OPTIMIZING GENOMIC LANGUAGE MODELS FOR EFFICIENT TRAINING, FINE-TUNING, AND INFERENCE

Ali Saadat, Jacques Fellay
School of Life Sciences
Ecole Polytechnique Fédérale de Lausanne
Lausanne, Switzerland
{ali.saadat, jacques.fellay}@epfl.ch

ABSTRACT

Genomic language models are widely used as general-purpose encoders for sequences, enabling transfer to tasks such as regulatory-element prediction, variant scoring, and genome-wide annotation. In practice, however, both deployment and adaptation are frequently constrained by GPU memory and throughput, especially for long inputs, heterogeneous sequence lengths, and larger checkpoints. We present a unified, efficiency-oriented implementation for inference and fine-tuning of two encoder-only model families: GENA-LM (110M, 336M) and Nucleotide Transformer (500M, 2.5B). Our framework combines (i) IO-aware attention kernels (FlashAttention) that avoid materializing the full attention matrix, (ii) padding-free variable-length execution via sequence packing, (iii) token-budget batching to stabilize utilization under mixed lengths, and (iv) parameter-efficient adaptation using LoRA. We additionally support optional memory knobs, including activation checkpointing, ZeRO stage-2 CPU offload, and 4-bit/8-bit quantization. Across four model sizes, we validate numerical fidelity via pseudo-perplexity agreement with a reference implementation, characterize inference memory and throughput scaling with sequence length and batch size, and isolate training-time memory contributions in a progressive ablation. On downstream binary classification benchmarks (promoter and enhancer prediction), the efficient implementation yields considerable fine-tuning speedups and peak-memory reductions while maintaining broadly comparable task performance. Finally, we quantify quantization-induced drift by measuring embedding cosine similarity across model sizes, highlighting model-dependent trade-offs.

1 INTRODUCTION

Genomic language models (GLMs) learn contextual representations of DNA sequence from large unlabeled corpora and can be reused as pretrained encoders for a wide range of downstream biology tasks (Consens et al., 2025; Feng et al., 2025). This "foundation model" workflow is attractive because it concentrates the up-front cost of representation learning in a single backbone: once pretrained, the same encoder can be adapted to many prediction problems by attaching a lightweight head or performing task-specific fine-tuning (Vaswani et al., 2017; Bommasani et al., 2021). This reuse is particularly useful for sequence-based applications, where accurate prediction depends on combining signals across scales such as short motifs, local composition, and longer-range dependencies (Benegas et al., 2025).

Even when pretrained language models transfer well, the cost of running and adapting them can become a bottleneck, and genomic language models face the same challenge in long-context genomics settings (Wan et al., 2023; Bai et al., 2024). These constraints often limit routine workflows and motivate efficiency-focused implementations that reduce resource use while preserving model behavior. One way to address these limitations is to change the model architecture. For example, HyenaDNA replaces dense attention with convolutions, enabling longer contexts while scaling sub-quadratically with sequence length (Nguyen et al., 2023). However, architectural changes are not drop-in replacements for widely used Transformer models and do not directly address a common deployment

question: for existing models, how much efficiency can be recovered through implementation and training-system choices alone?

Recent work suggests this gap can be significant (Çelik & Xie, 2025). In protein language modeling, careful reimplementations that use modern kernels and variable-length execution have improved runtime and reduced memory while closely matching reference behavior (Çelik & Xie, 2025). Similar implementation-driven improvements have also been demonstrated in long-context regulatory genomics models—for example, Flashzoi refactors Borzoi's positional encoding to better support FlashAttention-2, reducing computational overhead while maintaining predictive performance (Hingerl et al., 2025). Motivated by these observations, we study implementation-level optimizations for widely deployed Transformer-based GLMs.

We integrate a set of complementary optimizations that target attention memory, padding waste, and fine-tuning footprint, and we evaluate (i) output agreement with a reference forward pass, (ii) inference memory/throughput scaling with sequence length and batch size, (iii) training-time memory under progressive optimization knobs, and (iv) downstream fine-tuning efficiency and task performance. Our results show that much of the deployment cost is implementation-dependent, providing a practical path to more efficient deployment and adaptation of existing GLMs.

2 METHOD

2.1 OVERVIEW

Our goal is to make existing Transformer-based genomic language models more practical for routine inference and fine-tuning by reducing GPU memory use and improving throughput without changing model architectures, pretrained weights, or task formulations. We focus on five questions:

1. Does the efficient implementation preserve the numerical behavior of the reference model?
2. How do peak memory and throughput scale with sequence length and batch size?
3. Which optimizations contribute most to training-time memory reduction?
4. Can downstream task performance be maintained while reducing training time and memory?
5. What are the memory–fidelity trade-offs of 4-bit and 8-bit quantization?

We address these questions using a unified implementation that combines exact-attention kernels, padding-free execution for variable-length inputs, and parameter-efficient adaptation for fine-tuning. Mathematical details are provided in Appendix A.

2.2 MODELS

We study two widely used encoder-only GLM families trained with masked language modeling: GENA-LM (Fishman et al., 2025) (110M, 336M) and Nucleotide Transformer (NT) (Dalla-Torre et al., 2025) (500M, 2.5B). While both families follow a similar pretraining recipe, they differ in model design and implementation choices, including positional encoding and tokenization. These differences help assess how broadly efficiency techniques carry over across families and where family-specific handling is required.

2.3 EFFICIENCY TECHNIQUES

We apply techniques that target distinct bottlenecks in attention, padding waste, and training footprint, while keeping the forward computation of attention exact when quantization is not enabled.

IO-aware exact attention via FlashAttention. Standard scaled dot-product attention materializes an $L \times L$ attention matrix and incurs $O(L^2)$ memory in sequence length L. We replace this with FlashAttention (Dao, 2023), which computes the same attention outputs while avoiding explicit materialization through tiled, fused kernels. This reduces attention-related activation memory to $O(L)$ and typically improves effective memory bandwidth utilization, providing the largest benefit at longer sequences.

Padding-free variable-length execution via sequence packing. Genomic datasets often exhibit substantial length heterogeneity. Padding to the longest sequence in a batch wastes compute and increases attention memory (Krell et al., 2022). We eliminate padding overhead by packing variable-length sequences into a contiguous token stream and maintaining boundaries through a prefix-sum array of cumulative sequence lengths. We then call FlashAttention's variable-length interface, which computes attention only over real tokens and prevents attention from crossing sequence boundaries.

Token-budget batching for stable utilization. To reduce step-time variance and limit worst-case memory spikes under heterogeneous lengths, we form batches by capping the total number of tokens per forward pass rather than fixing the number of sequences (Ott et al., 2019). Concretely, we construct each batch by adding examples until a global token budget is reached (after packing), then start a new batch. This yields more predictable memory and throughput, especially when combined with packed variable-length execution.

Activation checkpointing. During fine-tuning, activation storage can dominate memory for smaller and mid-sized models. We apply gradient checkpointing at the transformer-layer granularity, trading recomputation for reduced activation memory (Chen et al., 2016).

Parameter-efficient fine-tuning with LoRA. Full fine-tuning requires gradients and optimizer state for all parameters, which is prohibitive for large checkpoints. We use Low-Rank Adaptation (LoRA) (Hu et al., 2021), inserting low-rank updates into attention projections while freezing the pretrained weights. This reduces the number of trainable parameters and associated optimizer state by orders of magnitude.

Optional optimizer-state offload (ZeRO Stage-2). For the largest checkpoints, optimizer state can dominate GPU memory even after attention optimization. We optionally enable ZeRO Stage-2 (Rajbhandari et al., 2019) with CPU offload, which partitions optimizer states and gradients across devices and can offload them to host memory. This reduces peak GPU memory at the cost of additional CPU–GPU transfers.

Optional 4-bit / 8-bit quantization. To further reduce parameter memory at inference time, we optionally apply 4-bit or 8-bit quantization using bitsandbytes (Dettmers et al., 2022). Because quantization changes numerical behavior, we treat it as a separate knob and explicitly characterize drift relative to BF16.

2.4 Experimental Setup

Evaluation axes and metrics. We organize experiments to align with our five questions:

1. **Fidelity**: agreement between baseline and efficient implementations, measured by Pearson correlation (r) between pseudo-perplexity values.
2. **Inference scaling**: peak GPU memory (GB) and throughput (sequences/s) as functions of sequence length and batch size.
3. **Training memory**: peak GPU memory (GB) under progressive optimization stages.
4. **Downstream fine-tuning**: classification accuracy, training time (s), and peak GPU memory (GB).
5. **Quantization trade-offs**: parameter memory and embedding cosine similarity relative to BF16.

Data. We use two categories of inputs:

- **Synthetic sequences**: uniformly sampled from $\{$A,C,G,T$\}$ with fixed random seed (42) to isolate implementation and hardware effects. For pseudo-perplexity, we evaluate 50 sequences of length 1000.
- **Downstream tasks**: promoter prediction (300 bp) and enhancer prediction (400 bp) from the Nucleotide Transformer benchmark suite (Dalla-Torre et al., 2025).

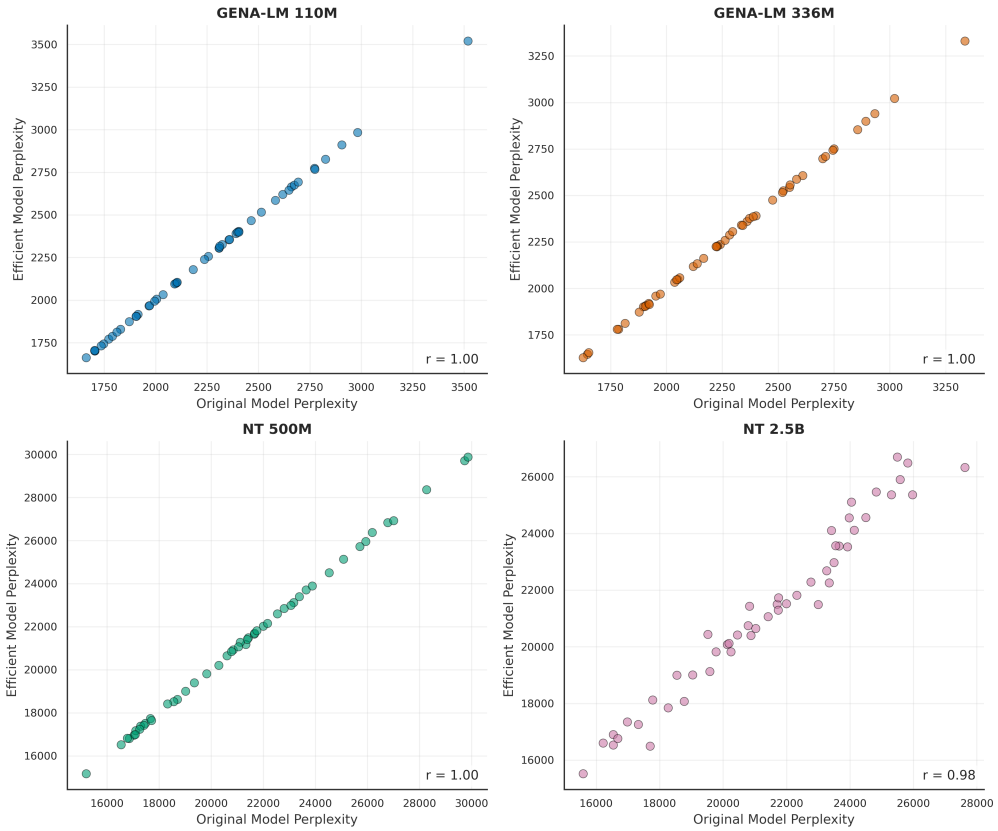


Figure 1: Pseudo-perplexity agreement between baseline and efficient implementations. Each point corresponds to one sequence (50 total). Pearson correlations are shown in the legend.

Fine-tuning configuration. Unless stated otherwise, we fine-tune with LoRA rank 16, dropout = 0.1, weight decay = 0.01, and batch size 16 for 5 epochs using AdamW with learning rate 10^{-4}. LoRA adapters are applied to query, key, and value projections.

Hardware and measurement protocol. All experiments run on NVIDIA H100 GPU. Peak memory corresponds to the maximum GPU memory observed during each operation. Inference benchmarks use 10 runs and report mean values.

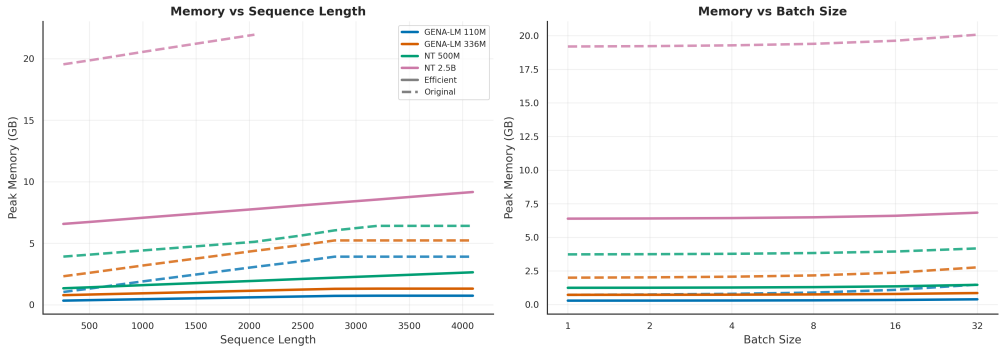
3 Results

3.1 Fidelity: Efficient Forward Pass Tracks the Reference Implementation

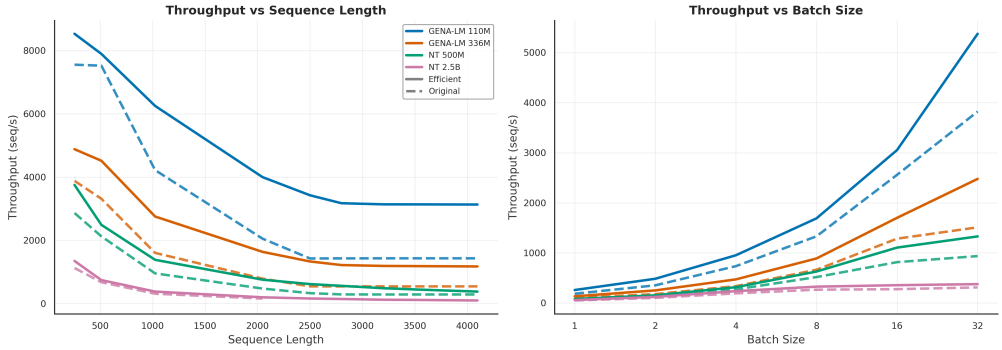
We first test whether the efficient implementation preserves model behavior by comparing pseudo-perplexity on 50 synthetic sequences of length 1000. Figure 1 shows strong agreement across all checkpoints, with Pearson correlations reported in the legend. This indicates that exact-attention kernels and packed variable-length execution preserve the forward-pass behavior of the reference models in the tested setting.

3.2 Inference Scaling: Memory Savings and Throughput Gains Increase with Length and Batch Size

We benchmark inference across sequence lengths from 256 to 4096 at batch size 16, and across batch sizes from 1 to 32 at sequence length 1024. Figure 2 compares baseline and efficient implementations.



(a) Peak GPU memory versus sequence length (left) and batch size (right).



(b) Inference throughput versus sequence length (left) and batch size (right).

Figure 2: Inference scaling for baseline versus efficient implementation. Memory savings and throughput gains grow with sequence length and batch size as attention- and padding-related costs become dominant.

Peak memory. The memory gap widens with both sequence length and batch size (Figure 2a). At shorter lengths, parameter storage is a dominant component. As L increases, attention activations become the limiting factor: baseline attention grows as $O(L^2)$, while FlashAttention reduces attention activation memory to $O(L)$. Packed execution further avoids padding-driven inflation when lengths vary, improving both memory efficiency and effective utilization.

Throughput. Throughput gains follow a similar trend (Figure 2b). FlashAttention reduces memory traffic through fused kernels, while packed execution removes compute spent on padding tokens. The combined benefit is most pronounced in regimes where attention and padding dominate runtime, namely longer sequences and larger batches.

3.3 Training Memory: Progressive Ablation Highlights Regime Shifts with Model Size

To isolate the contribution of each optimization, we measure peak GPU memory at sequence length 1000 and batch size 16 under progressive optimization stages (Figure 3).

FlashAttention. FlashAttention provides the largest single reduction across all checkpoints, consistent with attention activations being a major memory contributor at this length.

Checkpointing. Checkpointing provides a smaller incremental benefit than FlashAttention in this setting because FlashAttention already removes the largest attention-related activation term at L=1000. After that reduction, non-activation components (e.g., parameters, optimizer/gradient buffers, and miscellaneous allocator overheads) constitute a larger fraction of peak memory, limiting the marginal gains from further activation-saving alone.

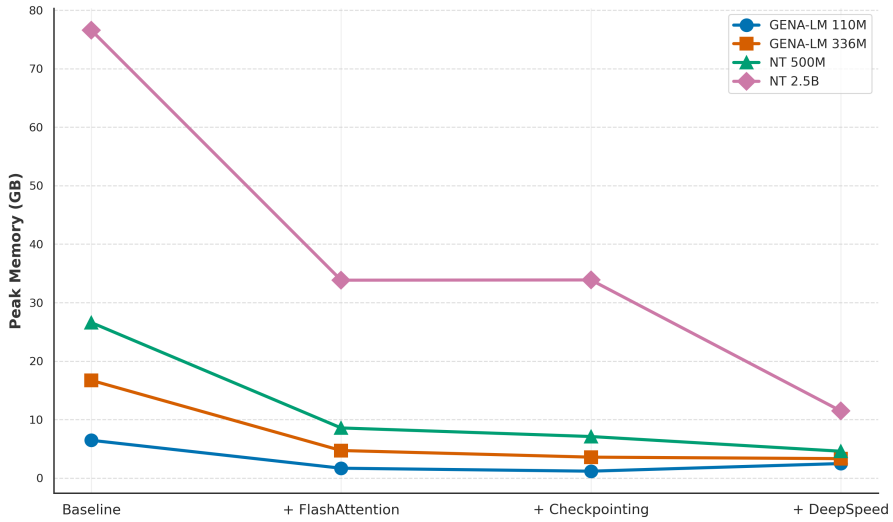


Figure 3: Training peak GPU memory under progressive optimizations. FlashAttention yields the largest initial reduction; checkpointing and ZeRO provide additional savings, with ZeRO most impactful for the largest model.

ZeRO offload. ZeRO Stage-2 CPU offload provides the largest incremental benefit for the NT 2.5B checkpoint (33.88 $\rightarrow$ 11.49 GB), where optimizer state dominates after attention is optimized. For smaller models, the CPU-offload machinery and transfer buffers can outweigh the savings (e.g., GENA-LM 110M increases from 1.18 to 2.49 GB), making ZeRO most useful when optimizer/gradient state is a substantial component of the training footprint, while offering limited benefit (or overhead) when peak memory is dominated by other factors.

3.4 Downstream Fine-Tuning: Large Efficiency Gains with Comparable Accuracy

We fine-tune all checkpoints on promoter and enhancer binary classification using LoRA and compare baseline and efficient implementations (Figure 4).

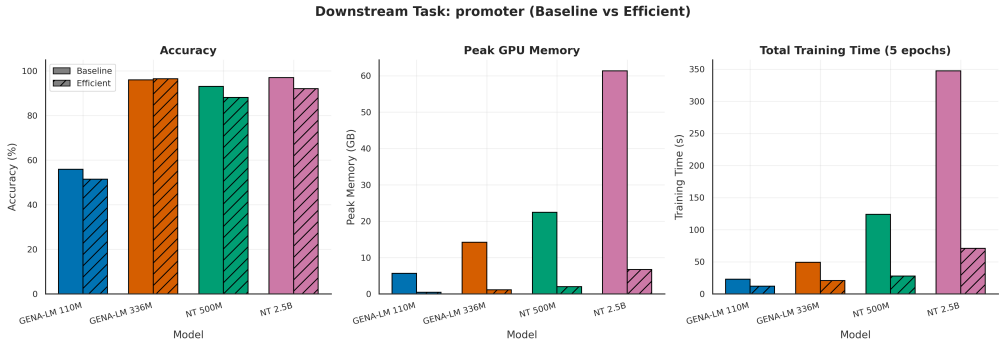
Efficiency. The efficient implementation substantially improves end-to-end fine-tuning efficiency. Speedups generally increase with model size, consistent with larger checkpoints benefiting more from improved kernel efficiency and reduced memory traffic.

Task performance. Across models and tasks, the efficient and baseline implementations achieve broadly comparable classification accuracy. Small differences are expected due to mixed precision and the interaction between training dynamics and implementation details, but the results suggest that the efficiency gains do not require significant sacrifice of downstream performance in these settings.

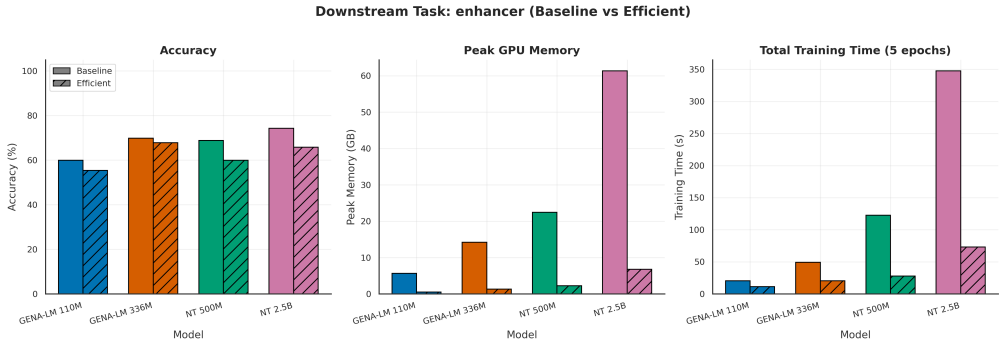
3.5 Quantization: Additional Memory Savings with Model-Dependent Drift

We evaluate 4-bit and 8-bit quantization using a conservative strategy for NT models and quantify drift relative to BF16 via embedding cosine similarity (Figure 5).

Fidelity. All models except NT 2.5B maintain embedding similarity above 0.98. NT 2.5B is more sensitive: quantization lowers similarity to roughly 0.9. A practical takeaway is that representational drift under quantization is model-dependent and can be noticeably larger for the largest checkpoint, even under the same quantization recipe.



(a) Promoter prediction.



(b) Enhancer prediction.

Figure 4: Downstream fine-tuning comparison. The efficient implementation achieves training speedup and peak-memory reduction with broadly comparable accuracy.

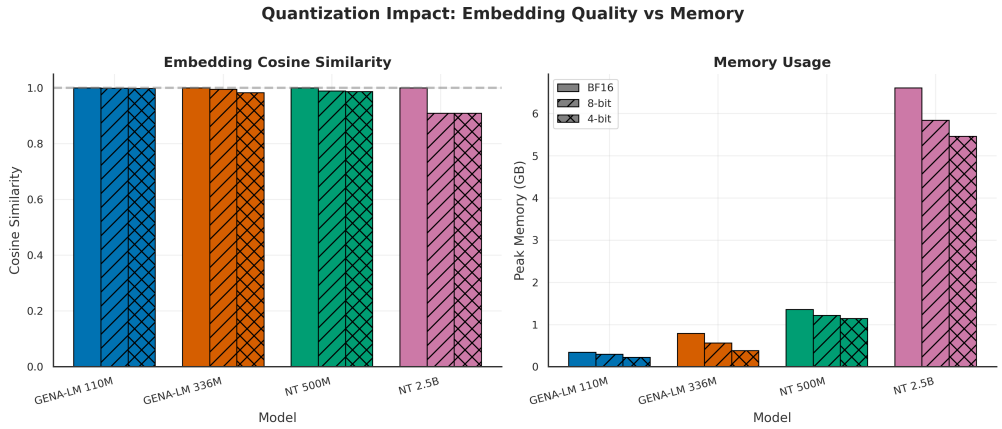


Figure 5: Quantization trade-offs. Left: embedding cosine similarity with BF16. Right: peak memory.

Memory. Quantization reduces parameter memory and can enable deployment on more memory-constrained hardware. For NT 2.5B, memory decreases from 6.7 GB (BF16) to 5.6 GB (4-bit). The relative gains are smaller for smaller models because non-parameter components (activations and temporary buffers) constitute a larger fraction of peak memory in many workloads.

4 DISCUSSION

Genomic language models are increasingly used as reusable encoders for DNA sequence, but their practical adoption is often constrained by GPU memory and throughput, especially for long inputs, heterogeneous sequence lengths, and large checkpoints. In this study, we presented a unified implementation designed to reduce these bottlenecks without modifying Transformer architectures or pretrained weights. Across two widely used encoder-only families, GENA-LM and Nucleotide Transformer, we combined exact-attention kernels (FlashAttention), padding-free variable-length execution (sequence packing), token-budget batching for stable utilization, and parameter-efficient adaptation (LoRA). We also evaluated optional memory-oriented knobs, including activation checkpointing, ZeRO Stage-2 CPU offload, and 4-bit/8-bit quantization.

A consistent pattern across experiments is that deployment cost is strongly shaped by implementation choices, particularly in long-sequence regimes. Fidelity experiments showed strong agreement between the baseline and efficient implementations when using exact-attention kernels and packed variable-length execution, supporting the premise that substantial efficiency gains are possible without altering model function in the non-quantized setting. Inference benchmarks further indicated that benefits increase with sequence length and batch size, consistent with the asymptotic mismatch between baseline $O(L^2)$ attention memory and the $O(L)$ activation footprint enabled by FlashAttention, as well as the removal of padding-related waste under packing. The progressive training ablation highlighted a regime shift with model size: once attention and activation memory are reduced, large checkpoints become dominated by optimizer state, making ZeRO offload most impactful for the largest model while offering limited benefit (or even overhead) for smaller checkpoints.

On downstream promoter and enhancer classification, the same system-level changes produced large reductions in wall-clock training time and peak GPU memory while maintaining broadly comparable accuracy. This suggests that for fine-tuning workflows, efficiency-oriented implementations can deliver large practical gains without requiring new model variants or specialized architectures. Quantization provided an additional path to reduce parameter memory.

This study has several limitations. First, fidelity was evaluated using pseudo-perplexity agreement on synthetic sequences at a fixed length, which is a direct check of output consistency but does not exhaustively cover longer contexts, real genomic sequence distributions, or rare edge cases. Second, benchmarks were run on H100 94GB GPUs, so absolute throughput and the relative impact of offloading may differ on smaller GPUs or on systems with different CPU–GPU bandwidth and interconnect characteristics. Third, downstream evaluation was limited to two short-input binary classification tasks, which do not fully represent long-context use cases such as sliding-window locus scanning, genome-wide annotation, or variant-effect scoring over larger receptive fields. Finally, quantization was assessed via embedding cosine similarity as a proxy for representational drift; while informative, task-level evaluation under quantized inference or fine-tuning would be required to translate these changes into application-level impact.

Overall, our results indicate that a large fraction of GLM deployment cost is implementation-dependent, and that substantial efficiency gains are achievable for widely used Transformer checkpoints without architectural changes. By providing a modular implementation and a systematic evaluation suite, this work offers a practical path to extending GLM usage to longer sequences, larger models, and memory-constrained deployment settings common in applied genomics pipelines.

REFERENCES

Guangji Bai, Zheng Chai, Chen Ling, Shiyu Wang, Jiaying Lu, Nan Zhang, Tingwei Shi, Ziyang Yu, Mengdan Zhu, Yifei Zhang, Xinyuan Song, Carl Yang, Yue Cheng, and Liang Zhao. Beyond efficiency: A systematic survey of resource-efficient large language models. 2024.

Gonzalo Benegas, Chengzhong Ye, Carlos Albors, Jianan Canal Li, and Yun S Song. Genomic language models: opportunities and challenges. *Trends Genet.*, 41(4):286–302, April 2025.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel,

Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W Thomas, Florian Tramèr, Rose E Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models. 2021.

Muhammed Hasan Çelik and Xiaohui Xie. Efficient inference, training, and fine-tuning of protein language models. *iScience*, 28(10):113495, October 2025.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost, 2016. URL https://arxiv.org/abs/1604.06174.

Micaela E Consens, Cameron Dufault, Michael Wainberg, Duncan Forster, Mehran Karimzadeh, Hani Goodarzi, Fabian J Theis, Alan Moses, and Bo Wang. Transformers and genome language models. *Nat. Mach. Intell.*, March 2025.

Hugo Dalla-Torre, Liam Gonzalez, Javier Mendoza-Revilla, Nicolas Lopez Carranza, Adam Henryk Grzywaczewski, Francesco Oteri, Christian Dallago, Evan Trop, Bernardo P de Almeida, Hassan Sirelkhatim, Guillaume Richard, Marcin Skwark, Karim Beguir, Marie Lopez, and Thomas Pierrot. Nucleotide transformer: building and evaluating robust foundation models for human genomics. *Nat. Methods*, 22(2):287–297, February 2025.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. 2023.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. LLM.int8(): 8-bit matrix multiplication for transformers at scale. 2022.

Haonan Feng, Lang Wu, Bingxin Zhao, Chad Huff, Jianjun Zhang, Jia Wu, Lifeng Lin, Peng Wei, and Chong Wu. Benchmarking DNA foundation models for genomic and genetic tasks. *Nat. Commun.*, 16(1):10780, November 2025.

Veniamin Fishman, Yuri Kuratov, Aleksei Shmelev, Maxim Petrov, Dmitry Penzar, Denis Shepelin, Nikolay Chekanov, Olga Kardymon, and Mikhail Burtsev. GENA-LM: a family of open-source foundational DNA language models for long sequences. *Nucleic Acids Res.*, 53(2), January 2025.

Johannes C Hingerl, Alexander Karollus, and Julien Gagneur. Flashzoi: an enhanced borzoi for accelerated genomic analysis. *Bioinformatics*, 41(9), September 2025.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank adaptation of large language models. 2021.

Mario Michael Krell, Matej Kosec, Sergio P. Perez, and Andrew Fitzgibbon. Efficient sequence packing without cross-contamination: Accelerating large language models without impacting performance, 2022. URL https://arxiv.org/abs/2107.02027.

Eric Nguyen, Michael Poli, Marjan Faizi, Armin Thomas, Callum Birch-Sykes, Michael Wornow, Aman Patel, Clayton Rabideau, Stefano Massaroli, Yoshua Bengio, Stefano Ermon, Stephen A Baccus, and Chris Ré. HyenaDNA: Long-range genomic sequence modeling at single nucleotide resolution. 2023.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In Waleed Ammar, Annie Louis, and Nasrin Mostafazadeh (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pp. 48–53, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-4009. URL https://aclanthology.org/N19-4009/.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. ZeRO: Memory optimizations toward training trillion parameter models. 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Zhongwei Wan, Xin Wang, Che Liu, Samiul Alam, Yu Zheng, Jiachen Liu, Zhongnan Qu, Shen Yan, Yi Zhu, Quanlu Zhang, Mosharaf Chowdhury, and Mi Zhang. Efficient large language models: A survey. 2023.

A Technical Notes

A.1 Packing and variable-length attention

We consider a batch of B sequences with token lengths $\{L_i\}_{i=1}^{B}$. For padded execution, sequences are padded to $L_{\max} = \max_i L_i$. For packed execution, only real tokens are concatenated into a single stream of length $T = \sum_i L_i$, together with cumulative offsets

$$\text{cu\_seqlens}[0] = 0, \qquad \text{cu\_seqlens}[i] = \sum_{j=1}^{i} L_j, \;\; i = 1, \ldots, B. \tag{1}$$

We use FlashAttention's variable-length interface with `cu_seqlens` so that attention is computed independently within each segment and does not cross sequence boundaries.

A.2 Fidelity metric: pseudo-perplexity for masked LMs

Perplexity is not directly defined for masked language models. We therefore use pseudo-perplexity via a pseudo-likelihood objective that masks one position at a time. For a tokenized sequence $x = (x_1, \ldots, x_L)$, let $x_{\setminus t}$ denote the sequence where x_t is replaced by `[MASK]`. The model yields conditional probabilities $p_\theta(x_t \mid x_{\setminus t})$. We compute

$$\text{pPPL}(x) = \exp\left(-\frac{1}{|\mathcal{I}(x)|} \sum_{t \in \mathcal{I}(x)} \log p_\theta(x_t \mid x_{\setminus t}) \right), \tag{2}$$

where $\mathcal{I}(x)$ excludes padding and special tokens (e.g., `[CLS]`, `[SEP]`) for consistency across implementations. To quantify agreement between baseline and efficient implementations, we compute pPPL per sequence and report Pearson correlation across sequences.

A.3 Benchmarking conventions

Peak GPU memory. We report peak allocated GPU memory during the measured region using the framework allocator statistics (e.g., PyTorch CUDA memory stats). Peak measurements can be affected by temporary buffers, caching, and fragmentation; we therefore compare implementations under matched settings.

Throughput. We report inference throughput as sequences/second. When lengths vary, tokens/second is also informative:

$$\text{tokens/s} = \frac{\sum_{i=1}^{B} L_i}{\text{step time (s)}}. \tag{3}$$

All throughput results use the same measurement protocol described in the main text.