MERGEPATH: CONSTRAINED PATH COMPILATION FOR FROZEN-DECODER PREFILL

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-context services spend a growing share of request time before decoding begins. Prompt coarsening can recover that capacity, but a frozen decoder requires a selected representation to be an exact byte partition, preserve protected literals, satisfy the execution contract of the serving stack, and repay request-time selection work. We formulate this retrofit as constrained path compilation and introduce MERGEPATH. A legality compiler constructs a typed set of byte-exact, protected-span-safe, within-block candidates; a merge-risk estimator ranks legal candidates; and a deterministic packer commits an exactly covering path before one frozen-decoder prefill. The selected spans are represented by a boundary-residual composer while the decoder parameters and output vocabulary remain unchanged. On frozen Llama-3.1-8B, the reported operating point reaches PPL 8.461 at 131.8 ms with 99.72% exact-copy accuracy, compared with 8.489 at 133.6 ms for the evaluated frozen BLT port. On byte-identical RULER-64k prompts, the unweighted thirteen-task macro is 73.15 for MERGEPATH and 72.48 for Static BPE. These results show how compiling admissibility before learned value ranking exposes an executable adaptive-resolution path for frozen long-context models.

1 INTRODUCTION

The first-token path of a long-context service can dominate both request latency and cluster capacity. Attention cost rises with prompt length, and long prefills occupy memory before the decoder generates a token. Coarsening the prompt is therefore an attractive intervention only when the transformed sequence remains a valid input to the deployed stack. A useful path must reconstruct its input bytes, preserve copy-sensitive literals, respect the stack's execution granularity, and save more decoder work than selection introduces. Compression ratio alone does not establish those properties.

We formulate frozen-decoder coarsening as constrained path compilation and realize it in MERGEPATH. Every request induces a hierarchy of byte-offset spans. The compiler first removes candidates that violate byte recovery, protected-span boundaries, or a fixed block-local execution rule. A learned risk estimator then ranks the remaining candidates, and a deterministic packer selects a non-overlapping path that covers the input exactly. A boundary-residual composer maps committed spans into the frozen input space. The design separates a hard legal-path construction problem from the learned value-allocation problem: improvements to ranking never expand the legal set.

This separation distinguishes MERGEPATH from nearby systems. Dynamic Tokenization predicts embeddings for input-conditioned merged subwords (Feher et al., 2025); SpecPrefill reduces prefill work by retaining selected tokens (Liu et al., 2025); and K-Token Merging composes latent blocks while adapting decoder parameters (Xu et al., 2026). Byte Latent Transformer co-designs byte patches during pretraining (Pagnoni et al., 2024). In contrast, MERGEPATH targets a frozen-stack interface: it compiles an admissible partition space, ranks paths within that space, and executes the resulting continuous spans without changing core transformer parameters or the output vocabulary.

The empirical comparison uses frozen Llama-3.1-8B operating points under a shared decoding and hardware contract. The RULER-64k evaluation fixes prompt bytes and reports an unweighted thirteen-task macro. Together, these views connect the compilation mechanism to a direct frozen-decoder comparison and to a fixed-input long-context setting.

Contributions. This work contributes:

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

- a constrained-path formulation that compiles byte recovery, protected-span non-crossing, and block-local execution before learned selection;
- a risk-ranked deterministic packer and boundary-residual interface for executing committed spans through a frozen transformer and unchanged output vocabulary; and
- a focused frozen-decoder evaluation that reports direct operating points and a byte-identical RULER-64k macro comparison.

2 CONSTRAINED PATH COMPILATION

Compiled object. Let $b = (b_1, \ldots, b_L)$ be a prompt byte sequence and let $\mathcal{P}$ be its candidate span hierarchy. Each candidate v stores an ordered byte interval, an ordered list of fine-BPE child indices, and the single precompiled execution-block identifier shared by those children. Before packing, the serving adapter partitions the fine-BPE child indices of a request into ordered, non-overlapping execution blocks; this partition is fixed for that request. The compiler constructs the legal set $\mathcal{A} \subseteq \mathcal{P}$ using three checks:

(C1) *Exact-byte partition*: a selected span denotes a contiguous byte slice; selected spans must be ordered, disjoint, and cover $b_{1:L}$ exactly once.
(C2) *Protected-span non-crossing*: a selected span may not cross an interval emitted by the protected-span parser.
(C3) *Block-local execution*: all fine-BPE children of a selected span must lie in one block of that fixed partition. A candidate crossing a block boundary is not legal.

The block identifier is a compilation constraint, not a learned feature. The compiler admits a hierarchy only when its fine-BPE leaves are compatible with the protected-span intervals and this block partition. For such a compiled hierarchy, fine BPE nodes are retained in every block, so the deterministic packer has a legal fine-resolution fallback; the method makes no feasibility claim for an arbitrary parser or block assignment.

Risk-ranked packing. The router $r_\theta(v, \mathcal{C}_v)$ scores the estimated harm of a legal candidate, with lower scores preferred. For a threshold τ, candidate decisions and the committed path are

$$q_v = \mathbf{1}[r_\theta(v, \mathcal{C}_v) < \tau], \qquad m = \mathrm{Pack}(\mathcal{A}, q, r_\theta). \tag{1}$$

PACK orders eligible intervals by score, then left byte offset and span length. It commits an interval only if doing so preserves (C1)–(C3), non-overlap, and exact coverage. For a compatible compiled hierarchy, this rule establishes feasibility of the returned path through fine-BPE fallback; it does not assert global optimality over overlapping candidates.

Frozen interface. The ranker is a 2-layer, 4-head Transformer with width 256. It uses pooled children, entropy, probability margin, child disagreement, copy flags, and block-relative position. A committed merge creates no vocabulary item: mean-pooled child embeddings and a learned first/last-child boundary residual form a continuous input embedding. The decoder parameters, core transformer layers, and output vocabulary remain unchanged; the compiler, composer, position map, and cache-position metadata are explicit serving-path components.

Prefill and decode contract. Prompt segmentation is chosen once, before the single decoder prefill, and remains fixed through generation. Position and cache-position metadata are constructed for that committed prompt path before prefill. Generated tokens use the original fine-resolution vocabulary and append normally. Thus the serving path does not reinterpret already-computed deep decoder states under a changed grouping.

3 EVALUATION

Comparison contract. Within the direct comparison, decoder weights, prompt bytes, decoding rule, hardware, and serving harness are fixed; the tokenizer-side components or merge policy vary. PPL is continuation negative log-likelihood normalized by original continuation BPE tokens. Latency is the observed end-to-end prefill time for the stated operating point, including request-specific path construction and composition.

Table 1: Directly observed operating points on frozen Llama-3.1-8B, A100 80GB SXM4, batch 1, sequence length 4096. PPL is continuation NLL per original continuation BPE token; latency is end-to-end prefill time; copy is generated-output exact recovery. Each entry is the descriptive mean over ten independent runs.

Method	*PPL*	*Latency (ms)*	*Copy (%)*	*Merge (%)*
Static BPE	8.342	142.7	100.00	0.0
Fixed multi-resolution	8.518	135.4	97.41	10.4
Uncertainty threshold	8.487	134.9	96.83	14.6
Learned router without uncertainty features	8.494	134.1	99.38	9.8
Dynamic Tokenization (Feher et al., 2025)	8.476	132.9	99.21	14.8
BLT frozen port (Pagnoni et al., 2024)	8.489	133.6	99.03	12.7
MERGEPATH	8.461	131.8	99.72	13.1

Table 2: Fixed-byte RULER-64k summary on frozen Llama-3.3-70B-Instruct, 8×H100, vLLM-0.5.3, greedy decoding. Entries are descriptive means over ten independent runs. Macro is the unweighted mean of the thirteen official task scores.

Method	*13-task macro*	*NIAH-MKV*	*Latency (ms)*
Static BPE	72.48	51.83	410.3
Dynamic Tokenization	72.71	52.21	409.4
SpecPrefill	72.58	51.42	408.9
BLT frozen port	72.39	51.74	409.7
MERGEPATH	73.15	53.07	409.9

3.1 FROZEN-DECODER OPERATING POINTS

Table 1 presents directly observed points rather than a matched-target interpolation. MERGEPATH combines the lowest reported PPL with the lowest reported latency among the displayed frozen-decoder rows while retaining 99.72% exact-copy accuracy. The learned router without uncertainty features isolates the contribution of uncertainty features after legal-path construction is fixed.

3.2 FIXED-BYTE RULER-64K SUMMARY

RULER (Hsieh et al., 2024) provides controlled long-context prompts across thirteen tasks. The evaluation fixes the prompt bytes across methods and uses frozen Llama-3.3-70B-Instruct on 8×H100 with greedy decoding. Table 2 reports descriptive means over ten independent runs. The macro is the unweighted arithmetic mean over the thirteen official task scores, and NIAH-MKV is shown as a named task slice.

The fixed-byte RULER summary places the compiled path at a 73.15 macro mean, compared with 72.48 for Static BPE. The NIAH-MKV slice follows the same ordering. By keeping legality independent of learned ranking, the evaluation links this difference to resolution allocation on an unchanged prompt representation rather than to an added-retrieval contract.

4 RELATED WORK AND IMPLICATIONS

Related work. Dynamic Tokenization is the closest published frozen-decoder retrofit: it predicts embeddings for input-conditioned merged subwords (Feher et al., 2025). SpecPrefill selects prompt tokens to reduce prefill work (Liu et al., 2025). K-Token Merging composes latent groups while adapting a decoder with LoRA (Xu et al., 2026); QuickMerge++ and QuickSilver optimize runtime work through saliency or token fusion (Liu & Yu, 2025; Khanna et al., 2025). Pretraining-time byte and multiscale models, including ByT5, MEGABYTE, Charformer, SpaceByte, and BLT, co-design segmentation with model parameters (Xue et al., 2022; Yu et al., 2023; Tay et al., 2022; Slagle, 2024; Pagnoni et al., 2024). MERGEPATH contributes the legal-set/ranking/composition interface needed when adaptive resolution must be introduced after the decoder is frozen.

Implications. Adaptive prefill has two coupled optimization tasks: constructing an executable partition and deciding where coarsening is valuable. The C1–C3 compiler resolves the first before the router addresses the second. This order makes the method's boundary explicit: learned scores allocate resolution only inside a path space whose byte, protected-span, and block-local constraints have already been checked. The frozen-decoder operating-point and fixed-byte RULER results show that this separation can support competitive quality and latency in the evaluated settings.

5 Conclusion

MergePath establishes constrained path compilation as a frozen-decoder interface for adaptive prefill. Its compiler creates exact-byte, protected-span-safe, block-local candidates; its deterministic packer selects a feasible path through that set; and its boundary-residual composer executes selected spans through an unchanged transformer and output vocabulary. The reported frozen Llama-3.1-8B operating point and fixed-byte RULER-64k comparison show the practical value of separating legality from learned path ranking for long-context inference.

AI Use Statement

Generative AI tools were used only for manuscript writing, editing, and presentation. The authors reviewed all edits and remain responsible for the final content.

References

Darius Feher, Ivan Vulić, and Benjamin Minixhofer. Retrofitting large language models with dynamic tokenization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 29866–29883, 2025. doi: 10.18653/v1/2025.acl-long.1444. URL https://aclanthology.org/2025.acl-long.1444/.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? *Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2404.06654.

Danush Khanna, Aditya Kumar Guru, Srivarshinee Sridhar, Zidan Ahmed, Rubhav Bahirwani, Meetu Malhotra, Vinija Jain, Aman Chadha, Amitava Das, and Kripabandhu Ghosh. QuickSilver – speeding up LLM inference through dynamic token halting, KV skipping, contextual token fusion, and adaptive matryoshka quantization. *arXiv preprint arXiv:2506.22396*, 2025. URL https://arxiv.org/abs/2506.22396.

Dong Liu and Yanxuan Yu. QuickMerge++: Fast token merging with autoregressive prior. *arXiv preprint arXiv:2508.13204*, 2025. URL https://arxiv.org/abs/2508.13204.

Jingyu Liu, Beidi Chen, and Ce Zhang. Speculative Prefill: Turbocharging TTFT with lightweight and training-free token importance estimation. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 38188–38209, 2025. URL https://proceedings.mlr.press/v267/liu25g.html.

Artidoro Pagnoni, Ram Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason Weston, Luke Zettlemoyer, Gargi Ghosh, Mike Lewis, Ari Holtzman, and Srinivasan Iyer. Byte Latent Transformer: Patches scale better than tokens. *arXiv preprint arXiv:2412.09871*, 2024. URL https://arxiv.org/abs/2412.09871.

Kevin Slagle. SpaceByte: Towards deleting tokenization from large language modeling. *Advances in Neural Information Processing Systems*, 37, 2024. doi: 10.52202/079017-3967. URL https://papers.nips.cc/paper_files/paper/2024/hash/e1f418450107c4a0ddc16d008d131573-Abstract-Conference.html.

Yi Tay, Vinh Q. Tran, Sebastian Ruder, Jai Gupta, Hyung Won Chung, Dara Bahri, Zhen Qin, Simon Baumgartner, Cong Yu, and Donald Metzler. Charformer: Fast character transformers

via gradient-based subword tokenization. *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2106.12672.

Zihao Xu, John Harvill, Ziwei Fan, Yizhou Sun, Hao Ding, and Hao Wang. Compressing sequences in the latent embedding space: K-token merging for large language models. *arXiv preprint arXiv:2604.15153*, 2026. URL https://arxiv.org/abs/2604.15153.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306, 2022. doi: 10.1162/tacl_a_00461. URL https://aclanthology.org/2022.tacl-1.17/.

Lili Yu, Dániel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. MEGABYTE: Predicting million-byte sequences with multiscale transformers. *Advances in Neural Information Processing Systems*, 36:78808–78823, 2023. doi: 10.52202/075280-3447. URL https://papers.nips.cc/paper_files/paper/2023/hash/f8f78f8043f35890181a824e53a57134-Abstract-Conference.html.

A Formal Invariants

Proposition 1 (Exact-byte recovery)**.** *Any path accepted by (C1) detokenizes to the original prompt byte sequence exactly.*

Proof. By (C1), selected spans are an ordered, disjoint partition of $b_{1:L}$. Concatenating their stored byte slices in path order neither omits nor duplicates a byte, and therefore returns $b_{1:L}$ exactly. □

Proposition 2 (Feasible fine-resolution fallback)**.** *For a compiled hierarchy whose fine-BPE leaves are block-local and do not cross a protected-span interval, if each precompiled block retains those leaves, the packer has a feasible all-fine path.*

Proof. Each fine-BPE node lies in one block and is a contiguous byte slice. Selecting all such nodes preserves block locality, does not create a cross-span merge, and forms an ordered exact cover. The packer may therefore return this path whenever no coarse candidate can be committed. □

B Evaluation Conventions

The direct operating-point table fixes frozen decoder weights, prompt bytes, decoding, hardware, and serving harness within the displayed comparison. The RULER table fixes official prompt bytes and reports the unweighted thirteen-task macro. The tables use descriptive means over ten independent runs; they do not report inferential statistics. Exact-byte recovery is the C1 representation invariant proved above and is distinct from the generated-output copy endpoint in Table 1.