PHOTON: Hierarchical Autoregressive Modeling for Lightspeed and Memory-Efficient Language Generation

Yuma Ichikawa[1, 2], **Naoya Takagi**[1], **Takumi Nakagawa**[1, 3], **Yuzi Kanazawa**[1], **Akira Sakai**[1, 4]

[1]Fujitsu Limited, [2]RIKEN Center for AIP, [3]Institute of Science Tokyo, [4]Tokai University

Correspondence: ichikawa.yuma@fujitsu.com

Abstract

Transformers operate as *horizontal token-by-token scanners*; at each generation step, attending to an ever-growing sequence of token-level states. This access pattern increases prefill latency and makes long-context decoding more memory-bound, as KV-cache reads and writes dominate inference time over arithmetic operations. We propose **P**arallel **H**ierarchical **O**peration for **TO**p-down **N**etworks (**PHOTON**), a hierarchical autoregressive model that replaces horizontal scanning with vertical, multi-resolution context scanning. PHOTON maintains a hierarchy of latent streams: a bottom-up encoder compresses tokens into low-rate contextual states, while lightweight top-down decoders reconstruct fine-grained token representations in parallel. We further introduce *recursive generation* that updates only the coarsest latent stream and eliminates bottom-up re-encoding. Experimental results show that PHOTON is superior to competitive Transformer-based language models regarding the throughput-quality trade-off, providing advantages in long-context and multi-query tasks. In particular, this reduces decode-time KV-cache traffic, yielding up to $10^3\times$ higher throughput per unit memory.

1 Introduction

Transformer-based language models have achieved remarkable capabilities; however, the inference cost rapidly increases with context length under recent serving workloads (Bahdanau et al., 2014; Vaswani et al., 2017). Even with KV caching, autoregressive Transformers operate as horizontal *token-by-token scanners*; each new token attends to a continually growing flat history of token-level states. The *prefill* stage computes and stores the KV cache for the entire prompt. During *decoding*, throughput becomes increasingly memory-bound as the context grows, since each step repeatedly reads from and updates a large KV cache. As a result, performance is often limited by memory bandwidth rather than computational capacity. The bottleneck is most pronounced in long-context, multi-query serving.

This raises a simple question: ***Must generation remain horizontal token-by-token scanning over a flat history?*** The structure of natural language suggests otherwise (Chomsky, 2002; Lambek, 1958; Hauser et al., 2002; Halle, 1973). Natural language is inherently hierarchical: subwords form words, words form sentences, and sentences create documents. Moreover, coherent generation relies on maintaining an evolving discourse state rather than repeatedly revisiting all fine-grained token representations (Mann and Thompson, 1988; Grosz and Sidner, 1986; Grosz et al., 1995). These observations motivate *vertical scanning*, which represents context through compact coarse states and descends to token-level detail only when necessary.

Hierarchical or multi-scale sequence modeling has recently been explored (Pappagari et al., 2019; Han et al., 2021; Dai et al., 2020; Nawrot et al., 2022, 2023; Fleshman and Van Durme, 2023; Mujika, 2023; Yu et al., 2023; Ho et al., 2024). In particular, Block Transformer (Ho et al., 2024) reduces inference-time KV overhead by separating coarse block-level computation from token-level decoding. However, it employs only a single-level hierarchy that largely serves as a block-structured attention mechanism for efficiency, rather than maintaining a persistent multi-level state that is updated across abstractions during inference.

We introduce **P**arallel **H**ierarchical **O**peration for **TO**p-down **N**etworks (**PHOTON**), a hierarchical autoregressive model that replaces horizontal token-by-token scanning with multi-resolution vertical scanning over contextual states. As illustrated in Figure 1, PHOTON constructs a hierarchy of latent streams via (i) a bottom-up encoder that compresses tokens into low-rate contextual states and (ii) a top-down decoder stack that progressively re-

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 38365–38383

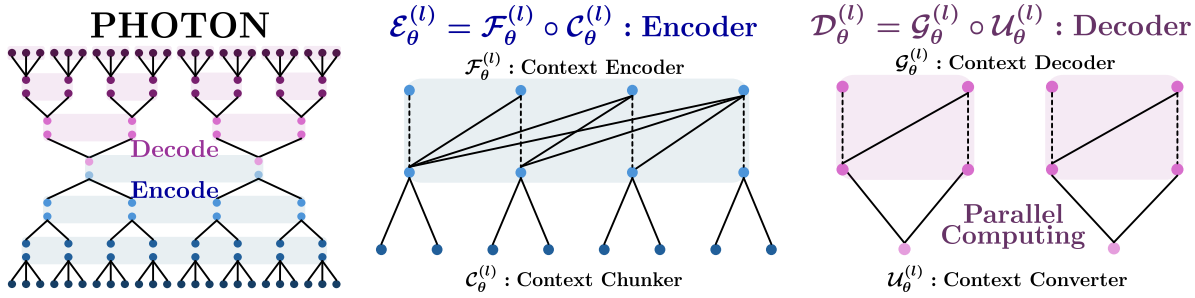


Figure 1: PHOTON overview (left). The hierarchical encoder (middle) compresses token-level states into coarser latent streams via a context chunker and an autoregressive context encoder. The hierarchical decoder (right) expands each coarse latent using a context converter and a chunk-local causal decoder with bounded attention.

constructs finer representations using local autoregressive modules with bounded attention. Since these local decoders operate independently across lower-level contexts, decoding can proceed in parallel across these contexts. PHOTON is trained using standard next-token prediction along with auxiliary objectives that enforce *recursive consistency* at multiple levels by aligning bottom-up summaries with top-down reconstructions. We also propose recursive generation: unlike Block Transformer, PHOTON avoids bottom-up re-encoding of newly generated tokens by directly updating the coarse stream from decoder-side reconstructions. This design retains only the hierarchical decoder on the GPU during decoding, thus reducing both model residency and KV-cache footprint.

Experiments show that PHOTON achieves a better Pareto frontier in terms of throughput per unit memory and quality compared to both vanilla and Block Transformer baselines. In particular, by reducing decode-time KV-cache traffic, PHOTON achieves up to $10^3\times$ higher throughput per unit of memory.

2 PHOTON

2.1 Architecture

PHOTON, as shown in Figure 1, is a hierarchical autoregressive language model that replaces token-by-token *horizontal* scanning with *vertical*, multi-resolution context scanning. It maintains two complementary paths: (i) a bottom-up encoder that groups adjacent states into chunks and compresses each chunk into a single, coarser contextual summary, yielding a progressively lower-rate stream of latents as the hierarchy ascends; and (ii) a top-down decoder that reverses this process, expanding each coarse latent back into finer states using chunk-local causal attention conditioned only on the encompassing higher-level latent. Since each chunk attends to a bounded local window rather than the full sequence, independent chunks at the same level are decoded in parallel. Table 1 collects the symbols used below, and a formal treatment of chunk index sets and causal masks is deferred to Appendix A.1.

Table 1: Notation used throughout Section 2.

Symbol	Meaning
T	input sequence length (in tokens)
L	number of hierarchy levels
C_l	chunk length at level l
$C_{\le l}$	$\prod_{k=1}^{l} C_k$, cumulative compression
M_l	# units at level l; $M_0{=}T$, $M_l{=}M_{l-1}/C_l$
D_l	hidden dimension at level l
R_l	# of conditioning vectors from the converter
$X^{(l)}$	contextualized states at level l (encoder side)
$A^{(l)}$	chunk summaries fed to the context encoder
$U^{(l)}$	converter outputs conditioning the decoder
$\widehat{X}^{(l)}$	top-down reconstruction at level l
$\mathrm{KV}^{(L)}$	top-level KV cache

Setup. Let $\mathcal{V}$ be the vocabulary, $t_{1:T} \in \mathcal{V}^T$ the input tokens, and $X^{(0)} \in \mathbb{R}^{T\times D_0}$ their embeddings. We choose chunk lengths such that T is divisible by $C_{\le L} := \prod_{k=1}^{L} C_k$; all learnable parameters are denoted by θ. Level l consists of M_l contiguous chunks, each spanning C_l level-$(l{-}1)$ units, thereby partitioning the complete set of level-$(l{-}1)$ indices into contiguous blocks $\{I_g^{(l)}\}_{g=1}^{M_l}$, as detailed in Appendix A.1. We denote $X^{(l-1)}_{I_g^{(l)}} \in \mathbb{R}^{C_l\times D_{l-1}}$ as the subtensor of level-$(l{-}1)$ states that belongs to the g-th level-l chunk.

2.1.1 Hierarchical Encoder

At each level l, the encoder operates in two stages: (i) it aggregates level-$(l-1)$ representations into chunk-level features, and (ii) it contextualizes these

features using an autoregressive context encoder, as shown in Figure 1. Formally,

$$X^{(l)} = \mathcal{E}_\theta^{(l)}(X^{(l-1)}) \coloneqq \mathcal{F}_\theta^{(l)} \circ \mathcal{C}_\theta^{(l)}(X^{(l-1)}),$$

where the chunker $\mathcal{C}_\theta^{(l)}$ and the context encoder $\mathcal{F}_\theta^{(l)}$ are defined below.

Context Chunker. The chunker maps each representation $X^{(l-1)}_{I_g^{(l)}}$ to a single chunk representation:

$$A^{(l)} \coloneqq [A^{(l)}_{1:M_l}] \in \mathbb{R}^{M_l \times D_l}, A^{(l)}_g \coloneqq \mathcal{C}_\theta^{(l)}(X^{(l-1)}_{I_g^{(l)}}).$$

In practice, $\mathcal{C}_\theta^{(l)}$ can be implemented by concatenating the representations within a chunk, followed by a linear projection or a one-dimensional convolution. This study employs concatenation as a representative instantiation.

Context Encoder. The context encoder captures dependencies among chunk embeddings $\{A_g^{(l)}\}_{g=1}^{M_l}$ using an autoregressive Transformer $X^{(l)} = \mathcal{F}_\theta^{(l)}(A^{(l)})$, producing contextualized chunk-level states at level l. Unlike standard Transformers that model sequences token-by-token, our encoder operates over chunk-level contexts, enabling long-range modeling at a coarser temporal resolution.

2.1.2 Hierarchical Decoder

At each level l, the decoder (i) converts the latent representation of each level l into a short conditioning prefix and (ii) reconstructs the level-$(l-1)$ stream using a local autoregressive decoder that operates independently within each chunk, as shown in Figure 1. Formally,

$$\widehat{X}^{(l-1)} = \mathcal{D}_\theta^{(l)}(\widehat{X}^{(l)}) \coloneqq \mathcal{G}_\theta^{(l)} \circ \mathcal{U}_\theta^{(l)}(\widehat{X}^{(l)}),$$

where the converter $\mathcal{U}_\theta^{(l)}$ and the local autoregressive decoder $\mathcal{G}_\theta^{(l)}$ are defined below.

Context Converter. To preserve causality, the decoder generates the g-th chunk at level $(l-1)$, conditioned on the previous level-l latent. We introduce a learned starting latent $\widehat{X}_0^{(l)} \in \mathbb{R}^{D_l}$ and define $U_{g-1}^{(l)} \coloneqq \mathcal{U}_\theta^{(l)}(\widehat{X}_{g-1}^{(l)}) \in \mathbb{R}^{R_l \times D_{l-1}}$ for all $g \in [M_l]$. In our implementation, $\mathcal{U}_\theta^{(l)}$ is a one-dimensional convolution that expands a single vector into R_l conditioning vectors.

Context Decoder. Given $U_{g-1}^{(l)}$, the local decoder $\mathcal{G}_\theta^{(l)}$ generates the level-$(l-1)$ states autoregressively within each chunk for $g \in [M_l]$ and $j \in [C_l]$.

$$\widehat{X}^{(l-1)}_{I_g^{(l)},j} = \mathcal{G}_\theta^{(l)}\left(U_{g-1}^{(l)}, \widehat{X}^{(l-1)}_{I_g^{(l)},<j}\right),$$

where $\widehat{X}^{(l-1)}_{I_g^{(l)},<j}$ are the previously decoded states in the same chunk. Concretely, $\mathcal{G}_\theta^{(l)}$ is a causal Transformer applied to the concatenated sequence $[U_{g-1}^{(l)}; \widehat{X}^{(l-1)}_{I_g^{(l)},<j}]$ using the standard causal mask, as detailed in Appendix A.1. Crucially, position j attends only to the R_l converter outputs and the $j-1$ preceding positions within the same chunk, meaning the attention span is bounded by R_l+C_l and is independent of T. Since each chunk is conditioned only on its enclosing higher-level context $U_{g-1}^{(l)}$, chunks are decoded *in parallel* across g.

Overall, PHOTON maps token embeddings through the encoder hierarchy and back through the decoder hierarchy:

$$\widehat{X}^{(0)} = \mathcal{D}_\theta^{(1)} \circ \cdots \circ \mathcal{D}_\theta^{(L)} \circ \mathcal{E}_\theta^{(L)} \circ \cdots \circ \mathcal{E}_\theta^{(1)}(X^{(0)}).$$

Then, $\widehat{X}^{(0)}$ is projected onto the vocabulary logits.

Takeaway

Hierarchical Encoder compresses the token sequence into *chunk-level latents*, storing long-range context in a compact form.

Hierarchical Decoder decodes lower-level latents *in parallel* with local attention, conditioned on higher-level latents.

2.2 Learning Objective

We train PHOTON using standard next-token prediction along with an auxiliary regularizer that promotes consistency among its hierarchical latent streams. Specifically, we minimize the augmented language-modeling loss by including a reconstruction term:

$$\mathcal{L}_{\text{PHOTON}}(\theta; \mathcal{D}) = \mathcal{L}_{\text{token}} + \alpha \mathcal{L}_{\text{rec}}, \quad \alpha \in \mathbb{R}_+,$$

where $\mathcal{D}$ denotes the training corpus.

Next-Token Loss. Our primary objective is to maximize the autoregressive likelihood. Given the

token-level logits computed from $\widehat{X}^{(0)}$, we minimize the negative log-likelihood as follows:

$$\mathcal{L}_{\text{token}} = -\sum_{i=1}^{T-1} \log p_\theta(t_{i+1} \mid t_{1:i}).$$

This maintains PHOTON as a standard decoder-only language model at the output level.

Recursive Loss. Next-token supervision alone does not ensure that the model maintains a coherent hierarchy; intermediate latent streams may not be recoverable from the top-down pathway. We introduce an auxiliary objective that explicitly enforces hierarchical consistency by matching each encoder state to its corresponding top-down reconstruction at every level:

$$\mathcal{L}_{\text{rec}} = \sum_{l=1}^{L} \sum_{g=1}^{M_l} D(\widehat{X}^{(l-1)}_{I_g^{(l)}}, X^{(l-1)}_{I_g^{(l)}}), \qquad (1)$$

where $D(\cdot,\cdot)$ measures the dissimilarity between tensors of the same shape. This study defines D as the cosine distance $(1-\text{CosineSimilarity})$, computed for each position and averaged. By promoting accurate top-down recovery of compressed bottom-up information, this loss improves the fidelity of the multi-rate latent state. As introduced in section 3.2, it also fosters recursive consistency, enabling faster inference by allowing decoder-side reconstructions to replace costly bottom-up re-encoding during recursive generation.

2.3 Generation

2.3.1 Inference Bottlenecks in Transformers

Autoregressive Transformers face two inference bottlenecks that become more pronounced with longer contexts. During *prefill*, the model processes the entire prompt in parallel, with latency primarily determined by self-attention and feed-forward computations, making this phase mainly compute-bound. During *decoding*, generation occurs token by token, repeatedly accessing an expanding KV cache to attend to previous states. Since the KV cache grows linearly with context length and batch size, decoding often becomes memory-bound: throughput is limited by KV cache bandwidth rather than arithmetic throughput.

2.3.2 Hierarchical Generation

PHOTON mitigates sequence-length dependence by transferring global computation to low-rate latent streams and restricting token-level decoding to fixed-size local windows. During inference, each chunk is independently decoded in parallel within the same higher-level context; this process is known as hierarchical generation (HierGen).

Computation. At level l, the global encoder processes $M_l \approx T/C_{\le l}$ latent units, resulting in a prefill cost that scales as $\sum_{l=1}^{L} \mathcal{O}(M_l^2) = \sum_{l=1}^{L} \mathcal{O}((T/C_{\le l})^2)$, thereby replacing the vanilla $\mathcal{O}(T^2)$ dependence with a sum over shorter sequences. In contrast, each local decoder attends only to its chunk, with a maximum span of $R_l + C_l$, yielding $\mathcal{O}(1)$ computations per generated token related to T, which facilitates chunk-parallel decoding at a fixed window size.

KV Cache Size. The global encoder stores keys and values for M_l latent units at level l, resulting in total global KV storage of $\sum_{l=1}^{L} \mathcal{O}(M_l) = \sum_{l=1}^{L} \mathcal{O}(T/C_{\le l})$. Each local decoder, however, requires only the KV cache for the *current* chunk, constrained by $\mathcal{O}(R_l + C_l)$ at level l and is independent of T.

KV Cache Access. At level l, the global context encoder advances once per level l chunk, i.e., for M_l causal steps over a length-T continuation. Since step g addresses a prefix of length $\Theta(g)$, the cumulative global KV reads at level l scale as $\sum_{g=1}^{M_l} \mathcal{O}(g) = \mathcal{O}((T/C_{\le l})^2)$, and the total global KV read traffic is $\sum_{l=1}^{L} \mathcal{O}((T/C_{\le l})^2)$. By contrast, each local decoder reads KV only within a bounded window of size $\mathcal{O}(R_l + C_l)$; over a length-T continuation, this results in total local KV reads $\mathcal{O}(T(R_l+C_l)/C_{\le(l-1)})$, which grows only linearly in T. Therefore, for large T, global reads dominate; PHOTON reduces total decode-time KV traffic by shortening the global cached sequences at coarser levels and updating them less frequently while decoding tokens in parallel across chunks using bounded local attention.

2.3.3 Recursive Generation

Hierarchical generation confines token-level autoregression to bounded local windows; however, an encoder-consistent implementation typically requires re-encoding newly generated tokens in a bottom-up manner to refresh the hierarchical states on the encoder-side. This re-encoding triggers additional global-attention updates and increases KV-cache traffic across multiple levels. We introduce recursive generation (RecGen), a decoding schedule that eliminates the need for bottom-up

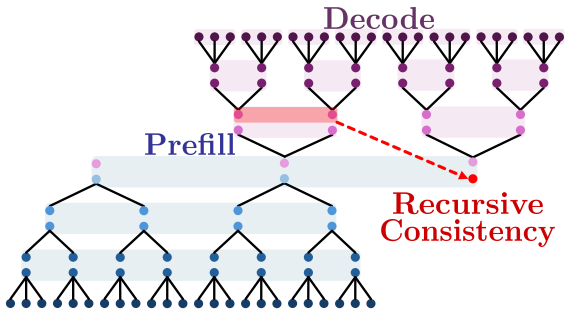


Figure 2: Recursive generation (RecGen): only the top-level KV cache grows during decoding. After a one-time hierarchical prefill, RecGen discards encoder-side caches at levels $1, \ldots, L-1$, decodes each meta-context top-down, and updates the top-level state directly from the decoder-side bottleneck reconstruction via the same chunker $\mathcal{C}_\theta^{(L)}$, eliminating bottom-up re-encoding.

re-encoding. During decoding, only the coarsest stream grows as a global state; it is updated using a summary computed directly from the decoder-side bottleneck reconstruction. We refer to the $C_{\leq L}$ tokens generated by a single top-level step as a meta-context.

Recursive Decoding. Let $\mathrm{KV}_{\leq g}^{(L)}$ denote the KV cache of the top-level causal context encoder $\mathcal{F}_\theta^{(L)}$ after processing g top-level inputs, and define its streaming update as

$$\left(X_g^{(L)}, \mathrm{KV}_{\leq g}^{(L)}\right) = \mathsf{STEP}_{\mathcal{F}_\theta^{(L)}}\left(A_g^{(L)}, \mathrm{KV}_{\leq g-1}^{(L)}\right).$$

After a one-time hierarchical prefill on the prompt, we retain $\mathrm{KV}^{(L)}$ and discard encoder-side caches at levels $1, \ldots, L-1$. At each meta context step from g to $g+1$, we (i) run the top-down decoders conditioned on $X_g^{(L)}$ to sample the next tokens and obtain the bottleneck reconstruction $\widehat{X}^{(L-1)}_{I_{g+1}^{(L)}}$, (ii) summarize it using the same chunker $\widehat{A}_{g+1}^{(L)} := \mathcal{C}_\theta^{(L)}(\widehat{X}^{(L-1)}_{I_{g+1}^{(L)}})$, and (iii) update the coarsest stream by

$$\left(X_{g+1}^{(L)}, \mathrm{KV}_{\leq g+1}^{(L)}\right) = \mathsf{STEP}_{\mathcal{F}_\theta^{(L)}}\left(\widehat{A}_{g+1}^{(L)}, \mathrm{KV}_{\leq g}^{(L)}\right).$$

Therefore, $\mathrm{KV}^{(L)}$ is the only context-growing global cache during decoding; all KV caches in the top-down decoders are local, constrained by fixed windows, and can be discarded at chunk boundaries.

Recursive Consistency and Equivalence. RecGen replaces encoder-side re-encoding with a summary computed from the decoder-side bottleneck reconstruction. A sufficient condition for exact equivalence to HierGen with bottom-up re-encoding is bottleneck *recursive consistency*, specifically $\widehat{X}^{(L-1)} = X^{(L-1)}$. Under this condition, we have $\widehat{A}_g^{(L)} = A_g^{(L)}$, indicating that both procedures apply the same top-level updates and therefore induce the same distribution over output token sequences. Equation (1) explicitly promotes this consistency, making decoder-derived summaries a reliable substitute for encoder-side re-encoding. We provide the derivation and additional theoretical results in Appendix B.

Inference Cost. The prefill remains unchanged; we run the hierarchical encoder once on the prompt. All differences arise during decoding. In HierGen, the model maintains and advances global encoder streams across all levels, resulting in cumulative KV reads $\sum_{l=1}^{L} \mathcal{O}(M_l^2) = \sum_{l=1}^{L} \mathcal{O}((T/C_{\leq l})^2)$. In contrast, RecGen retains only the top-level cache $\mathrm{KV}^{(L)}$ and performs a single global streaming update per meta-context, reducing the global KV footprint from $\sum_{l=1}^{L} \mathcal{O}(T/C_{\leq l})$ to $\mathcal{O}(T/C_{\leq L})$. As a result, global KV reads collapse to the top-level term $\mathcal{O}((T/C_{\leq L})^2)$. All KV accesses within the top-down decoders are constrained by fixed attention windows, leading to linear increases with T.

Takeaway

RecGen speeds up decoding over **HierGen** by keeping global KV only at the top level and skipping bottom-up re-encoding, halving the GPU-resident model footprint.

Memory	**HierGen**	**RecGen**
Size	$\sum_l \mathcal{O}(T/C_{\leq l})$	$\mathcal{O}(T/C_{\leq L})$
Access	$\sum_l \mathcal{O}((T/C_{\leq l})^2)$	$\mathcal{O}((T/C_{\leq L})^2)$

3 Experiments

Architecture and Training Configurations. All models are based on LLaMA Transformer architecture (Touvron et al., 2023). We compare PHOTON to vanilla Transformer and block Transformer (Ho et al., 2024). For each PHOTON configuration, we tune the baseline architectures to have approximately the same number of trainable parameters; see the details in Appendix D.2. PHOTON is trained on the Pile (Gao et al., 2020) with a context length of 2048 for one epoch. For the Vanilla and

Table 2: Inference efficiency and language modeling quality for PHOTON and Vanilla/Block Transformer baselines at matched model sizes. Memory is measured per sample (GiB), throughput in K tokens/s, and TPM in K tokens/s/GiB. Quality is evaluated by WikiText perplexity and zero-shot accuracy on HS, SciQ, and ARCe.

Models	TPM		Memory		Throughput		PPL and Zero-shot Accuracy			
	PF↑	DE↑	PF↓	DE↓	PF↑	DE↑	WikiText↓	HS↑	SciQ↑	ARCe↑
Vanilla Transformer										
600 M	3.24	7.35	0.275	0.230	0.89	1.69	22.3793	41.24	72.30	43.94
900 M	3.99	6.61	0.298	0.354	1.19	2.34	20.4102	44.72	76.10	47.73
1.2 B	1.21	2.56	0.439	0.390	0.53	1.00	19.6831	45.65	81.50	49.33
Block Transformer										
600 M	562.73	1528.71	0.044	0.031	24.76	47.39	27.2478	37.01	70.30	42.63
900 M	485.74	1386.05	0.054	0.038	26.23	52.67	26.3706	37.06	71.20	43.43
1.2 B	205.00	540.20	0.070	0.051	14.35	27.55	22.8429	41.74	74.90	45.83
PHOTON										
600 M	1262.58	3062.17	0.031	0.023	39.14	70.43	29.9055	35.49	67.70	42.97
900 M	1141.62	2797.04	0.037	0.027	42.24	75.52	26.2325	38.24	69.00	44.32
1.2 B	543.86	1216.67	0.044	0.036	23.93	43.80	23.7863	40.70	69.30	46.25

Block Transformer baselines, we align the training compute budget with the corresponding PHOTON model in terms of total FLOPs. We fix the training context length at 2048 tokens to enable a direct comparison with Block Transformer; behavior beyond the training window is examined separately by extrapolating the context to 4096 and 8192 tokens using YaRN, where recursive consistency remains stable, as shown in Appendix C.4. All experiments are conducted on NVIDIA DGX H200 GPUs. Additional implementation details are included in Appendix D.

Evaluation. We evaluate (i) inference efficiency and (ii) language modeling quality. Following the empirical protocol of Ho et al. (2024), we measure the per-sample KV-cache memory footprint and throughput (K tokens/s) under two complementary serving regimes: *prefill-heavy* (PF), with a long prompt and short continuation (2048 input/128 output), and *decode-heavy* (DE), with a short prompt and long continuation (128 input/2048 output). To summarize memory efficiency in multi-query serving, we also report throughput-per-memory (TPM), defined as TPM=Throughput/Memory in K tokens/s/GiB. For quality, we report WikiText perplexity (PPL) (Merity et al., 2017) and zero-shot accuracy on HellaSwag (HS) (Zellers et al., 2019), SciQ (Welbl et al., 2017), and ARC-Easy (ARCe) (Clark et al., 2018).

3.1 Main Results

Throughput-per-Memory Gains. PHOTON primarily enhances inference efficiency by reducing the KV cache footprint and associated memory traffic during serving. This section adopts a representative two-level hierarchy ($L = 2$) with chunk lengths $C_1 = 4$ and $C_2 = 4$; Appendix C.2 varies these chunk lengths and demonstrates that PHOTON maintains the most favorable TPM-quality trade-off across context-length settings. In the main results, we set the recursive reconstruction weight to $\alpha = 0.0$ to isolate gains from the hierarchical architecture and bounded local parallel decoding of PHOTON, which differ from the Block Transformer architecture. Appendix C.1 further studies α and shows that a moderate value, around $\alpha \approx 0.3$, maximizes downstream zero-shot accuracy, supporting the effectiveness of the proposed recursive regularization.

Table 2 presents the KV cache memory and throughput per-sample under PF and DE settings. Across model scales, PHOTON reduces KV-cache memory while increasing throughput, resulting in substantial gains in TPM. In the PF regime, PHOTON reduces KV-cache memory per sample by $8.9\times$ (600M), $8.1\times$ (900M), and $10.0\times$ (1.2B), while enhancing throughput by $44.0\times$, $35.5\times$, and $45.2\times$, respectively. In the DE regime, PHOTON reduces KV memory by $10.0\times$ (600M), $13.1\times$ (900M), and $10.8\times$ (1.2B), and increases throughput by $41.7\times$, $32.3\times$, and $43.8\times$. As a result, PHOTON achieves substantially higher TPM than a vanilla Transformer in both regimes; for the 1.2B model in DE, TPM increases from 2.56 to 1216.67 K tok/s/GiB, corresponding to a $475\times$ improvement. Overall, these results support our central

Table 3: Recursive loss after completing training across different sequence lengths.

Length	**2048**	**4096**	**8192**
Recursive loss ↓	0.0740	0.0891	0.1041

Table 4: Inference efficiency gains from RecGen over standard HierGen for 600M PHOTON model. Memory usage per sample is reported in GiB, throughput, and TPM under PF and DE settings.

Metric	Setting	Generation	
		HierGen	**RecGen**
TPM	PF↑	1262.58	**2828.06**
	DE↑	3062.17	**13642.50**
Memory	PF↓	0.031	**0.031**
	DE↓	0.023	**0.012**
Throughput	PF↑	39.14	**87.67**
	DE↑	70.43	**163.71**

claim that maintaining a hierarchical state reduces decode-time KV traffic and enables more memory-efficient generation.

TPM vs. Language Modeling Quality. Figure 3 shows the trade-off between TPM and language modeling quality. PHOTON achieves significantly higher TPM with only moderate degradation in WikiText PPL. At 600M, TPM increases from 3.24 K tokens/s/GiB to 1262.58 K tokens/s/GiB in the PF regime, and from 7.35 to 3062.17 in DE regime, while PPL slightly increases from 22.38 to 29.91. For the 1.2B model, TPM rises from 1.21 K tokens/s/GiB to 543.86 K tokens/s/GiB in the PF regime and from 2.56 K tokens/s/GiB to 1216.67 K tokens/s/GiB in DE regime, while PPL moderately increases from 19.68 to 23.79. As a result, PHOTON achieves significantly higher TPM than Vanilla models in both regimes. Compared to the Block Transformer, PHOTON achieves higher TPM at both model sizes across both regimes; dominating the Block Transformer concerning the Pareto frontier in Figure 3. This indicates that PHOTON provides a superior TPM-quality operating point, and the saved memory bandwidth can be traded back for quality through test-time scaling within a fixed memory budget.

3.2 Recursive Generation

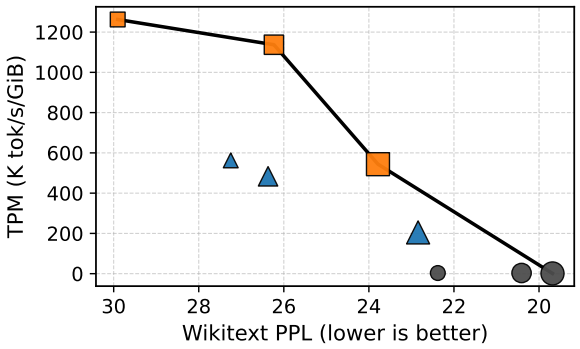
We evaluate RecGen as a drop-in decoding procedure for the 600M PHOTON model in both the PF and DE regimes, using HierGen, the standard procedure that re-encodes newly generated tokens bottom-up at every step, as the reference. As shown in Table 4, RecGen significantly improves inference efficiency while leaving generation behavior nearly unchanged. We attribute this improvement to the steady decrease of the recursive loss throughout training to a small absolute value, as illustrated in

Figure 5a of Appendix C.3: the decoder-side bottleneck reconstruction closely tracks the encoder-side state that RecGen replaces. A natural concern is whether this recursive consistency deteriorates as the context length increases; if it does, RecGen would lose fidelity to HierGen on longer inputs. To address this directly, rather than through post-hoc positional extrapolation, we also pre-train PHOTON from scratch at context lengths of $4{,}096$ and $8{,}192$ tokens. As reported in Table 3, the converged recursive loss increases only slightly with the training context length, rising from 0.074 at $2{,}048$ tokens to 0.089 at $4{,}096$, and 0.104 at $8{,}192$, remaining small in absolute terms. This indicates that recursive consistency, and therefore the HierGen equivalence on which RecGen is based, continues to hold even when the global stream is four times longer than the default training length.

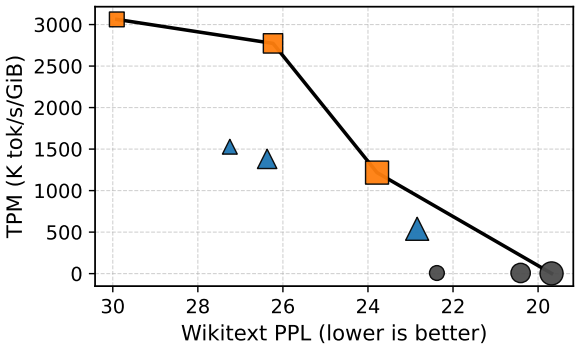
The efficiency gains are substantial. In the PF regime, RecGen raises throughput from 39.14 to 87.67 K tok/s, resulting in a $\approx 2.2\times$ improvement at a fixed memory footprint, translating to a $\approx 2.2\times$ TPM gain. In the DE regime, throughput increases from 70.43 to 163.71 K tok/s, marking a $\approx 2.3\times$ improvement, while per-sample memory decreases from 0.023 to 0.012 GiB, reflecting a $\approx 1.9\times$ reduction and totaling a $\approx 4.5\times$ TPM gain. Compounded with the Vanilla 600M Transformer baseline reported in Table 2—whose DE-regime TPM is only 7.35 K tok/s/GiB—RecGen, equipped with PHOTON, achieves an end-to-end TPM advantage of up to $\approx 1{,}856\times$ over the Vanilla baseline. In summary, RecGen transforms global KV-cache traffic into bounded local computation, making it particularly well suited for multi-query workloads under a fixed GPU-memory budget.

3.3 Converting TPM Capacity into Accuracy via Self-Consistency

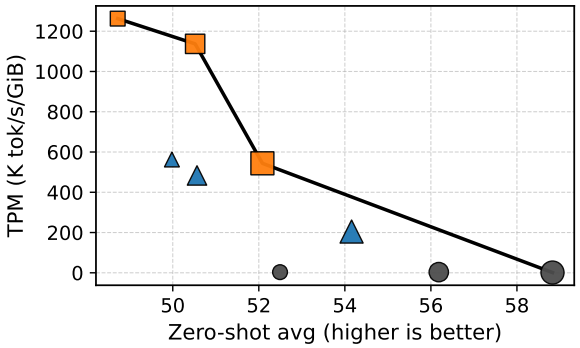
The per-stream KV-cache footprint of PHOTON is several orders of magnitude smaller than that of a vanilla Transformer at comparable quality, as shown in Table 2. A natural application of this



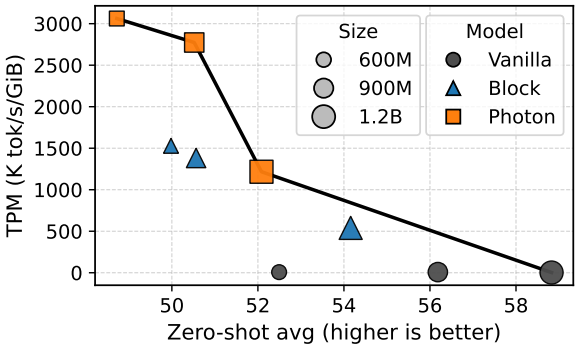
(a) TPM vs. WikiText PPL in prefill-heavy (PF).



(b) TPM vs. WikiText PPL in decode-heavy (DE).



(c) TPM vs. zero-shot accuracy (PF)



(d) TPM vs. zero-shot accuracy (DE)

Figure 3: TPM–quality trade-offs under PF and DE regimes. Panels (a,b) plot TPM against WikiText perplexity, and panels (c,d) plot TPM against average zero-shot accuracy over HS, SciQ, and ARCe. The dotted line denotes the Pareto frontier in each panel. Across both regimes and all metrics, PHOTON consistently yields a more favorable TPM–quality frontier than Vanilla and Block Transformers.

available TPM budget is *self-consistency*: we decode N lightly perturbed copies of the same question in parallel using the PHOTON model and aggregate the results. Even with N=9, the aggregated KV-cache footprint remains within the budget required by a single vanilla Transformer stream on our hardware.

For each question q, we synthesize $N-1$ perturbed variants $\tilde{q}_1, \ldots, \tilde{q}_{N-1}$ using one of the strategies described below. We then decode the N queries $q, \tilde{q}_1, \ldots, \tilde{q}_{N-1}$ in parallel, using the same PHOTON model to predict the answer $\hat{y} = \arg\max_{y \in \mathcal{Y}} \max_{0 \le i \le N-1} \log p_\theta(y \mid q_i)$, where $q_0 = q$ and $\mathcal{Y}$ denote the candidate set. We utilize $N = 9$ throughout and report zero-shot accuracy on ARC-Easy and SciQ.

Paraphrase rewrites q using GPT-4 while preserving its meaning. *Token-Insert* samples a token uniformly from q and inserts it at a random position within q. *Span-Insert* selects a contiguous span of up to 15 tokens from q and inserts it at a random position in q. The latter two strategies do not use an external model, so any resulting improvements cannot be attributed to the quality of GPT-4.

Table 5: **PHOTON's TPM headroom converts into zero-shot accuracy gains via N=9 self-consistency.** Zero-shot accuracy on ARC-Easy and SciQ under single-query decoding versus three query-perturbation strategies, each aggregated over N=9 parallel decodes through the same PHOTON model. Best per column in bold.

Method	ARCe	SciQ
Single-query baseline	0.4621	0.6910
Paraphrase, N=9	**0.5741**	0.7660
Token-Insert, N=9	0.5244	0.7790
Span-Insert, N=9	0.5126	**0.7900**

4 Related Work

PHOTON is associated with (i) hierarchical and multi-scale Transformers, (ii) inference efficiency in KV terms, and (iii) global-local modeling in tokenizer-free language models.

Hierarchical and Multi-scale Transformers. A substantial body of work reduces the effective length of global attention by introducing intermediate representations. For example, hierarchical encoders for long documents (Pappagari et al., 2019), patch-based or segment-based Transformers (Han et al., 2021), and down-sampled or up-sampled architectures such as Funnel and Hourglass (Dai et al., 2020; Nawrot et al., 2022; Zhu and Soricut, 2021). These approaches primarily aim for better representations, improved training stability, or reduced training costs. However, during inference, they typically maintain token-level autoregressive decoding, resulting in a KV cache that grows linearly with context length. In contrast, PHOTON treats hierarchy as an inference primitive; it maintains persistent multi-rate latent streams as global state and confines token-level computation to strictly bounded local refinement.

KV-Efficient Inference. Previous work accelerates long-context inference primarily by modifying attention mechanisms, such as sparse or windowed patterns (Child et al., 2019; Beltagy et al., 2020), and by dynamically retaining a subset of tokens during training or inference (Nawrot et al., 2023; Fu et al., 2025). Although effective, these methods maintain a single token-level timeline: decoding occurs token by token, which limits inference bandwidth due to repeated KV-cache reads and writes. PHOTON addresses the same bottleneck but employs a different design principle: it reduces KV traffic through a persistent hierarchical state. Specifically, PHOTON factorizes generation into an encoder-decoder hierarchy that represents global context as low-rate latent streams and produces tokens with strictly bounded local causal decoders. This structure enables parallel decoding of independent chunks conditioned on higher-level latents, thereby shrinking the globally cached sequence and reducing the need for global updates.

Tokenizer-free Models. Tokenizer-free and byte-level language models often adopt global-local hierarchies to model long byte streams tractably, as seen in MEGABYTE (Yu et al., 2023), SpaceByte (Slagle, 2024), and learned segmentation approaches (Zakershahrak and Ghodratnama, 2025; Fleshman and Van Durme, 2023; Mujika, 2023). In these models, the hierarchy induces subword-like units from bytes for a single autoregressive stream: a global module models patches or segments, while a local module reconstructs bytes or characters without maintaining a persistent multi-level state during decoding. PHOTON targets a different objective. Instead of learning subword units from bytes, it introduces higher-level contextual latents to reduce redundant computation and accelerate inference for subword-token language models. This results in a compact encoder-decoder design in which the encoder maintains a persistent coarse context summary, while local decoders use it to refine token-level generation, thereby reducing inference-time memory traffic.

5 Conclusion

We presented PHOTON, a hierarchical autoregressive language model that replaces horizontal *token-by-token scanning* in Transformers with *vertical* multi-resolution context scanning. PHOTON builds a hierarchy of latent streams through (i) a bottom-up encoder that compresses the token sequence into low-rate contextual states and (ii) a top-down decoder stack that reconstructs progressively finer representations using local autoregressive modules with bounded attention, enabling parallel decoding across independent contexts. We further introduced *recursive generation*, updating only the coarsest latent stream during decoding and avoiding bottom-up re-encoding, thus reducing KV-cache growth and memory traffic. Experiments demonstrate that PHOTON consistently improves the TPM–quality trade-off over strong vanilla and Block Transformer baselines, achieving up to $10^3\times$ higher TPM. These results suggest that the persistent hierarchical state and local reconstruction provide a promising direction for scalable, memory-efficient language generation.

Limitations

This work has several limitations. First, we train and evaluate PHOTON using a single pretraining corpus and a relatively small set of downstream benchmarks; broader coverage is needed to confirm whether the observed trends generalize across different data mixtures and task families. Second, our largest model contains 1.2 billion parameters, and we have yet to characterize how the efficiency-quality trade-off of PHOTON performs at larger scales. Third, while we report representative results for certain architectural configurations, we do not include a comprehensive sensitivity analysis of key design and training choices, such as chunk sizes and converter widths. A more exhaustive ablation study would help isolate the contributions of each component and clarify which settings are most robust.

A limitation is that our evaluation primarily focuses on efficiency and predictive quality, rather than robustness, calibration, or safety-oriented behaviors. Specifically, because PHOTON is designed to increase throughput and reduce memory costs for language generation, these efficiency gains could lower the risk of generating misleading, harmful, or otherwise inappropriate text if the model is deployed without appropriate safeguards. Our experiments do not assess downstream misuse risks, toxicity, bias, or factuality; these areas remain important directions for future work.

We also note that our pretraining and evaluation rely on existing large-scale corpora and benchmarks rather than newly collected human-subject data. Such data may contain noisy, biased, offensive, or personally identifiable content inherited from their original sources. In this work, we do not conduct a new manual audit or anonymization procedure for these datasets; instead, we rely on their standard research use and do not collect, annotate, or redistribute any new personal data. Future work should include a more systematic analysis of data quality, privacy-sensitive content, and harmful content, especially in deployment-oriented settings.

Acknowledgments

This work was partially supported by JST BOOST, Japan (Grant No. JPMJBY24D0), and RIKEN Center for AIP.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*.

Noam Chomsky. 2002. *Syntactic structures*. Walter de Gruyter.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *ArXiv*, abs/1803.05457.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. 2020. Funnel-transformer: Filtering out sequential redundancy for efficient language processing. In *Advances in Neural Information Processing Systems*.

William Fleshman and Benjamin Van Durme. 2023. Toucan: Token-aware character level language modeling. *arXiv preprint arXiv:2311.08620*.

Zhen Fu and 1 others. 2025. Sliding window attention training for efficient large-context language models. *arXiv preprint arXiv:2502.18845*.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, and 1 others. 2020. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.

Barbara J Grosz, Aravind Joshi, and Scott Weinstein. 1995. Centering: A framework for modeling the local coherence of discourse. *Computational linguistics*, 21(2):203–225.

Barbara J Grosz and Candace L Sidner. 1986. Attention, intentions, and the structure of discourse. *Computational linguistics*, 12(3):175–204.

Morris Halle. 1973. Prolegomena to a theory of word formation. *Linguistic inquiry*, 4(1):3–16.

Kai Han, An Xiao, Enhua Wu, Jianyuan Guo, Chunjing Xu, and Yunhe Wang. 2021. Transformer in transformer. In *Advances in Neural Information Processing Systems*.

Marc D Hauser, Noam Chomsky, and W Tecumseh Fitch. 2002. The faculty of language: what is it, who has it, and how did it evolve? *science*, 298(5598):1569–1579.

Namgyu Ho, Sangmin Bae, Taehyeon Kim, Hyunjik Jo, Yireun Kim, Tal Schuster, Adam Fisch, James Thorne, and Se-Young Yun. 2024. Block transformer: Global-to-local language modeling for fast inference. In *Advances in Neural Information Processing Systems*.

Joachim Lambek. 1958. The mathematics of sentence structure. *The American Mathematical Monthly*, 65(3):154–170.

William C Mann and Sandra A Thompson. 1988. Rhetorical structure theory: Toward a functional theory of text organization. *Text-interdisciplinary Journal for the Study of Discourse*, 8(3):243–281.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *International Conference on Learning Representations*.

Asier Mujika. 2023. Hierarchical attention encoder–decoder architectures for long-range sequence modeling. *arXiv preprint arXiv:2306.01070*.

Piotr Nawrot, Jan Chorowski, Adrian Łańcucki, and Edoardo Maria Ponti. 2023. Efficient transformers with dynamic token pooling. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*.

Piotr Nawrot, Szymon Tworkowski, Michał Tyrolski, Łukasz Kaiser, Yuhuai Wu, Christian Szegedy, and Henryk Michalewski. 2022. Hierarchical transformers are more efficient language models. In *Findings of the Association for Computational Linguistics: NAACL*.

Raghavendra Pappagari, Piotr Zelasko, Jesús Villalba, Yishay Carmiel, and Najim Dehak. 2019. Hierarchical transformers for long document classification. In *2019 IEEE automatic speech recognition and understanding workshop (ASRU)*.

Kevin Slagle. 2024. Spacebyte: Towards deleting tokenization from large language modeling. In *Advances in Neural Information Processing Systems*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

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

Johannes Welbl, Nelson F. Liu, and Matt Gardner. 2017. Crowdsourcing multiple choice science questions. *ArXiv*, abs/1707.06209.

Lili Yu, Dániel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. 2023. Megabyte: Predicting million-byte sequences with multiscale transformers. In *Advances in Neural Information Processing Systems*.

Mehrdad Zakershahrak and Samira Ghodratnama. 2025. H-net++: Hierarchical dynamic chunking for tokenizer-free language modelling in morphologically-rich languages. *arXiv preprint arXiv:2508.05628*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

Zhenhai Zhu and Radu Soricut. 2021. H-transformer-1d: Fast one-dimensional hierarchical attention for sequences. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics*.

A Formal Notation

A.1 Chunk Index Sets and Causal Masks

This appendix collects the formal definitions that were deferred from Section 2 to maintain the conciseness of the main text.

Chunk index sets. For each level $l \in [L]$ and each chunk index $g \in [M_l]$, the set of level-$(l-1)$ positions that form the g-th level-l chunk is

$$I_g^{(l)} := \{(g-1)C_l + i \mid i \in [C_l]\} \subseteq [M_{l-1}],$$

and the corresponding level-$(l-1)$ subtensor is

$$X^{(l-1)}_{I_g^{(l)}} := \left[X^{(l-1)}_{(g-1)C_l+1}, \ldots, X^{(l-1)}_{gC_l}\right].$$

The sets $\{I_g^{(l)}\}_{g=1}^{M_l}$ partition $[M_{l-1}]$ into contiguous blocks. Because $C_l \geq 1$, $M_l \leq M_{l-1}$, and therefore $M_0 \geq M_1 \geq \cdots \geq M_L$.

Local causal mask. Within a chunk, the local decoder $\mathcal{G}_\theta^{(l)}$ applies a standard causal mask

$$\mathcal{M}^{(l-1)}_{R_l,j} \in \{0, -\infty\}^{(R_l+j-1)\times(R_l+j-1)},$$

to the concatenation $[U_{g-1}^{(l)}; \widehat{X}^{(l-1)}_{I_g^{(l)},<j}]$. Under this mask, position j attends only to the R_l converter outputs and to the $j-1$ preceding positions within the same chunk; there is no cross-chunk attention at level $l-1$.

Meta-context indices. For the coarsest level, we denote $B := C_{\leq L} = \prod_{k=1}^{L} C_k$ as the meta-context (i.e., top-level chunk) length in tokens, and

$$J_g := \{(g-1)B + 1, \ldots, gB\}, \quad g \in [G],$$

for the token indices of the g-th meta-context, where $G = T/B$.

B Additional Theoretical Results

B.1 Preliminaries

We use the notation of Appendix A.1, in particular the meta-context length $B = C_{\leq L} = \prod_{k=1}^{L} C_k$ and the meta-context index sets $\{J_g\}_{g=1}^{G}$ with $G = T/B$. Throughout, we focus on the coarsest causal context encoder $\mathcal{F}_\theta^{(L)}$ and its KV cache. Let $\mathrm{KV}^{(L)}_{\leq g}$ denote the KV cache after processing g top-level inputs. We express a one-step streaming update by a deterministic operator:

$$\left(X_g^{(L)}, \mathrm{KV}^{(L)}_{\leq g}\right) = \mathsf{STEP}_{\mathcal{F}_\theta^{(L)}}\left(A_g^{(L)}, \mathrm{KV}^{(L)}_{\leq g-1}\right), \tag{2}$$

where $A_g^{(L)} \in \mathbb{R}^{D_L}$ is the g-th top-level input summary.

Assumption B.1 (Deterministic Streaming Update)**.** *The operator* $\mathsf{STEP}_{\mathcal{F}_\theta^{(L)}}$ *in Equation* (2) *is deterministic given its arguments.*

Assumption B.1 holds for standard causal Transformers with KV caching: given an input vector and a cache, the next hidden state and the updated cache are uniquely determined. As an interesting direction for future work, one could relax this assumption by allowing the model to output context *stochastically*, i.e., replacing $\mathsf{STEP}_{\mathcal{F}_\theta^{(L)}}$ with a conditional distribution over updates—potentially enabling richer uncertainty-aware streaming and alternative decoding schedules.

B.2 Two Decoding Procedures

We compare two inference procedures that use the same top-down decoder stack $\{\mathcal{D}_\theta^{(l)}\}_{l=1}^{L}$ and differ only in their construction of the subsequent top-level input.

Definition B.2 (Hierarchical Generation (HierGen))**.** *Fix a prompt and perform a one-time hierarchical prefill to initialize all encoder-side states and caches. During decoding, at each meta context step* $g \to g+1$*, hierarchical generation:*

1. *samples the next meta context tokens* $t_{J_{g+1}}$ *by running the top-down decoders conditioned on the current top-level state* $X_g^{(L)}$*;*
2. *re-encodes the newly generated tokens bottom-up to obtain the encoder-side bottleneck block* $X^{(L-1)}_{I_{g+1}^{(L)}}$ *and its top-level input summary*
$$A_{g+1}^{(L)} := \mathcal{C}_\theta^{(L)}\big(X^{(L-1)}_{I_{g+1}^{(L)}}\big);$$
3. *advances the top-level stream via Equation* (2) *with* $A_{g+1}^{(L)}$*.*

Let p_θ^{Hier} *denote the induced distribution over generated continuations.*

Definition B.3 (Recursive Generation (RecGen))**.** *Fix a prompt and perform a one-time hierarchical prefill, retaining* only *the top-level cache* $\mathrm{KV}^{(L)}$ *thereafter. During decoding, at each meta context step* $g \to g+1$*, RecGen:*

1. *samples the next meta context tokens* $t_{J_{g+1}}$ *by running the same top-down decoders conditioned on* $X_g^{(L)}$*, and simultaneously obtains the decoder-side bottleneck reconstruction* $\widehat{X}^{(L-1)}_{I_{g+1}^{(L)}}$ *produced by the top-down stack;*

2. *forms the next top-level input summary directly from the reconstruction:*

$$\widehat{A}^{(L)}_{g+1} := \mathcal{C}^{(L)}_{\theta}\big(\widehat{X}^{(L-1)}_{I^{(L)}_{g+1}}\big);$$

3. *advances the top-level stream via Equation (2) with* $\widehat{A}^{(L)}_{g+1}$.

Let $p^{\mathrm{Rec}}_{\theta}$ *denote the induced distribution over generated continuations.*

B.3 Recursive Consistency and Exactness

Definition B.4 (Recursive consistency)**.** *We say that* recursive consistency *holds if, for every meta context* $g \in [G]$*, the reconstruction at the decoder-side bottleneck matches the encoder-side bottleneck state:*

$$\widehat{X}^{(L-1)}_{I^{(L)}_g} = X^{(L-1)}_{I^{(L)}_g}. \tag{3}$$

From Definition B.4, the reconstruction at the decoder-side bottleneck is equal to the state at the encoder-side bottleneck for each meta-context. Since the top-level input summary is computed deterministically from this bottleneck representation, it follows that the top-level inputs obtained through re-encoding and recursion must coincide, yielding the following Lemma B.5.

Lemma B.5. *Under Definition B.4, the top-level inputs produced by re-encoding and recursion agree for all* $g \in [G]$*:*

$$\widehat{A}^{(L)}_{g} = A^{(L)}_{g}. \tag{4}$$

Proof. By Equation (3) and the definition of the chunker $\mathcal{C}^{(L)}_{\theta}$,

$$\widehat{A}^{(L)}_{g} = \mathcal{C}^{(L)}_{\theta}\big(\widehat{X}^{(L-1)}_{I^{(L)}_g}\big) = \mathcal{C}^{(L)}_{\theta}\big(X^{(L-1)}_{I^{(L)}_g}\big) = A^{(L)}_{g},$$

which is Equation (4). □

Lemma B.5 demonstrates that recursion yields the same top-level inputs as bottom-up re-encoding at each meta-context step. Combined with Assumption B.1, the resulting top-level hidden states and KV caches evolve identically under recursive and exact hierarchical generation. Consequently, both procedures induce the same conditional distribution at each decoding step, directly implying Theorem B.6.

Theorem B.6. *Assume Assumption B.1 and bottleneck recursive consistency (Definition B.4). Then RecGen (Definition B.3) induces the same distribution over output token sequences as exact hierarchical generation (Definition B.2):*

$$p^{\mathrm{Rec}}_{\theta} = p^{\mathrm{Hier}}_{\theta}.$$

Proof. We couple the two procedures by using the same random draws for token sampling whenever their conditional distributions match. We prove by induction on g that the top-level state and cache coincide for all $g \in \{0, \ldots, G\}$

$$\Big(X^{(L)}_{g}, \mathrm{KV}^{(L)}_{\le g}\Big)_{\mathrm{Rec}} = \Big(X^{(L)}_{g}, \mathrm{KV}^{(L)}_{\le g}\Big)_{\mathrm{Hier}}. \tag{5}$$

They coincide at $g = 0$ after the shared prompt prefill. Assume Equation (5) holds at step g. Conditioned on the identical top-level state $X^{(L)}_{g}$, both procedures run the same top-down decoder stack with the same causal masking, thus defining the same conditional distribution over the next meta context $t_{J_{g+1}}$. Using the coupling, both procedures sample the same realization of $t_{J_{g+1}}$.

By Lemma B.5, the top-level inputs then agree: $\widehat{A}^{(L)}_{g+1} = A^{(L)}_{g+1}$. Finally, by Assumption B.1 and the shared cache $\mathrm{KV}^{(L)}_{\le g}$, the deterministic update Equation (2) yields identical $\big(X^{(L)}_{g+1}, \mathrm{KV}^{(L)}_{\le g+1}\big)$. Thus, Equation (5) holds for $g+1$. By induction, the coupled procedures generate identical token sequences almost surely, implying that their induced distributions coincide. □

Theorem B.6 shows that exactness requires equality only at the bottleneck interface $L \to L{-}1$. No assumptions are needed about lower-level reconstructions to establish equality of the *output* distribution.

C Additional Experiments

C.1 Ablations over Strength of Recursive Loss

We ablate the hyperparameter α and evaluate its impact while keeping all other settings fixed, including the 4×4 block configuration. Figure 4 summarizes the results in terms of WikiText perplexity (PPL) and *Average Zero-shot Accuracy*, with the latter computed as the mean accuracy over ARC-Easy, HellaSwag, and SciQ. WikiText perplexity remains largely stable for $\alpha \le 0.3$ and increases at $\alpha = 0.4$, indicating that an excessively strong recursive loss can negatively impact language modeling. Average zero-shot accuracy exhibits a non-monotonic trend: it decreases from $\alpha = 0.0$ to

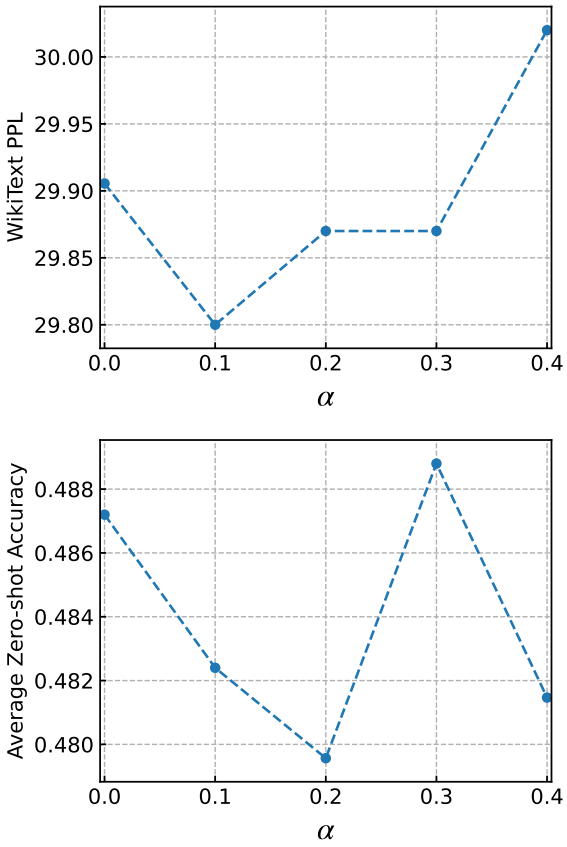


Figure 4: Moderate recursive regularization ($\alpha \approx 0.3$) is best for zero-shot accuracy, while perplexity is largely insensitive to α below 0.4. *Top:* WikiText perplexity (lower is better). *Bottom:* average zero-shot accuracy over ARCe, HellaSwag, and SciQ (higher is better).

$\alpha = 0.2$, then recovers and peaks at $\alpha = 0.3$, before declining again at $\alpha = 0.4$. These results suggest that a moderate nonzero α can provide a slight downstream gain over $\alpha = 0.0$, but the benefit is sensitive to tuning and does not improve monotonically with α.

C.2 Ablations over Chunk Lengths

Chunk lengths $\{C_1, C_2\}$ directly determine the compression ratio of PHOTON's hierarchical state and, the *effective context length* processed by the global (coarse) encoders. With a two-level hierarchy ($L = 2$) and a fixed token length T, the top-level sequence length is $M_2 = T/(C_1C_2)$: smaller chunks yield longer latent streams and more frequent global updates, which can improve modeling fidelity while increasing global KV-cache traffic. To quantify this trade-off, we adhere to the primary setup outlined in Section 3, keeping all hyperparameters fixed (600M model, Pile pretraining, training context length of 2048, and the same evaluation protocol). We compare a finer hierarchy (C_1=2, C_2=2; 2×2) to the default configuration (C_1=4, C_2=4; 4×4).

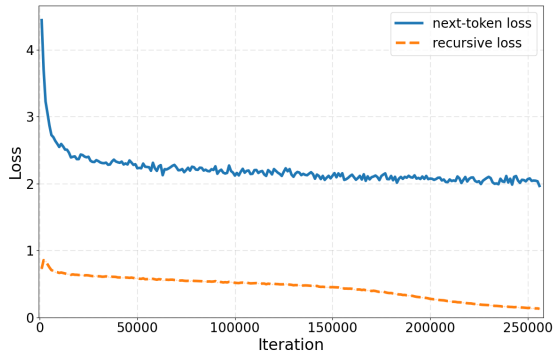
Table 6: Ablation study of chunk length. Memory usage per sample is reported in GiB, throughput, and TPM under PF and DE settings.

Metric	Setting	Chunk Length (C_1, C_2)	
		$(2, 2)$	$(4, 4)$
TPM	PF↑	175.45	1262.58
	DE↑	418.30	3062.17
Memory	PF↓	0.066	0.031
	DE↓	0.053	0.023
Throughput	PF↑	11.58	39.14
	DE↑	22.17	70.43
PPL ↓	-	24.28	29.91
Accuracy↑	-	52.01	48.72

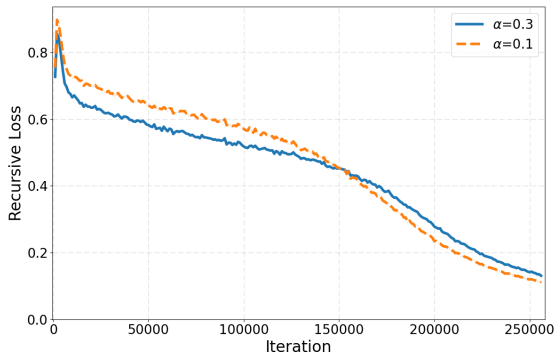
Table 6 shows that 2×2 substantially improves quality, reducing WikiText PPL from 29.91 to 24.28 and increasing average zero-shot accuracy from 48.72 to 52.01, while incurring the expected cost in efficiency: under PF, throughput decreases from 39.14 to 11.58 K tok/s, and KV memory rises from 0.031 to 0.066 GiB (TPM drops from 1262.58 to 175.45); under DE, throughput decreases from 70.43 to 22.17 K tok/s, and memory increases from 0.023 to 0.053 GiB (TPM drops from 3062.17 to 418.30). These results highlight $\{C_1, C_2\}$ as a simple lever for navigating the TPM–quality Pareto frontier: smaller chunks mitigate the compression bottleneck and increase the temporal resolution of the global latent stream, thereby improving perplexity and downstream accuracy, while larger chunks maximize throughput by minimizing global-state growth and KV traffic. Notably, even the quality-oriented 2×2 configuration remains significantly more memory-efficient than a vanilla Transformer at the same scale, as shown in Table 2, while also narrowing the quality gap. This suggests that PHOTON can flexibly convert some of its efficiency gains back into modeling performance without deviating from the Pareto frontier.

C.3 Recursive Consistency

Recursive Loss vs. Next-Token Loss. RecGen replaces bottom-up re-encoding with summaries computed from the decoder-side bottleneck reconstruction, making *recursive consistency* between encoder states and top-down reconstructions crit-



(a) The recursive loss decreases steadily during training and converges to a small absolute value, indicating that the top-down reconstruction closely tracks the encoder-side bottleneck. Token-level NLL and the cosine-distance-based recursive loss are not directly comparable in units; the key observation is the smallness and steady decrease of the recursive term.



(b) Recursive loss continues to decrease throughout training for both $\alpha \in \{0.1, 0.3\}$, suggesting that longer training can further tighten recursive consistency. A smaller α yields a slightly lower loss late in training, consistent with reduced regularization pressure on the hierarchical state.

ical. A practical indicator of this consistency is the recursive loss in Eq. 1. Figure 5a reports the training dynamics for the 600M PHOTON model with $\alpha = 0.3$ ($(C_1, C_2) = (4, 4)$): the recursive loss steadily decreases and reaches a small absolute value by the end of training. Although the token-level negative log-likelihood and the cosine-distance-based recursive loss are not directly comparable in scale or units, the diminishing recursive term suggests that the top-down pathway can closely approximate the encoder-side bottleneck state. This provides empirical support that RecGen should only minimally perturb generation behavior when the bottleneck mismatch is sufficiently small. We also find that the recursive loss decreases even when $\alpha = 0$, indicating that the hierarchical architecture itself may encourage self-consistent summaries; a more detailed characterization of how residual mismatch translates into long-horizon distributional drift under RecGen is left for future work

Evaluating Likelihood of RecGen. RecGen updates the global state using model-generated reconstructions instead of ground-truth tokens, making exact likelihood evaluation under the *RecGen-induced* process difficult to incorporate into standard teacher-forcing pipelines. As a result, computing perplexity and zero-shot accuracy *under RecGen* in a fully consistent likelihood-based protocol is non-trivial. Therefore, we report perplexity and zero-shot results under standard teacher-forced execution, i.e., HierGen and reserve a more rigorous likelihood-based evaluation of RecGen for future work.

Dependence on Reconstruction Weight α. Figure 5b compares the recursive-loss trajectories of the 600M PHOTON model with $\alpha \in \{0.3, 0.1\}$. A smaller weight results in a slightly lower recursive loss late in training, and in both cases, the loss continues to decrease until the end of optimization. These trends suggest that recursive consistency can be improved not only by tuning the auxiliary objective but also by extending the training duration, which may further narrow the gap between RecGen and the encoder-consistent decoding procedure.

C.4 Long-Context Behaviour under Context-Window Extension

Our main experiments fix the maximum context length to 2048 for a fair, directly comparable evaluation against Block Transformer (Ho et al., 2024). A natural question is whether the recursive-consistency mechanism continues to hold as the context length increases. Since retraining from scratch with substantially longer contexts is expensive, we use the 600M PHOTON model trained with a recursive-loss coefficient $\alpha = 0.25$ and *extrapolate* its context window using YaRN (**?**), *without* any YaRN fine-tuning. We apply YaRN only to the encoder-side context encoders because the token sequence lengths processed by the local decoders remain unchanged as the global context grows.

We evaluate both the next-token loss and the recursive loss on long-document samples from The Pile. Table 7 reports the losses across context lengths of 2048, 4096, and 8192 tokens. The recursive loss remains small (0.133–0.147) and does not blow up over the largest range we could test; in fact, the next-token loss *decreases* as the context

Table 7: **Long-context behaviour via YaRN-extrapolated 600M PHOTON.** Next-token and recursive losses on long-document Pile samples at context lengths of 2048, 4096, and 8192. YaRN is applied only to the encoder-side components; no additional fine-tuning is performed. The recursive loss remains stable and shows no sign of rapid divergence.

Loss	2048 (original)	4096	8192
Next-token loss ↓	1.978	1.933	1.862
Recursive loss ↓	0.133	0.145	0.147

grows, consistent with having more available context. These observations provide no evidence of catastrophic mismatch under longer contexts and support the idea that recursive consistency, the condition under which RecGen is exactly equivalent to HierGen (Theorem B.6), remains approximately satisfied after YaRN extrapolation.

C.5 Single-Batch Latency

While throughput and TPM are our primary metrics for serving efficiency, we also report single-batch latency because it directly shows the delay before the first token is produced, without any benefit from batching or concurrency. As shown in Table 8, PHOTON introduces little overhead from its hierarchical design and consistently achieves lower latency than both the vanilla Transformer and the Block Transformer across all tested model sizes under both DE and PF settings

C.6 More Larger Models

Table 9 shows the larger-scale experiments that were conducted under a modified setup relative to Table 2. Specifically, we trained the Qwen3-based PHOTON models on RefinedWeb (664 B tokens) and a training context length of 4096, while measuring inference efficiency at a context length of 2048 under the PF/DE protocol as same as main text. We trained a LLaMA-based vanilla transformer trained on the same dataset and context length as a baseline. All other experimental conditions were matched to those used in Table 2. At the 2.4B scale, we observed the same trend as for models up to 1.2B.

D Additional Implementation Details

D.1 Training Setting

We employ the 134B Pile-uncopyrighted dataset, consisting of 177,008,913 documents and 134,217,728,000 tokens (Gao et al., 2020). We also utilize the Llama tokenizer, which has a vocabulary size of $\#(\mathcal{V}) = 32{,}000$. For training, we set the total batch size to 256, the context window to 2048, and the number of training epochs to 1 for PHOTON. For both Vanilla Transformer and Block Transformer, we align the training compute budget with the corresponding PHOTON model in terms of total Flops. We employ the Adam optimizer with a learning rate of 3×10^{-4} and a warm-up period of 3,000 steps. The scalar hyperparameters of PHOTON are set to $\alpha = 0.0$. We further ablate the strength of the recursive loss by sweeping α, see Appendix C.1; increasing α generally improves downstream zero-shot performance. All experiments are conducted on an NVIDIA DGX H200 system.

D.2 Architecture

The vanilla Transformer employs an LLaMA architecture with adjusted parameter sizes. The configurations for the 600M, 900M, and 1.2B models are presented in Table 10, Table 11, and Table 12, respectively. Block Transformer follows the architecture proposed in the original paper with adjustment of parameters. The specifications for the 600M, 900M and 1.2B models are provided in Table 13, Table 14, and Table 15, respectively. The block decoders and token decoders in PHOTON are based on the LLaMA decoder architecture. The parameter configurations for the 600M, 900M, and 1.2B models are presented in Table 16, Table 17, and Table 18, respectively.

Table 8: Single-batch latency and hierarchical overhead

Model	Setting	600M (ms)	900M (ms)	1200M (ms)
Transformer	DE↓	28553.16	30806.52	42264.81
Transformer	PF↓	2290.33	1935.19	2745.48
Block Transformer	DE↓	10806.24	13310.22	14793.04
Block Transformer	PF↓	1994.21	2540.84	2645.92
PHOTON	DE↓	7320.83	8295.38	9278.10
PHOTON	PF↓	462.60	905.59	607.08

Table 9: Inference efficiency and language modeling quality for larger-scale models. Memory is measured per sample (GiB), throughput in K tokens/s, and TPM in K tokens/s/GiB. Throughput is computed using the number of *output* tokens only, following the main-paper definition: 128 output tokens for PF (2048 input/128 output) and 2048 output tokens for DE (128 input/2048 output). Quality is evaluated by WikiText perplexity and zero-shot accuracy on HS, SciQ, and ARCe.

Models	TPM		Memory		Throughput		PPL and Zero-shot Accuracy			
	PF↑	DE↑	PF↓	DE↓	PF↑	DE↑	WikiText↓	HS↑	SciQ↑	ARCe↑
Vanilla Transformer										
2.4 B	3.89	13.10	0.378	0.270	1.47	3.54	14.7164	63.16	84.60	56.19
PHOTON										
2.4 B	714.42	2808.20	0.048	0.027	33.95	74.42	21.0551	55.05	74.50	49.45

Table 10: Parameter breakdown for the Vanilla Transformer (600M).

Module	Hidden/ Int. / Layers	Params
Token Embedding (vocab=32000, d=1664)	1664 /– / –	53,248,000
Transformer Blocks (atten h=32, key value h=32, head d =52)	1664/4096/16	504,418,304
Final Norm (RMSNorm)	1664/ – / –	1,664
LM Head	1664/ – / –	53,248,000
Total	–	610,915,968

Table 11: Parameter breakdown for the Vanilla Transformer (900M).

Module	Hidden / Int. / Layers	Params
Token Embedding (vocab=32000, d=1792)	1792 / – / –	57,344,000
Transformer Blocks (atten h=32, key value h=32, head d =56)	1792/4608/20	752,424,960
Final Norm (RMSNorm)	1792/ – / –	1792
LM Head	1792/ – / –	57,344,000
Total	–	867,114,752

Table 12: Parameter breakdown for the Vanilla Transformer (1.2B).

Module	Hidden / Int. / Layers	Params
Token Embedding (vocab=32000, d=1920)	1920 / – / –	61,440,000
Transformer Blocks (atten h=32, key value h=32, head d =60)	1920/5120/24	1,061,775,360
Final Norm (RMSNorm)	1920/ – / –	1,920
LM Head	1920/ – / –	61,440,000
Total	–	1,184,657,280

Table 13: Parameter breakdown for the Block Transformer (600M).

Module	Hidden / Int. / Layers	Params
Embedder (vocab=32000, d=416)	– / – / –	13,312,000
BlockDecoder (atten h=32, key value h=32, head d =52)	1664 / 4096 / 8	252,210,816
Ctx Converter (in d = 1664, out d = 1664)	– / – / –	5,541,120
Embedder (vocab=32000, d=1664)	– / – / –	53,248,000
TokenDecoder (atten h=32, key value h=32, head d =52)	1664 / 4096 / 8	252,210,816
LM Head	1664 / – / –	53,248,000
Total	–	629,770,752

Table 14: Parameter breakdown for the Block Transformer (900M).

Module	Hidden / Int. / Layers	Params
Embedder (vocab=32000, d=448)	– / – / –	14,336,000
BlockDecoder (atten h=32, key value h=32, head d =56)	1792/4608/10	376,214,272
Ctx Converter (in d = 1792, out d = 1792)	– / – / –	6,426,112
Embedder (vocab=32000, d=1792)	– / – / –	57,344,000
TokenDecoder (atten h=32, key value h=32, head d =56)	1792/4608/10	376,214,272
LM Head	1792 / – / –	57,344,000
Total	–	887,878,656

Table 15: Parameter breakdown for the Block Transformer (1.2B).

Module	Hidden / Int. / Layers	Params
Embedder (vocab=32000, d=480)	– / – / –	15,360,000
BlockDecoder (atten h=32, key value h=32, head d =60)	1920 / 5120 / 12	530,889,600
Ctx Converter (in d = 2048, out d = 2048)	– / – / –	7,376,640
Embedder (vocab=32000, d=1664)	– / – / –	61,440,000
TokenDecoder (atten h=32, key value h=32, head d =60)	1920 / 5120 / 12	530,889,600
LM Head	1920/ – / –	61,440,000
Total	–	1,207,395,840

Table 16: PHOTON (600M)

Level	Module	Hidden / Int. / Layers	Params
Enc. ($l = 1$)	Embedder (vocab=32000, d=416)	– / – / –	13,312,000
	Ctx Chunker (block=4, concatenate)	– / – / –	–
	Ctx Encoder (atten h=32, kv h=32, head d=52)	1664 / 4096 / 4	126,106,240
Enc. ($l = 2$)	Ctx Chunker (block=4, linear)	– / – / –	11,083,904
	Ctx Encoder (atten h=32, kv h=32, head d=52)	1664 / 4096 / 4	126,106,240
Dec. ($l = 2$)	Ctx Converter (in d=1664, out d=1664)	– / – / –	5,541,120
	Ctx Decoder (atten h=32, kv h=32, head d=52)	1664 / 4096 / 4	126,106,240
Dec. ($l = 1$)	Ctx Converter (in d = 1664, out d = 1664)	– / – / –	5,541,120
	Embedder (vocab=32000, d=1664)	– / – / –	53,248,000
	Ctx Decoder (atten h=32, kv h=32, head d=52)	1664 / 4096 / 4	126,106,240
	LM Head (in d=1664, out d=32000)	1664 / – / –	53,248,000
Total	–	–	646,399,104

Table 17: PHOTON (900M)

Level	Module	Hidden / Int. / Layers	Params
Enc. ($l = 1$)	Embedder (vocab=32000, d=448)	– / – / –	14,336,000
	Ctx Chunker (block=4, concatenate)	– / – / –	–
	Ctx Encoder (atten h=32, kv h=32, head d=56)	1792 / 4608 / 5	188,108,032
Enc. ($l = 2$)	Ctx Chunker (block=4, linear)	– / – / –	12,854,016
	Ctx Encoder (atten h=32, kv h=32, head d=56)	1792 / 4608 / 5	188,108,032
Dec. ($l = 2$)	Ctx Converter (in d=1792, out d=1792)	– / – / –	6,426,112
	Ctx Decoder (atten h=32, kv h=32, head d=56)	1792 / 4608 / 5	188,108,032
Dec. ($l = 1$)	Ctx Converter (in d = 1792, out d = 1792)	– / – / –	6,426,112
	Embedder (vocab=32000, d=1792)	– / – / –	57,344,000
	Ctx Decoder (atten h=32, kv h=32, head d=56)	1792 / 4608 / 5	188,108,032
	LM Head (in d=1792, out d=32000)	1792 / – / –	57,344,000
Total	–	–	907,162,368

Table 18: PHOTON (1.2B)

Level	Module	Hidden / Int. / Layers	Params
Enc. ($l = 1$)	Embedder (vocab=32000, d=480)	– / – / –	15,360,000
	Ctx Chunker (block=4, concatenate)	– / – / –	–
	Ctx Encoder (atten h=32, kv h=32, head d=60)	1920 / 5120 / 6	265,445,760
Enc. ($l = 2$)	Ctx Chunker (block=4, linear)	– / – / –	14,755,200
	Ctx Encoder (atten h=32, kv h=32, head d=60)	1920 / 5120 / 6	265,445,760
Dec. ($l = 2$)	Ctx Converter (in d=9728, out d=2432)	– / – / –	7,376,640
	Ctx Decoder (atten h=32, kv h=32, head d=60)	1920 / 5120 / 6	265,445,760
Dec. ($l = 1$)	Ctx Converter (in d = 2432, out d = 2432)	– / – / –	7,376,640
	Embedder (vocab=32000, d=1920)	– / – / –	61,440,000
	Ctx Decoder (atten h=32, kv h=32, head d=60)	1920 / 5120 / 6	265,445,760
	LM Head (in d=1920, out d=32000)	1920 / – / –	61,440,000
Total	–	–	1,229,531,520