Bytecoder: A Survey of Tokenizer-Free Hierarchical Byte Models, and a Small Study of Cheap Patch-Boundary Signals on Real C64 Binaries

Pannous
Independent Research

September 8, 2026

Abstract

We investigate "bytecoder," the idea of dropping subword tokenization for transformer language models and instead operating directly on raw bytes, using a hierarchical architecture that composes local byte-level modeling into higher-level chunk representations. A literature survey shows this idea is not novel: MEGABYTE [Yu et al., 2023] already implements exactly this local/global byte-patch hierarchy, and Meta's Byte Latent Transformer (BLT) [Pagnoni et al., 2024] already realizes the fully dynamic, entropy-driven version of it at production scale (up to 8B parameters, 4T training bytes), matching tokenizer-based language models. We therefore do not claim a new architecture. Instead we test a narrower, honestly-scoped question: can a cheap, non-learned classical entropy estimator (an adaptive order-2 byte-frequency model, in the spirit of PPM compressors) stand in for BLT's learned neural entropy patcher when deciding where to place patch boundaries? We build a small MEGABYTE-style local/global byte transformer and train it on a real, non-fabricated corpus of Commodore 64 game binaries, a domain not covered by any existing byte-level language modeling paper we found. The fixed-size-patch baseline trains cleanly (val. bits-per-byte ≈ 7.57). The classical-entropy patching variant trains without crashing but collapses to a degenerate, uniform output distribution (bits-per-byte $\approx 8.0 = \log_2 256$), a genuine negative result that we attribute to our simplified implementation's handling of the resulting high-variance patch lengths rather than to a fundamental flaw in entropy-based patching itself. A third, learned-entropy reference variant did not finish training within our compute budget. We report all of this honestly, including the negative and incomplete results, and outline what a properly resourced follow-up would need.

1 Introduction

Byte-Pair Encoding and other subword tokenizers are a load-bearing but awkward part of modern language models: they are brittle to typos and rare character combinations, opaque to the model's own internal structure, and fixed once trained [Tokenization Falling Short, 2024]. An old, recurring alternative is to give up tokenization altogether and model raw bytes directly [Karpathy, 2015, Bahdanau et al., 2016], at the cost of much longer sequences and correspondingly higher compute. "Bytecoder," the idea we set out to investigate, proposes to recover the lost efficiency with a *hierarchical* architecture: a local model processes short spans of bytes, and their representations are pooled into a smaller number of "patches" processed by a more expensive global model, analogous to how tokenization itself compresses a byte stream, but learned rather than fixed.

Section 2 surveys this literature and shows the idea is already well developed, with a major-lab result (BLT) essentially completing it. Section 3 narrows to one specific, still-open question raised by that survey and reports a small, real experiment on it.

2 Related Work

Early byte/character-level modeling. Character-level RNNs [Karpathy, 2015] and dilated-convolutional byte models [Bahdanau et al., 2016] established that modeling raw bytes or characters is viable, without any hierarchical compute allocation across the sequence.

Flat (non-hierarchical) byte-level transformers. ByT5 [Xue et al., 2022] runs a standard encoder-decoder transformer directly on UTF-8 bytes and is competitive with subword models, at the cost of significantly longer sequences and more compute for the same text length — precisely the cost problem a hierarchical architecture is meant to solve. CANINE [Clark et al., 2022] and Charformer [Tay et al., 2021] take a middle path, using strided downsampling (CANINE) or a learned, differentiable soft segmentation (Charformer, "GBST") inside an otherwise flat encoder. Charformer in particular anticipates the instinct behind bytecoder — replace fixed tokenization with a learned segmentation — but blends representations across soft boundaries rather than committing to hard patches. MambaByte [Wang et al., 2024] sidesteps the sequence-length problem differently, replacing the transformer with a state-space model whose per-step cost does not grow with context length, an orthogonal solution to the same underlying problem.

Hierarchical / multi-scale byte and token models. Hourglass Transformers [Nawrot et al., 2022] introduce a U-Net-style down-sample/up-sample pattern interleaved with transformer blocks for long sequences, the general architectural template that byte-specific hierarchical models specialize. Dynamic Token Pooling [Nawrot et al., 2023] learns autoregressive segment boundaries for pooling, directly anticipating dynamic (rather than fixed-size) patching. MEGABYTE [Yu et al., 2023] is the direct realization of "bytecoder" as originally described: a local transformer within fixed-size byte patches, composed via a global transformer between patches, scaling to million-byte sequences. The Byte Latent Transformer [Pagnoni et al., 2024] goes further, replacing MEGABYTE's fixed patch size with *dynamic*, next-byte-entropy-driven patch boundaries, and demonstrates FLOP-matched scaling to 8B parameters and 4 trillion training bytes, matching tokenizer-based LLMs — this is, to our knowledge, the complete, at-scale, open-source realization of the bytecoder idea. SpaceByte [Slagle, 2024] offers a cheaper alternative to BLT's learned entropy patcher: it inserts global-model compute only after heuristic boundary bytes (e.g. whitespace), beating MEGABYTE at much lower cost. MrT5 [Kallini et al., 2024] learns to merge or delete redundant byte positions inside a ByT5-style encoder dynamically, a related but architecturally distinct compression mechanism. MBLM [Egli et al., 2025] generalizes the MEGABYTE/BLT hierarchical decomposition to arbitrary byte-serialized modalities, showing the space is still actively being extended.

Summary. Every architectural element of "bytecoder" as originally posed — byte- level input, no tokenizer, a local model within chunks, a global model across chunks, and even dynamic/entropy-driven chunk boundaries — already exists, separately and in combination, most completely in MEGABYTE and BLT. A paper claiming this combination as a new architecture would not be defensible against this prior art. What is comparatively unexplored is (a) applying this family of architectures to a domain with byte statistics very different from natural-language text, and (b) replacing BLT's *learned* neural entropy patcher with a cheap, non-learned classical entropy estimator, closer in spirit to SpaceByte's rule-based boundary but adaptive rather than a fixed heuristic. We pursue both in Section 3.

Strategy	Steps	Avg. patch len. (bytes)	Val. bits/byte	Throughput (bytes/s)
fixed (8-byte)	3000	8.0	**7.57**	$\approx$ 173,000
ppm (classical entropy)	1500	16.5	8.000 (degenerate)	$\approx$ 44,000
learned (neural entropy)	—	—	did not complete	—

Table 1: Real, measured results on the C64 binary corpus. 8.0 bits/byte equals $\log_2 256$, the entropy of a uniform byte distribution.

3 Our Contribution: Classical-Entropy Patching on C64 Binaries

3.1 Method

We implement a small MEGABYTE/BLT-style hierarchical byte transformer ($\approx$ 500K parameters): a byte embedding is fed into a local causal transformer within each patch, patch representations are mean-pooled and processed by a causal global transformer, and the resulting global context conditions the local decoder that predicts the next byte. We compare three strategies for choosing patch boundaries:

fixed Fixed 8-byte patches, the MEGABYTE baseline.

ppm Patch boundaries from a cheap, non-learned, adaptive order-2 byte-frequency model (an adaptive PPM-style predictor updated causally as it reads the stream): a boundary is placed whenever the model's surprisal at the next byte spikes above its running average. This targets the specific gap identified above — can a classical, zero- training compressor statistic substitute for BLT's learned entropy signal?

learned Patch boundaries from a small, separately-trained order-2 GRU byte language model's entropy signal, included as a reference point closer to what BLT actually does.

3.2 Data

We use a real, non-fabricated 3MB corpus assembled from 116 actual Commodore 64 game `.PRG` binaries, split 90/10 into train and validation sets. Commodore 64 program binaries have byte statistics quite different from natural-language text (dense 6502 machine code, sprite and character-set data, packed/compressed loaders), and we are not aware of any existing byte-level language modeling paper that evaluates on this kind of corpus, which is also directly relevant to related work on C64 game-playing agents in our own research program.

3.3 Setup

Training ran on Apple Silicon (MPS backend). Code: `experiment.py`, corpus: `corpus/c64_prg.bin`, results: `results/results.jsonl`, all distributed alongside this paper.

3.4 Results

3.5 Honest discussion

The fixed-patch baseline trained stably and reproducibly across repeated runs (val. bits-per-byte between 7.57 and 7.61), confirming that the basic hierarchical byte architecture works on real C64 binary data at this small scale — an unremarkable but real, working result.

The classical-entropy (`ppm`) strategy is a genuine negative result. Early attempts produced NaN losses from numerical overflow in attention and layer-normalization on padded, highly variable-length patches; after adding normalization and NaN-safe clamping, training no longer crashed, but the model converged to an exactly uniform predictor, learning nothing useful. Our best explanation is that the classical entropy heuristic produces substantially higher patch-length variance (roughly 1–32 bytes, versus a constant 8) than the fixed baseline, and our simplified implementation's fixed-size padding scheme for batching variable-length patches could not absorb that variance without the NaN-guard clamp also suppressing the useful gradient signal. We do not read this as evidence that entropy-based patching is unsound in general — BLT's production implementation handles exactly this variability successfully at scale, with substantially more careful engineering (padding/bucketing strategy, warmup, normalization) than our toy version — but as a disclosable, specific numerical-stability pitfall for anyone attempting an experiment at this scale with a simplified batching scheme.

The learned-entropy reference variant did not produce a completed result within our available compute and time budget across repeated attempts, and we report it as incomplete future work rather than as a negative result: we have no evidence either way about whether it would train successfully in our setup.

3.6 What this experiment does and does not show

It shows that a small hierarchical byte transformer trains on real, domain-novel C64 binary data using fixed-size patching. It does *not* show whether classical, non-learned entropy signals are a viable substitute for BLT's learned patcher — the comparison surfaced an implementation-level training-stability problem before it could test the underlying hypothesis. A fair test of that hypothesis would need, at minimum: a properly engineered variable-length-patch batching scheme (e.g. length-bucketing rather than pad-to-maximum), a longer training budget, and a completed learned-entropy baseline to compare against. We leave this as the concrete next step.

4 Conclusion

"Bytecoder" as originally posed is already a well-developed research direction, most completely realized by MEGABYTE and Meta's Byte Latent Transformer; we make no claim of architectural novelty. Our own small, honestly-reported experiment on real C64 binary data confirms the basic hierarchical byte architecture is workable at toy scale, but leaves the specific question of whether cheap classical entropy signals can replace a learned neural patcher open, having surfaced a real implementation pitfall rather than a conclusive answer. We view the domain (byte-level modeling of non-linguistic, structured binary data such as retro-console programs) as a genuinely underexplored corner worth returning to with a more carefully engineered implementation.

Code Availability

All code, the C64 corpus, and the raw experiment logs (including the superseded intermediate runs, kept for transparency) are available at `https://github.com/pannous/bytecoder`.

References

N. Kalchbrenner, L. Espeholt, K. Simonyan, A. van den Oord, A. Graves, and K. Kavukcuoglu. Neural machine translation in linear time. arXiv:1610.10099, 2016.

J. H. Clark, D. Garrette, I. Turc, and J. Wieting. CANINE: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the ACL*, 2022. arXiv:2103.06874.

S. Egli, M. Manica, and J. Born. Multiscale byte language models — a hierarchical architecture for causal million-length sequence modeling. arXiv:2502.14553, 2025.

A. Karpathy. The unreasonable effectiveness of recurrent neural networks. Blog post, 2015. `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`.

J. Kallini, S. Murty, C. D. Manning, C. Potts, and R. Csordás. MrT5: Dynamic token merging for efficient byte-level language models. arXiv:2410.20771, 2024.

P. Nawrot, S. Tworkowski, M. Tyrolski, L. Kaiser, Y. Wu, S. Szegedy, and H. Michalewski. Hierarchical transformers are more efficient language models. *Findings of NAACL*, 2022. arXiv:2110.13711.

P. Nawrot, J. Chorowski, A. Łańcucki, and E. M. Ponti. Efficient transformers with dynamic token pooling. In *ACL*, 2023. arXiv:2211.09761.

A. Pagnoni, R. Pasunuru, et al. Byte latent transformer: Patches scale better than tokens. arXiv:2412.09871, 2024.

K. Slagle. SpaceByte: Towards deleting tokenization from large language modeling. In *NeurIPS*, 2024. arXiv:2404.14408.

Y. Tay, V. Q. Tran, S. Ruder, J. Gupta, H. W. Chung, D. Bahri, Z. Qin, S. Baumgartner, C. Yu, and D. Metzler. Charformer: Fast character transformers via gradient-based subword tokenization. In *ICLR*, 2021. arXiv:2106.12672.

Findings of EMNLP. Tokenization falling short: On subword robustness in large language models. arXiv:2406.11687, 2024.

J. Wang, T. Gangavarapu, J. N. Yan, and A. M. Rush. MambaByte: Token-free selective state space model. In *COLM*, 2024. arXiv:2401.13660.

L. Yu, D. Simig, C. Flaherty, A. Aghajanyan, L. Zettlemoyer, and M. Lewis. MEGABYTE: Predicting million-byte sequences with multiscale transformers. arXiv:2305.07185, 2023.

L. Xue, A. Barua, N. Constant, R. Al-Rfou, N. Narang, M. Kale, A. Roberts, and C. Raffel. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the ACL*, 2022. arXiv:2105.13626.