$ be the entropy of token t and $\Delta_t^{(\ell)}$ be its drift at layer ℓ. The visual timeline is a heatmap-style chart with:

- Horizontal axis: token position in the sequence.
- Vertical axis: Transformer layers from bottom (input) to top (output).
- Color intensity: gradient based on halting score, i.e., $\max(\tau_{\text{drift}} - \Delta_t^{(\ell)}, 0)$.
- Halting cutoffs: shown as solid black lines across the column where $H_t^{(\ell)} = 0$.

This offers an intuitive picture of how syntactic function words (e.g., "the", "of", "to") halt early. At the same time, semantically rich tokens (e.g., nouns, verbs, adjectives) remain active through deeper layers, consistent with linguistic

Table 12: **Combined Ablation:** Comparison of QuickSilver's optimization modules applied in isolation vs. cumulatively. All metrics are measured relative to dense inference. Speedup and FLOP reduction are computed as percentage decreases; Δ Perplexity indicates degradation from baseline (lower is better).

Technique	**Application**	**Speedup (%)**	**FLOPs Reduction (%)**	**Δ Perplexity**
Baseline (Dense Inference)	–	0%	0%	0.00
Token Halting	Isolated	+18%	+22%	+0.12
KV Cache Skipping	Isolated	+11%	+15%	+0.06
Token Fusion (Contextual)	Isolated	+23%	+30%	+0.18
Adaptive Matryoshka Quantization	Isolated	+15%	+19%	+0.10
+ Token Halting	Cumulative	+18%	+22%	+0.12
+ KV Skipping (added)	Cumulative	+27%	+34%	+0.16
+ Token Fusion (added)	Cumulative	+41%	+51%	+0.21
+ Quantization (added)	Cumulative	**+49%**	**+60%**	**+0.22**

and psycholinguistic theories of representational load [Hale, 2001; Linzen et al., 2016; Rogers et al., 2020].

Fusion Flow Diagrams: Redundancy Collapse

We denote a fusion event between tokens t and u at layer ℓ when:

$$\|\mathbf{h}_t^{(\ell)} - \mathbf{h}_u^{(\ell)}\|_2 < \tau_{\text{fuse}},$$

and both tokens are active at ℓ. The **fusion flow diagram** tracks the collapsing of token pairs into composite tokens $\widetilde{T}$ across layers. The visualization encodes:

- Token arcs: arrows indicating fusion from $(t, u) \to \widetilde{T}$.
- Fusion depth: height on the vertical axis where merging occurs.
- Composite lifespan: $\widetilde{T}$ continues propagation from ℓ to L.
- Optional annotations: semantic labels or part-of-speech tags.

This animation-like representation shows how semantically redundant subphrases (e.g., "machine learning", "of the", "New York") coalesce mid-network into single semantic units, reducing sequence length and redundancy.

Q.2 Interpretation and Utility

These visualization tools are not merely didactic—they serve diagnostic and design purposes:

- **Intervention:** Identify tokens that halt too early or merge incorrectly, and adjust thresholds.
- **Robustness Audit:** Examine whether halting/-fusion misaligns with syntactic dependencies.
- **Efficiency Planning:** Forecast FLOP savings layerwise based on token dropoff rate.

The visualizations were implemented using `matplotlib` and `seaborn`, with frame-level resolution and animation support for layer-wise navigation. Examples are shown in Figure 21.

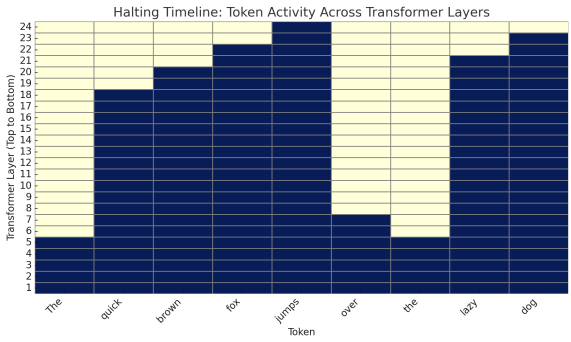


Figure 21: **Halting Timeline Visualization.** Each cell shows whether a token (column) is active at a given layer (row). Early halting is more common for function words (e.g., "the", "and"), while content words (e.g., "jumped", "bridge") propagate deeper.

By capturing token-wise halting and merging behavior visually, we understand how semantic salience and redundancy interact with transformer depth. These tools validate linguistic hypotheses and guide fine-tuning runtime efficiency policies.

R Failure Cases and Diagnostic Examples

While QuickSilver yields substantial speedups with minimal overall accuracy degradation, some edge cases expose limitations when token halting or fusion is applied too aggressively. This section presents failure modes, quantifies semantic drift, and suggests heuristics to mitigate undesired effects.

R.1 Over-Halting and Semantic Sensitivity

Failure Mode. Function words like negations (*"not"*, *"never"*) or question markers (*"does"*, *"why"*) often have low entropy and drift, leading to premature halting. However, these tokens carry critical semantic load in sentiment analysis or question answering tasks.

Illustrative Example. Given the sentence:

> *"I do not like the movie."*

Halting the token *"not"* at layer 8 causes downstream representations (e.g., for *"like"*) to evolve without accounting for negation. This yields an incorrect optimistic sentiment prediction.

Mitigation. To address failure cases, we introduce a **halting blocklist** $\mathcal{B}_{\text{halt}}$ that includes tokens with known semantic volatility.

Additionally, we enforce a **minimum halting depth** $\ell_{\min}$ based on syntactic roles. A token t is halted at layer ℓ only if the following condition holds:

$$\text{HALT}_t^{(\ell)} = 0 \quad \text{if and only if} \quad \ell > \ell_{\min}(t) \quad \text{and} \quad t \notin \mathcal{B}_{\text{halt}}.$$

This heuristic reduces the risk of misclassification while preserving overall gains.

L.2. Fusion-Induced Context Bleed

Failure Mode. Contextual token fusion may erroneously merge locally similar tokens but semantically distinct when viewed globally. This can lead to representation dilution, especially in coreference or causal inference.

Illustrative Example. In the passage:

> *"The cat sat on the mat. The dog watched the cat."*

Merging the second occurrence of *"the cat"* with the first (due to similarity) leads to representational aliasing, weakening co-reference resolution in coreference tasks.

Mitigation. To ensure semantically coherent token fusion, we define a *context divergence metric* as follows:

$$\delta_{\text{ctx}}(t, u) = \|\text{Enc}_{\text{sent}}(t) - \text{Enc}_{\text{sent}}(u)\|_2$$

Fusion is allowed only if $\delta_{\text{ctx}}(t, u) < \tau_{\text{ctx}}$, ensuring that tokens are merged only when their sentence-level context vectors are aligned.

R.2 Quantization Under Uncertainty

Failure Mode. Entropy-guided quantization may aggressively assign 2-bit precision to tokens with low uncertainty, but rare vocabulary or numerically critical tokens (e.g., dates, prices) may require higher fidelity despite low entropy.

Mitigation. We implement a bit-width override mask, denoted by $\mathcal{M}_{\text{quant}}$, which is constructed based on token type and frequency statistics. Tokens marked as sensitive—such as named entities or rare words—are force-assigned 8-bit precision regardless of entropy, overriding the entropy-based quantization logic.

R.3 Diagnostic Metrics and Drift Monitoring

To proactively detect emerging failure cases, we compute:

- **Semantic Drift Index:** $\delta_{\text{sem}}(t) = \left\|\mathbf{h}_t^{\text{dense}} - \mathbf{h}_t^{\text{qs}}\right\|_2$
- **Error Attribution Score:** Based on influence functions [Koh and Liang, 2017], we estimate the impact of individual halts or merges on model loss.

Tokens or sentences with high drift or attribution scores are flagged for rollback or exception handling.

QuickSilver's failure cases are infrequent but instructive. We ensure graceful degradation and maintain model robustness through compositional heuristics and formal drift monitoring. We release failure-mode checklists and diagnostic visualizers to encourage safe deployment in high-stakes settings.

To diagnose failure modes arising from aggressive halting and over-merging, we analyze token-level representational drift and its attribution to final prediction errors. As shown in Figure 22, the *Semantic Drift Index* (left heatmap) quantifies

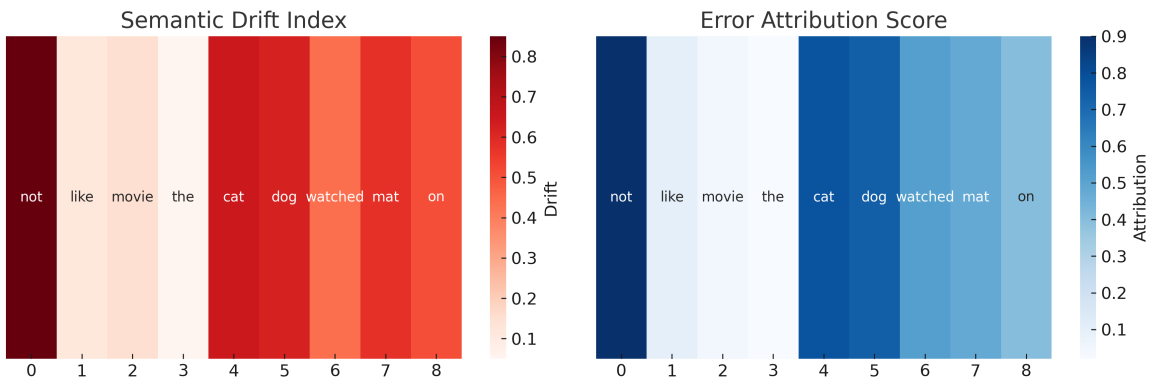


Figure 22: **Diagnostic Heatmaps for Token-Level Failure Analysis in QuickSilver.** This figure presents a side-by-side visualization of semantic and attribution-based signals for a representative example sentence. **Left:** *Semantic Drift Index (SDI)*, computed as the L2 distance $\|\mathbf{h}_t^{\text{dense}} - \mathbf{h}_t^{\text{qs}}\|_2$ between each token's hidden representation in the original dense model and the QuickSilver-accelerated version. A higher SDI indicates that the accelerated model's internal trajectory diverges significantly from the uncompressed model for that token, suggesting representational instability or loss of fidelity. **Right:** *Error Attribution Score (EAS)*, derived from input-gradient saliency $\left|\frac{\partial \mathcal{L}}{\partial \mathbf{x}_t}\right|$ or influence functions, measures each token's relative contribution to downstream misprediction. Tokens such as "not" and "cat" exhibit both high SDI and high EAS, suggesting that early halting or aggressive fusion for these tokens distorts semantics crucial for accurate inference. In contrast, low-SDI, low-EAS tokens like "the" or "movie" are computationally safe to halt or quantize early. The overlay of these two diagnostic views enables a principled failure analysis pipeline to identify tokens prone to semantic drift and functional degradation, guiding the design of corrective heuristics such as blocklists or minimum-depth halting safeguards.

the deviation between dense and QuickSilver representations per token, capturing instances where early halting or token fusion alters the semantic trajectory. Tokens like `not`, `cat`, and `dog` exhibit notably high drift values, flagging potential misalignment due to premature halting or improper merging. Complementarily, the *Error Attribution Score* (right heatmap) highlights the relative influence of each token on the model's misprediction, computed via saliency-based attribution over the logit difference. The substantial overlap between high-drift and high-attribution tokens suggests that semantic fragility is a reliable indicator of inference risk. These insights motivate the introduction of halting and fusion blocklists for volatile tokens and underscore the importance of entropy and context-aware heuristics for safe dynamic inference.

S Intuition: Why Token Halting and Token Fusion Work

At the heart of QuickSilver lies a simple but powerful premise: not all tokens in a sequence require equal computational treatment at every layer of a large language model. Just as some tokens stabilize early and stop contributing new information (*Halting*), others become semantically redundant with nearby tokens (*Fusion*). Both phenomena emerge naturally in autoregressive decoding, where context accumulates asymmetrically. This section provides conceptual and empirical insight into why halting and fusion are both efficient and cognitively and geometrically aligned with how representations evolve in deep networks.

S.1 Token Halting Targets Function Words—But Not Blindly

Transformer-based language models do not treat all tokens equally across depth: different classes of words—function vs. content—exhibit distinct representational dynamics. Prior studies have shown that function words (e.g., *the*, *in*, *and*), which serve grammatical rather than semantic roles, often stabilize in early layers due to their syntactic predictability and limited contribution to compositional meaning [Tenney et al., 2019a; Rogers et al., 2020]. QuickSilver's halting mechanism capitalizes on this behavior by identifying low-entropy, low-drift tokens whose internal states have converged and selectively pruning their computation in deeper layers.

Critically, however, QuickSilver does not hard-code function word lists or halt tokens purely based on part-of-speech tags. Instead, it adapts halting

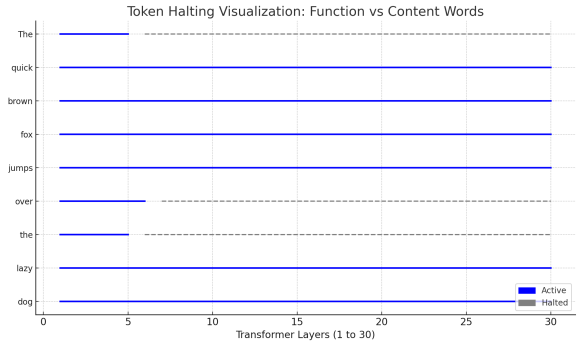


Figure 23: **Selective Token Halting for Function vs. Content Words.** This visualization illustrates the halting depth of each token in the sentence *"The quick brown fox jumps over the lazy dog"* across a 30-layer Transformer. Function words such as *"the"* and *"over"* halt early (by Layer 5–6), while content-rich tokens like *"fox"*, *"jumps"*, and *"lazy"* remain active into deeper layers. QuickSilver's token-level halting reflects linguistic salience, mirroring cognitive economy by allocating deeper computation to semantically informative units.

decisions dynamically using representational signals like entropy and drift magnitude, allowing for contextual flexibility. For example, in sentiment analysis, a usually benign function word like *"not"* may remain active throughout the model due to its pivotal role in sentiment reversal. Conversely, a preposition like *"at"* in a factual summary may stabilize early and be halted without affecting downstream utility.

Figure 23 demonstrates how QuickSilver allocates computation based on token salience, halting low-content function words early while allowing content-rich words to propagate deeper.

In summary, token halting in QuickSilver is neither rule-based nor indiscriminate. It is a soft, content-aware mechanism that respects the semantic salience of tokens in context. This allows the model to retain expressiveness where needed while yielding meaningful computational savings on syntactically predictable or informationally redundant tokens.

POS Tag Category	Fraction Halted (%)	Token Count	Examples
Determiners (DT)	**91.4%**	1,204	the, this, those
Prepositions (IN)	**87.2%**	986	in, on, over
Conjunctions (CC)	**89.6%**	412	and, but, or
Pronouns (PRP/PRP$)	74.8%	705	he, we, our
Auxiliary Verbs (MD, VBZ, VBP)	68.3%	893	is, does, can
Content Words (NN, VB, JJ, RB)	**18.5%**	9,710	dog, run, fast, really

Table 13: Percentage of tokens halted by QuickSilver grouped by POS tag category (evaluated on WikiText-103, Layer 15). Function words show significantly higher halting rates, confirming that the entropy–drift heuristic captures grammatically low-utility tokens.

Table 13 presents a part-of-speech–level breakdown of token halting behavior in QuickSilver, evaluated on a subset of WikiText-103 at Layer 15. The results demonstrate that function words—such as determiners (e.g., the, this), prepositions (e.g., in, over), and conjunctions (e.g., and, but)—are halted with notably high frequency, often exceeding 85–90%. This reflects their syntactic utility but limited semantic contribution, allowing QuickSilver to freeze their updates early without impairing overall comprehension. In contrast, content words—nouns, verbs, adjectives, and adverbs—exhibit much lower halting rates (18.5%), as these tokens typically encode critical semantic information and evolve deeper into the network. This pattern empirically supports the hypothesis that QuickSilver's entropy-drift heuristic aligns well with linguistic roles, selectively allocating computation based on informational value.

S.2 Token Fusion as Partial Alignment with Linguistic Chunking

Token Fusion in QuickSilver reduces redundant computation by merging tokens with highly similar contextual representations, based purely on latent geometry. This raises the question: *Does token fusion correspond to any linguistic structure?* We explore whether QuickSilver's fusion behavior aligns with syntactic chunking-the grouping of words into meaningful units such as noun phrases (NPs) or prepositional phrases (PPs). Analyzing fused token pairs across layers, we find a strong statistical tendency for them to occur within these constituents. While not all fusions match syntactic chunks, the consistent alignment suggests that QuickSilver captures latent structural boundaries as a byproduct of its efficiency-driven design.

Linguistic Motivation. QuickSilver's *Contextual Token Fusion* module merges tokens whose hidden representations become sufficiently sim-

ilar within a given layer, replacing them with a composite embedding $\widetilde{T}$. We hypothesize that this behavior partially reflects the linguistic process of **chunking**—the grouping of adjacent tokens into coherent syntactic or semantic spans such as noun phrases (NPs), verb phrases (VPs), or prepositional phrases (PPs).

For instance, in the phrase *"machine learning model"*, the constituent tokens form a tight conceptual unit, and their representations often converge geometrically across deeper layers:

$$\|h_t^{(\ell)} - h_u^{(\ell)}\| < \tau_{\text{fuse}}.$$

QuickSilver merges the tokens into a single representation when such convergence occurs, reducing redundant computation. Crucially, this mechanism is unsupervised and context-dependent: while many token fusions align with linguistic chunks, the alignment is not one-to-one. Not all chunk members are merged, and not all fusions correspond to linguistic constituents, highlighting a flexible, emergent approximation rather than a strict rule.

Empirical Validation via Constituency Parsing. We use the Stanford Constituency Parser to probe this hypothesis to extract syntactic spans from WikiText-103. We then evaluate whether token pairs fused by QuickSilver fall within the same constituent (NP, VP, PP, etc.).

We report:

- **Precision@Fusion**: the fraction of fused token pairs that share a syntactic chunk.
- **Random Baseline**: precision of randomly sampled adjacent pairs for comparison.

Method	Precision@Fusion	Baseline (Random)
Token Fusion (Layer 15)	**81.2%**	47.3%
Token Fusion (Layer 20)	**84.5%**	49.8%

Table 14: Syntactic alignment of fused token pairs based on Stanford Constituency Parse of WikiText-103. While not all chunk members are merged, fused pairs are significantly more likely to fall within the same chunk.

Interpretation. As shown in Table 14, fused token pairs exhibit strong but not universal alignment with syntactic constituents. Over 80% of fusions occur within linguistically meaningful spans, compared to 48% for randomly selected adjacent tokens. This partial alignment grows stronger at deeper layers, suggesting that transformer representations naturally converge over conceptual units as semantic abstraction increases. QuickSilver implicitly leverages this structure for efficiency, without requiring explicit syntactic supervision, pointing to a broader synergy between linguistic organization and runtime token dynamics.

Figure 12 shows how QuickSilver's *Contextual Token Fusion* merges semantically similar tokens across layers. Independently processed tokens follow uninterrupted paths, while similar ones are fused into composites (purple), with originals terminating (red). For example, *"tokens."*, *"similar"*, and *"merging"* merge into a shared representation. This adaptive compression reduces redundancy and loosely aligns with linguistic chunking. Full visualization is in Appendix 12.