COPY-PASTE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Transformers are intuitively thought to be efficient at copying from their inputs. In this work, we show that the ability of LLMs to copy in context can be improved through explicit modeling. We propose Copy-Paste Language Models (CPLMs), adding a lightweight Copy-Paste head similar to pointer networks on top of language models. CPLMs improve training efficiency, downstream performance, and offer new operating points in long-context and retrieval setups. We validate our approach by pretraining models from 135M to 1.7B parameters, showing improvements across all scales and backbones with negligible training and inference overheads. We explain the success of CPLMs by showing that they partially mitigate the LM head gradient bottleneck through an alternative gradient path.

1 INTRODUCTION

Language is rich with lexical repetitions. Documents revisit entities and events, dialogues echo earlier turns, and code refers back to the same variables and functions. As a result, ∼60% of the tokens in FineWeb-Edu (Lozhkov et al., 2024), a popular pre-training corpus, already appear earlier in their 4k-token context. Copying from the context is therefore one of the most basic operations a language model has to perform. Indeed, Transformer LMs devote many attention heads to copying, such as induction heads (Olsson et al., 2022) and retrieval heads (Wu et al., 2025; Lin et al., 2026).

Yet the standard neural LM setup can make copying surprisingly hard to learn. Every prediction is produced by the LM head, a linear projection far narrower than the vocabulary. This low-rank layer limits the distributions a model can express (Yang et al., 2018a; Finlayson et al., 2024), while also discarding most of the training signal before it reaches the backbone (Godey & Artzi, 2026). Copying is exposed to both effects, however easily the backbone can model the copy mechanism. As a consequence, vanilla Transformer models fail on a trivial synthetic copying task at large vocabulary sizes (Section 4.1). This raises the question of whether copying could be encoded more efficiently.

We introduce Copy-Paste Language Models (CPLMs), which are designed to facilitate copying in LMs. CPLMs are inspired by pointer networks (Vinyals et al., 2015), which were designed to give encoder-decoder and recurrent models access to their inputs, and largely fell out of favor as the

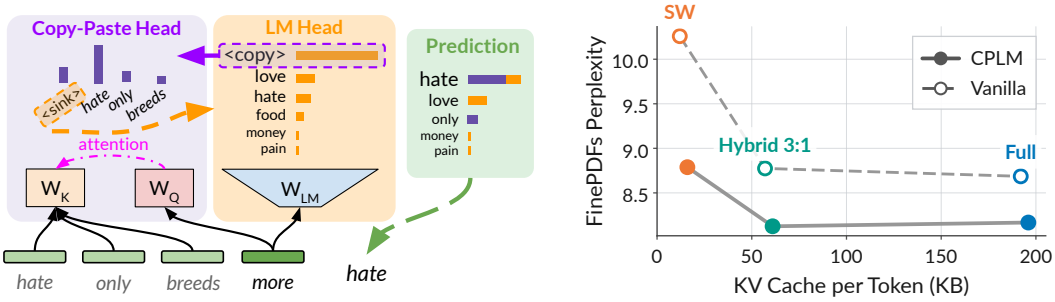


Figure 1: **Left:** Our Copy-Paste (CP) head. The LM head uses the probability of a special `<copy>` token to gate the copy distribution; a special `<sink>` token allows rolling back to the LM head. The prediction mixes both heads. **Right:** FinePDF perplexity of our 1.7B models vs. memory footprint. CPLMs beat their vanilla counterparts by 1–1.5 PPL, and the sliding-window (SW) CPLM matches a full-attention LM at 12× less memory per token, copy cache included.

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

problem was solved by self-attention. Yet we argue that a key obstacle to copying lies on the output side, in the LM head, and that a pointer offers a way around it. A CPLM adds a Copy-Paste (CP) head adjacent to the LM head, which predicts the next token by pointing at a past position in the context. The LM head decides when to copy by emitting a dedicated `<copy>` token, and a sink lets the CP head defer back to the LM head when nothing in the context is worth copying. Crucially, this copy mechanism is learned directly in the vocabulary space, and adds a second, much less compressed gradient pathway next to the LM head. Our contribution is thus not the copy mechanism itself, which remains close to prior pointer models, but showing that such a head substantially improves modern LM pretraining across backbones, and that it does so mostly by mitigating the LM head gradient bottleneck, as supported by targeted interventions (Section 4.1 and Appendix H). The CP head adds two lightweight projection matrices and negligible parameter and compute overheads (resp. 0.5% and 3% at 1.7B scale, and below 1% of training throughput at 9B and 30B; Appendix B).

Making copying easy to learn yields better models overall. We pretrain CPLMs from 135M to 1.7B parameters on up to 200B tokens across full-attention, sliding-window, hybrid and Gated DeltaNet backbones. CPLMs converge faster, reaching the final loss of their vanilla counterparts up to $2\times$ faster, and improve perplexity at every scale and on every backbone, by 1–1.5 points on out-of-distribution FinePDFs documents at 1.7B scale. These gains are not restricted to copied tokens: CPLMs also better predict the tokens they do not copy (Appendix G.1). Critically, the benefits carry over to supervised fine-tuning, where CPLMs improve on math, code and instruction following.

CPLMs also change how models use their context. Sliding-window CPLMs match full-attention and hybrid models using up to $12\times$ less memory per token, and they retrieve information from contexts far longer than those seen in training much better than their vanilla counterparts. CPLMs also rely on fewer induction heads in their backbone, suggesting that the CP head takes over part of their role. Finally, the CP head can read documents through their copy keys alone, which offers a memory-efficient alternative for retrieval-augmented setups.

2 METHOD

2.1 BACKGROUND & NOTATION

Standard LM Head A neural language model with hidden dimension D processes a token sequence $x_1, \ldots, x_T$, taken from a vocabulary $\mathcal{V}$ of size V, and produces an output representation $h_t \in \mathbb{R}^D$ for each position t, which only depends on the prefix $x_{\leq t}$. The LM head predicts the next token from h_t, using a learnable embedding $w_v \in \mathbb{R}^D$ for each token $v \in \mathcal{V}$. For scores z_u indexed by a set $\mathcal{S}$, we write $\mathrm{softmax}_{u\in\mathcal{S}}(z_u)_v = \exp(z_v)/\sum_{u\in\mathcal{S}}\exp(z_u)$ for the probability $p(v \mid \cdot)$, with $v \in \mathcal{S}$. With the embeddings w_v forming the LM head matrix $W \in \mathbb{R}^{V\times D}$, the LM distribution is:

$$p_{\mathrm{LM}}(v \mid x_{\leq t}) \;=\; \underset{u\in\mathcal{V}}{\mathrm{softmax}}\left(w_u^\top h_t\right)_v . \tag{1}$$

Softmax Bottleneck(s) Commonly, the vocabulary size V is much larger than the hidden dimension D. As all the logits $w_v^\top h_t$ are obtained from a D-dimensional vector, the LM head cannot represent every distribution over $\mathcal{V}$, which limits the model expressivity. This is commonly known as the softmax bottleneck, and has proven to impede expressivity (Yang et al., 2018a; Grivas et al., 2022; Finlayson et al., 2024; Basri & Jacobs, 2026). Godey & Artzi (2026) observe that the LM head bottlenecks optimization by destructively compressing the V-dimensional gradients to a D-dimensional output space during the backward pass. Our method reduces both these issues with a second prediction and gradient pathway that does not suffer from such bottlenecks as much.

2.2 COPY-PASTE LANGUAGE MODELS

The Copy-Paste Head The softmax bottleneck occurs because the target space is the vocabulary, and the LM head is a fixed parameter across generation. We propose to alleviate these constraints by building a second pathway for next-token prediction that does not have a fixed target space, and where the prediction is not anchored to a static weight matrix.

The second head's targets are the context positions $\tau \in \{1, \dots, t\}$. The head scores each position against the current state, as in self-attention, using a query $q_t = W_Q h_t$ and keys $k_\tau = W_K h_\tau$:

$$\alpha_t(\tau) \;=\; \underset{\tau' \in \{1,\dots,t\}}{\mathrm{softmax}} \left(k_{\tau'}^\top q_t / \sqrt{D} \right)_\tau \quad . \tag{2}$$

For simplicity, we set the dimension of the copy queries and keys to the hidden dimension D by default, i.e. $W_Q, W_K \in \mathbb{R}^{D \times D}$; much smaller dimensions perform similarly (Appendix C.2). We obtain a distribution over tokens by marginalizing over the positions that emit the same token:

$$p_{\mathrm{CP}}(v \mid x_{\le t}) \;=\; \sum_{\tau \le t \,:\, x_\tau = v} \alpha_t(\tau) \quad . \tag{3}$$

This distribution is only supported on the tokens that appear in the context.

Triggering the CP head We add a <copy> token to the LM head, with embedding $w_{\texttt{<copy>}}$, and use the probability that the LM head assigns to it as a gate for the copy-paste mechanism:

$$c_t \;=\; \underset{u \in \mathcal{V} \cup \{\texttt{<copy>}\}}{\mathrm{softmax}} \left(w_u^\top h_t \right)_{\texttt{<copy>}} \quad . \tag{4}$$

Because <copy> is never an actual target and only serves as a gate, trained through c_t in the mixture below, p_{LM} remains the distribution of Equation (1) over the actual vocabulary, which is the LM head distribution with <copy> removed and renormalized. The next-token distribution is then the mixture

$$p(v \mid x_{\le t}) \;=\; (1 - c_t)\, p_{\mathrm{LM}}(v \mid x_{\le t}) \;+\; c_t\, p_{\mathrm{CP}}(v \mid x_{\le t}) \quad . \tag{5}$$

Sink mechanism When the copy-paste mechanism is triggered, the support of the next-token distribution can only include in-context tokens. This could induce failure modes where the model attempts to copy a token that is not actually in the context, with no valid alternatives available, resulting in a wrong prediction. To avoid this, we add a sink mechanism to roll back to the LM head prediction. We add a learnable sink key $k_{\texttt{<sink>}}$ to the keys of Equation (2), and write

$$s_t \;=\; \underset{\tau \in \{\texttt{<sink>}, 1, \dots, t\}}{\mathrm{softmax}} \left(k_\tau^\top q_t / \sqrt{D} \right)_{\texttt{<sink>}} \tag{6}$$

for the probability that the copy head places on the sink rather than on any context position. α_t and p_{CP} remain defined over the context positions only (Equations (2) and (3)). Equation (5) becomes

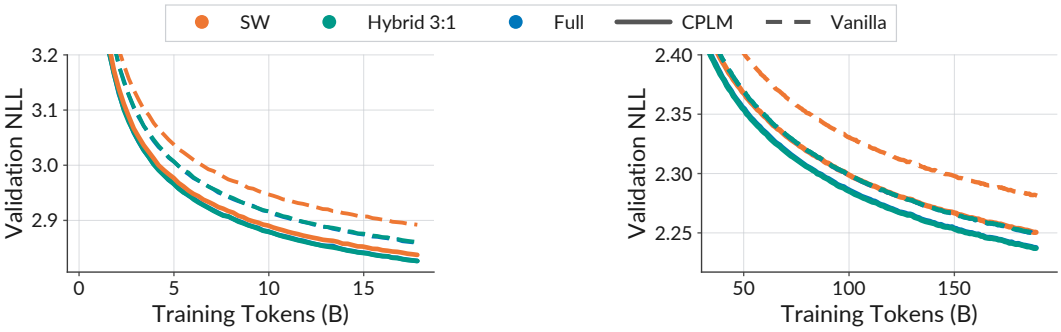
$$\boxed{p(v \mid x_{\le t}) \;=\; (1 - \lambda_t)\, p_{\mathrm{LM}}(v \mid x_{\le t}) \;+\; \lambda_t\, p_{\mathrm{CP}}(v \mid x_{\le t}), \qquad \lambda_t \;=\; c_t\,(1 - s_t) \quad .} \tag{7}$$

The effective gate λ_t is the probability of routing through the copy head *and* not falling back to the sink: c_t opens the copy pathway, and s_t lets the model close it again when no context position is a suitable target. Setting $s_t = 0$ recovers Equation (5). As a convex combination of two distributions over $\mathcal{V}$, Equation (7) is a valid distribution by construction, and the two pathways receive gradient in proportion to $1 - \lambda_t$ and λ_t respectively. CPLMs are trained end-to-end with the standard next-token cross-entropy on p only: there is no auxiliary loss and no copy supervision, and the model learns on its own when and where to copy. Equation (7) is close to the pointer sentinel mixture of Merity et al. (2017), where a sentinel inside the pointer softmax is the only gate: its probability weights the vocabulary distribution. In CPLMs, the gate c_t is instead predicted by the LM head together with the vocabulary, so that the copy decision is known before any copy attention is computed, and the sink only lets the CP head defer back to the LM head.

3 Experiments & Results

3.1 Pretraining

We pretrain models ranging from 135M to 1.7B parameters on up to 200B tokens of FineWeb-Edu data (Lozhkov et al., 2024). To study the interaction between backbone context handling and the CP head, we explore various backbones: full attention (Full), sliding-window (SW), and a hybrid architecture (Hybrid) where full-attention and sliding-window layers are interleaved. Hybrid models give sliding-window backbones access to the whole context through a few global layers, and thus serve as



(a) 135M (∼20B tokens)

(b) 1.7B (∼200B tokens)

Figure 2: FineWeb-Edu validation loss vs. training tokens (before LR decay), for $w = 256$. Colors distinguish backbone attention setups. Full and Hybrid training curves are essentially superposed.

Model	Type	HellaSwag	LAMBADA	PIQA	SciQ	ARC-e	ARC-c	WinoGrande	Avg.
Full	Vanilla	61.12	44.46	**75.24**	88.20	**68.39**	40.96	58.80	62.45
	CPLM	**61.70**	**46.92**	74.86	**89.20**	68.10	**41.21**	**59.43**	**63.06**
SW	Vanilla	61.00	42.46	74.48	87.00	67.17	**41.30**	58.01	61.63
	CPLM	**61.78**	**46.42**	**74.70**	**87.30**	**67.59**	41.13	**58.41**	**62.48**
Hybrid	Vanilla	61.08	43.70	74.48	**87.80**	**69.32**	40.96	58.17	62.22
	CPLM	**61.70**	**46.92**	74.48	87.40	66.46	**41.55**	**59.59**	**62.59**

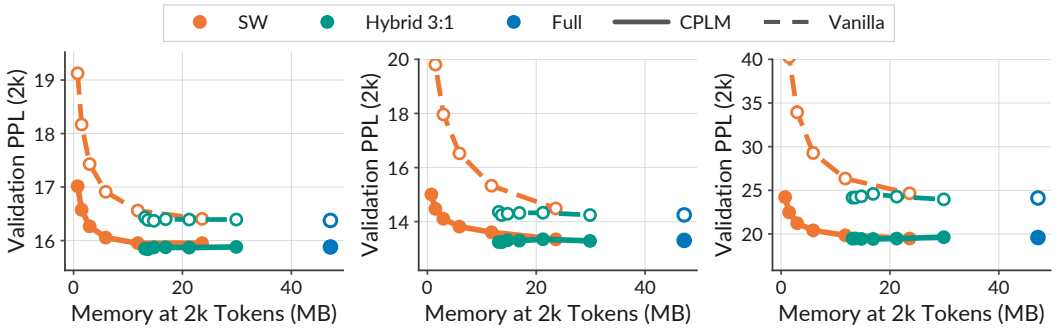
Table 1: Zero-shot accuracy (%) of our 1.7B base models (95k steps). Best of each pair in bold.

a natural baseline for adding global access to local backbones. Full and Hybrid backbones already access every position, so that CPLM gains on these backbones cannot stem from adding global access alone. We also report Gated DeltaNet or GDN (Yang et al., 2025) and Hybrid-GDN comparisons in Appendix C.5. At 135M scale, we run several ablations, notably varying the backbone sliding window w for SW and Hybrid models, from 32 to 1024. Based on our findings (Figure 3), we set $w = 256$ for all Hybrid and SW 1.7B models. Our 61 135M-scale runs cost ∼50 GH200 hours each, and our 1.7B runs ∼900 B200 hours each. We follow the SmolLM2 recipe (Allal et al., 2025), with identical hyperparameters for both arms (Appendix A).

We report our training dynamics for CPLMs and vanilla LMs at 135M and 1.7B scale in Figure 2, and at 360M scale in Appendix C.3. The dynamics are favorable to CPLMs in all configurations, reaching the final loss of vanilla models $1.25\times$ faster in the Full and Hybrid attention setups, and $2\times$ faster in the SW setup at 1.7B scale. The 1.7B SW CPLM matches the Full LM with $12\times$ less memory at 4k tokens, as it only caches one copy key per token beyond its window.

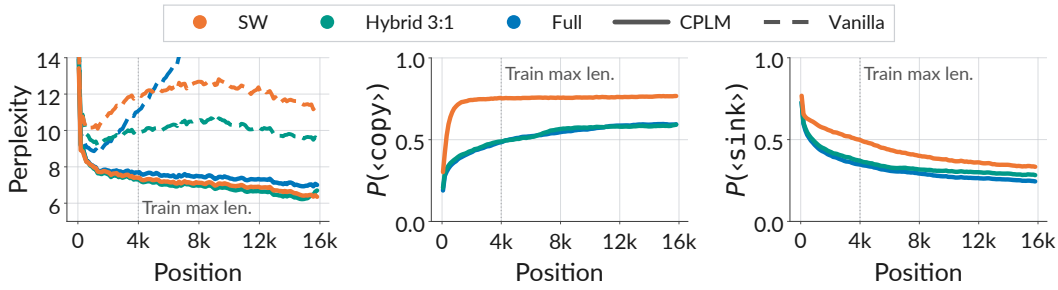
Downstream Evaluations In Table 1, we report the zero-shot performance of the 1.7B models on usual benchmarks: HellaSwag (Zellers et al., 2019), LAMBADA (Paperno et al., 2016), PIQA (Bisk et al., 2020), SciQ (Welbl et al., 2017), ARC (Clark et al., 2018), and WinoGrande (Sakaguchi et al., 2021). Overall, CPLMs outperform their vanilla counterparts by +0.4–0.9 accuracy points while inducing no meaningful compute overhead (∼3%, Appendix B). The largest gains are on LAMBADA (+2.5–4.0 points), whose target word often appears earlier in the passage and can thus be copied, while gains on the other tasks are smaller. CPLMs are not restricted to copying, however: they also lower the NLL of tokens that they do not copy (Appendix G.1).

Effect of the Window Size w Figure 3 shows the impact of the sliding window parameter w across different document sources, allowing direct comparison between CPLMs and vanilla LMs. Across all window sizes w, CPLMs have lower perplexity, dropping by 0.5–2 points on in-domain FineWeb-Edu, and by up to 15 points on other datasets, a trend that extends to non-English data (Appendix G.3). A SW CPLM with a relatively small window ($w \simeq 128$) generally matches the perplexity of a full-attention vanilla model while using much less memory. SW CPLMs also outperform all window-matched vanilla Hybrids, which use more memory.



(a) FineWeb-Edu (b) FinePDFs (OOD) (c) arXiv (OOD)

Figure 3: Validation perplexity vs. memory usage for sliding window sizes from 32 to 1024 at 135M scale. CPLMs improve the memory–performance frontier.



(a) Perplexity (OOD) (b) Copy gate (c) Sink

Figure 4: Per-position statistics of our 1.7B models on 5,000 FinePDFs documents of 16k tokens, i.e. up to $4\times$ the pretraining context. Full and Hybrid models use YaRN ($\times 4$). **(a)** Perplexity. **(b–c)** Average probability of the `<copy>` token and of the sink. CPLMs keep improving with context beyond their training length, and use the CP head more as the context grows.

Rolling Perplexity Figure 4 reports the perplexity of our 1.7B models as a function of the position in the document, up to $4\times$ their pretraining context. The perplexity of CPLMs decreases steadily with position across all backbones, including beyond the training length, reaching 6.5–7 at 16k tokens. In contrast, the perplexity of all vanilla models increases after roughly 1k tokens: SW and Hybrid models plateau around 10–13, and the Full model diverges beyond its training length. The CP head is used more as the context grows: the `<copy>` probability increases with position while the sink probability decreases, and the average copy distance grows almost linearly with position (Appendix G.2). The SW CPLM relies on the CP head the most, and copies tokens more than 2,000 positions back at 16k tokens, far beyond its window of 256 tokens.

Ablations In Appendix C, we conduct ablations at 135M scale to explore and validate several aspects of CPLMs. We run a multi-seed analysis at 135M scale, showing that our gains are 15–$40\times$ larger than the seed-to-seed standard deviation ($p < 10^{-3}$), in line with the low sensitivity of pretraining loss to random seeds (van der Wal et al., 2025). We show that the dimension of the queries and keys in the CP head barely matters, with dimension 64 achieving a similar performance to dimension 2,048, and that the sink matters most for small windows (Appendix C.4). Finally, we compare the CP head to alternative pointer designs (Appendix C.6): it outperforms the partitions that Chang et al. (2023) proposed for decoder LMs, and matches or outperforms sigmoid-gated (See et al., 2017) and shared-softmax (Gu et al., 2016) mixtures by small margins.

3.2 Supervised Fine-Tuning (SFT)

We hypothesize that stronger token copying abilities result in better chain-of-thought reasoning and coding abilities, because copy is inherently important for these tasks. To validate this hypothe-

		PPL (↓)		Math		Coding		Instruction	
Attention	Model	Tulu-3	Nemotron-3	MATH	GSM8K	HumanEval	MBPP	IFEval	Avg
Full	Vanilla	2.52	4.08	13.9	28.9	20.4	16.9	41.2	24.2
	CPLM	**2.47**	**3.78**	**17.5**	**33.1**	**21.1**	**17.5**	**43.0**	**26.4**
Hybrid 3:1	Vanilla	2.52	4.10	16.5	27.9	19.8	**19.0**	38.3	24.3
	CPLM	**2.48**	**3.80**	**17.4**	**30.4**	**22.1**	17.4	**40.1**	**25.5**
SW-256	Vanilla	2.79	5.65	7.2	18.6	15.4	16.1	37.2	18.9
	CPLM	**2.54**	**4.08**	**16.4**	**25.2**	**19.2**	**20.8**	**38.4**	**24.0**

Table 2: Downstream results after Supervised Fine-Tuning on our 1.7B models. PPL: Tulu-3 val. and `Nemotron3-SFT-Math-v4` perplexity. Evaluation details in Appendix D.

		Synthetic (0–16k)		LongBench (16k)						
Attention	Model	BABILong	RULER	Single-Doc	Multi-Doc	Summ.	Few-shot	Synth.	Code	**Avg**
Full	Vanilla	26.4	33.6	5.7	6.1	**11.8**	20.7	**1.7**	15.4	10.4
	CPLM	**32.0**	**39.6**	**6.8**	**6.7**	10.2	**28.3**	0.8	**18.6**	**12.2**
Hybrid 3:1	Vanilla	27.7	33.2	4.6	**5.3**	8.4	20.5	0.7	17.4	9.5
	CPLM	**32.9**	**37.8**	**5.4**	3.3	**14.0**	**35.1**	**1.7**	**25.6**	**14.3**
SW-256	Vanilla	13.8	7.9	3.0	2.9	6.4	16.6	1.4	14.8	7.4
	CPLM	**16.1**	**29.2**	**7.7**	**4.4**	**10.8**	**30.3**	**2.1**	**25.0**	**13.4**

Table 3: Long-context results for the 1.7B base models. BABILong: mean accuracy over qa1–qa5 and lengths 0k–16k; RULER: mean over its 13 tasks and lengths 4k–16k; LongBench: category means at 16k context, Avg over the 16 tasks. Full and Hybrid use YaRN ($\times$4 from 4096); SW-256 uses plain RoPE. **Bold** marks the better model of each pair. Details in Appendix F.

sis, we perform Supervised Fine-Tuning (SFT) on our pretrained 1.7B models. We use the Tulu-3 dataset (Lambert et al., 2025), well-suited for 1.7B scale, with sequences of up to 16k tokens.

Table 2 reports evaluation metrics for SFT 1.7B models. CPLMs outperform Vanilla LMs by 1.2–5.1 points in average score, and on every benchmark except MBPP for the Hybrid model. The gap is particularly consistent on math benchmarks that benefit from chain-of-thought reasoning, even at this tiny scale, and holds for every k in pass@k (Figure 12). The SW CPLM is once again on par with full-attention models while using significantly less memory. This corroborates the lower perplexity of CPLMs on both Tulu-3 and Nemotron-3 SFT data (NVIDIA et al., 2025).

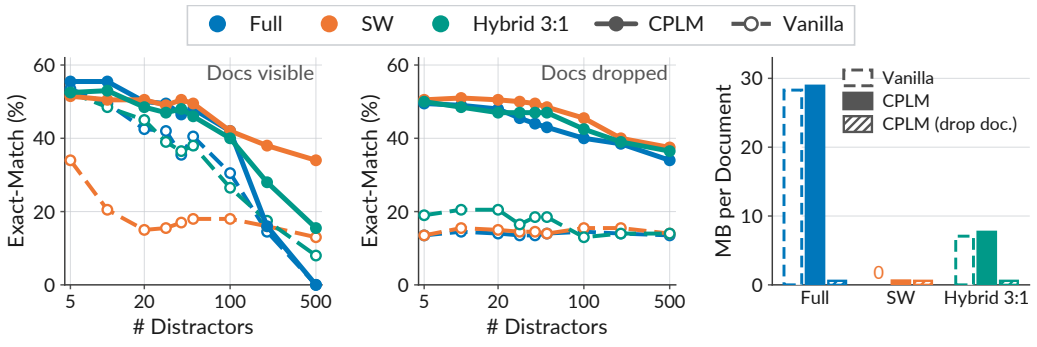
3.3 Long-Context Abilities

A model that copies better is likely to also retrieve information more accurately from long documents. We verify this in Table 3, where we show that our base 1.7B CPLMs outperform their vanilla counterparts on both synthetic tasks (Hsieh et al., 2024; Kuratov et al., 2024) by +7.5 average points, and on LongBench (Bai et al., 2024) by +4.1 average points.

We also run a single-needle retrieval test. A 7-digit "magic number" is placed at one of 8 depths inside neutral filler text, for context lengths from 1k to 64k tokens, i.e. up to $16\times$ the 4k pretraining context (details in Appendix E). Averaged over the whole depth $\times$ length grid, CPLM beats Vanilla for every architecture. The gain is small for Full attention (49.7 $\rightarrow$ 54.8), large for Hybrid 3:1 (58.2 $\rightarrow$ 83.6), and very large for SW (14.3 $\rightarrow$ 99.4). The SW CPLM finds the needle almost perfectly at every length and depth, with a window of only $w = 256$.

3.4 Efficient Retrieval

The CP head can also be seen as a form of external memory. We explore whether CPLMs can learn to copy from documents when the backbone cannot access the document's content. To this end, we fine-tune the base 1.7B vanilla models and CPLMs (Full attention, SW, and Hybrid 3:1) on the train set of HotpotQA (Yang et al., 2018b). To enable robustness to a variable number of distractor documents, each training sample contains the two gold paragraphs and $n \sim \mathcal{U}[5, 50]$ distractor paragraphs drawn from a shared pool, with no truncation. All models use the same recipe: one



(a) Document KV (b) No Document KV (c) Memory requirements

Figure 5: HotpotQA performance for 1.7B models. Models are fine-tuned and evaluated either with full document KV visible (**left**), or with only a single access to the document memory: last-layer KV for Vanilla, and CP states for CPLMs (**middle**).

epoch, learning rate 10^{-4} with cosine decay, and batch size 32. We then evaluate 200 validation questions with n from 5 to 500 distractors, so that differences below about 3.5 points are within noise. In the *Docs visible* regime, the backbone attends over the whole input. In the *Docs dropped* regime, each document is encoded independently by the backbone, and the question and answer tokens cannot attend to the documents in the backbone layers. A CPLM then reaches the documents only through its copy head. As a control, a vanilla model reaches the documents only through the last-layer keys and values. All models are fine-tuned in their evaluation regime.

Across backbones and distractor counts, CPLMs match or outperform their vanilla counterparts (Figure 5a) at a small memory overhead: the copy-paste keys add 2% to the cache of Full attention and 8% to Hybrid 3:1 (Figure 5c), and slow down decoding by less than 2% at this scale (Appendix B). The gap is largest with SW models, where the vanilla LM loses the documents that leave its window (15% EM at 20 distractors) while the SW CPLM still reaches 34% at 500 distractors, where every Full model fails. The CP head cache alone carries this retrieval ability (Figure 5b): with documents hidden from the backbone, CPLMs keep 90–97% of their accuracy with 5 distractors and degrade gracefully as distractors grow, still reaching 34–38% EM with 500 distractors, on par with or above every model that sees the documents, while using a 50× smaller total cache than a Full model. Since documents are encoded independently, their copy keys can also be computed once and reused across queries. In comparison, the last-layer control, with twice the memory of the CP head cache, stays at the level of a model that never sees the documents (14% EM).

4 Analysis

4.1 CPLMs Mitigate the LM Head Bottleneck

Godey & Artzi (2026) show that the LM head is a gradient bottleneck. For a vocabulary size V, the gradient of the loss with respect to the logits has rank $V - 1$ as soon as most tokens are predictable in some context. It reaches the backbone only after backpropagation through the LM head, whose rank is bounded by the hidden dimension D, and 95–99% of its norm is lost in pretrained LMs. They further show that, in terms of nullspace projection, the backward operator of the LM head behaves like a random matrix of the same shape: the lost fraction is essentially dictated by the ratio between D and V. As a result, small models fail to learn simple patterns once the vocabulary is large compared to their hidden dimension, even when they are expressive enough to model them.

The CP head does not escape this random-matrix behavior, but it has a much more favorable shape. Its logits are defined over the t positions of the context rather than over the V tokens of the vocabulary, and its output matrix, the copy keys $W_K h_{1:t}$, is computed from the context rather than stored as a parameter. Since $t \ll V$ for typical contexts, a rank-D projection of the CP head logit gradient loses a much smaller fraction of its norm than the LM head projection does, and this fraction depends on the context length rather than on the vocabulary size.

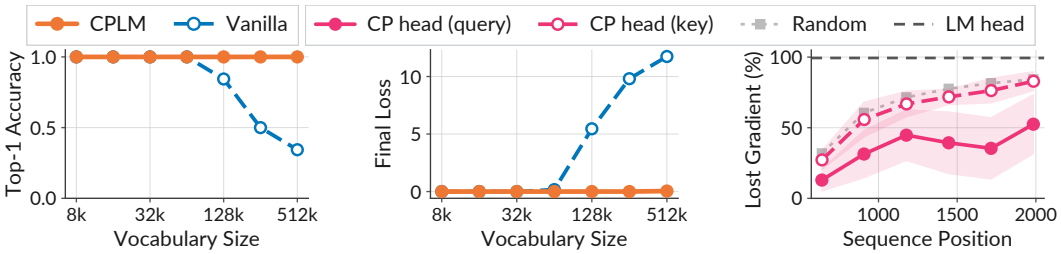
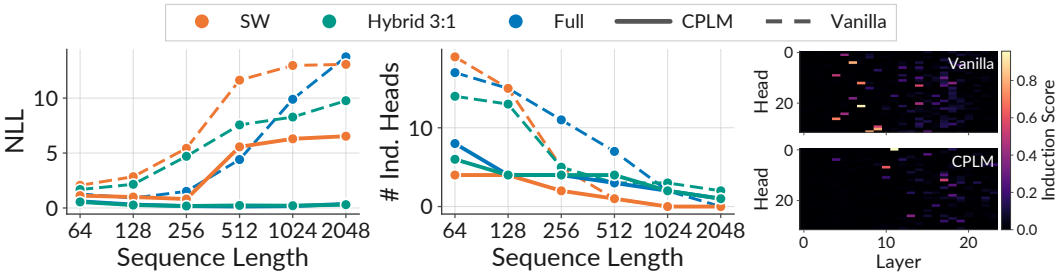


Figure 6: The CP head mitigates the LM head gradient bottleneck (Godey & Artzi, 2026). **Left and Center:** Accuracy and final validation loss for a 135M model on the SpamLang synthetic task, showing that CPLMs converge for all vocabulary sizes. **Right:** Fraction of the logit gradient norm that projects to the nullspace of $W_{\{LM,Q,K\}}$ for a 135M full-attention model. *Random*: same measurement for a random matrix with the shape of the CP head output matrix. Both CP head pathways let through much more of the signal than the LM head, especially at earlier positions.



(a) Induction performance (b) Induction head count (c) Induction head score

Figure 7: Induction analysis for 1.7B models. **Left**: We measure the performance of models when copying a randomly generated sequence; **Middle**: Total number of triggered induction heads across sequence lengths; **Right**: Heatmap of induction scores across all heads. Overall, CPLMs have stronger copy abilities while exhibiting far fewer induction heads.

We verify this using the experimental design of Godey & Artzi (2026), applied to CPLMs. In Figure 6, we first measure the fraction of the gradient norm lost in the LM head and in the CP head of our 135M Full vanilla model and CPLM, on FineWeb-Edu sequences of 2,048 tokens, and compare it to a random matrix of the same shape as the CP head output matrix. We then train both architectures on SpamLang, where each sequence repeats a single uniformly drawn symbol, with a fixed hidden dimension (576) and vocabularies ranging from 8k to 512k tokens.

As shown in Figure 6, the LM heads lose over 99% of the gradient norm, in line with Godey & Artzi (2026). The CP head keys stay close to the random reference, as expected, so their smaller effective vocabulary space lets them conserve up to 85% of the gradient norm, opening a much wider gradient channel than the LM head, especially at early positions. On SpamLang, vanilla models fail beyond a 64k vocabulary (34% accuracy at 512k), while CPLMs solve the task at every vocabulary size. This benefit is not limited to copied tokens: CPLMs reach a lower NLL than vanilla models even for tokens they are very unlikely to copy (Appendix G.1), suggesting that the extra gradient signal also improves the backbone. In Appendix H, a stop-gradient ablation and byte-level models both remove most of the benefits of CP heads, supporting this causal role.

4.2 CPLMs Need Fewer Induction Heads

We test whether the CP head takes over in-context copying from the backbone of our 1.7B models using the repeated-random-tokens probe from Olsson et al. (2022): a block of L random tokens is repeated once ($T{=}2L$). We measure the NLL on the repeated half (Figure 7a). For each backbone head, we also compute its induction score (Olsson et al., 2022), i.e. its average attention from a repeated token to the token that followed the previous occurrence of this token, and we count as induction heads the heads whose score exceeds 0.3 (Figures 7b and 7c). Full and Hybrid CPLMs

copy almost perfectly at every length. Their vanilla counterparts degrade as the repeat distance grows: at $T{=}2048$, Full Vanilla scores worse than chance (13.8 vs. $\ln|V|{=}10.8$), even though the repeat is inside its training context, as FineWeb-Edu documents are mostly shorter than 4k tokens, so that such long-range repeats are rare during training. CPLM backbones contain far fewer induction heads: 4 against 13–15 for vanilla models at $T{=}128$ (middle). They typically concentrate induction in a single sharp head that stays strong at all lengths (right). The explicit CP head thus absorbs the induction work that vanilla models spread across many heads, and does it more reliably.

5 Related Work

Softmax Bottleneck & Mitigations The low rank of the LM head limits the distributions an LM can express (Yang et al., 2018a). Remedies mix several LM heads (Yang et al., 2018a; 2019; Takase et al., 2018), apply element-wise non-linearities to the logits (Kanai et al., 2018; Ganea et al., 2019), or use several facet embeddings (Chang & McCallum, 2022). How much this limitation matters in practice is debated (Grivas et al., 2022; Basri & Jacobs, 2026; Finlayson et al., 2024; Godey et al., 2024). These methods target *forward* expressivity and keep an output matrix shared across contexts, so that they leave the *backward* bottleneck of Godey & Artzi (2026) in place, which the CP head mitigates with a second gradient pathway (Section 4.1).

Pointer & Copy Mechanisms The CP head is a form of pointer network (Vinyals et al., 2015). Pointers have been combined with a vocabulary softmax through a supervised switch (Gulcehre et al., 2016), a sigmoid gate (See et al., 2017), a shared normalizer (Gu et al., 2016), or a sentinel (Merity et al., 2017), mostly in encoder-decoder or recurrent models and to handle rare words. In decoder-only Transformers, where self-attention can already retrieve any previous token, pointers are rarely used (Chang et al., 2023), with two exceptions: TRIME (Zhong et al., 2022) trains a Transformer LM with a normalizer shared between vocabulary logits and similarity scores to past contexts, and Chang et al. (2023) fine-tune pretrained LMs with pointer partitions in the softmax to break the softmax bottleneck. Beyond our gradient-bottleneck view, we pretrain from scratch up to 1.7B parameters on four backbones, whereas TRIME stops at 247M parameters and Chang et al. (2023) only fine-tune, and we study sub-quadratic backbones, which prior pointer models did not consider and where CP heads help most (Section 3.4). Our gating design matches or outperforms alternative pointer designs by small margins (Appendix C.6).

Externalized Memory The CP head is also a context memory. The neural cache (Grave et al., 2017) and kNN-LM (Khandelwal et al., 2020) retrieve from past hidden states at inference, and TRIME trains such memories jointly; a tuned neural cache brings no gain on our models (Appendix C.7). Memorizing Transformers (Wu et al., 2022) attend to cached per-layer keys and values, RETRO (Borgeaud et al., 2022) cross-attends to retrieved chunks, CoG (Lan et al., 2023) copies phrases from a corpus, and several RAG methods precompute per-document KV caches (Lu et al., 2025; Ma et al., 2025; Yao et al., 2025). Like the cache, the CP head stores one vector per token, but in a dedicated learned key space. On HotpotQA, with documents hidden from the backbone, these keys alone keep 90–97% of CPLM accuracy with a $50\times$ smaller total cache than a full KV cache, whereas last-layer states with twice that memory fail (Section 3.4). The CP head also offsets the copying and recall weaknesses of sub-quadratic backbones (Jelassi et al., 2024; Arora et al., 2024): sliding-window CPLMs match full-attention vanilla LMs.

6 Conclusion

We introduced Copy-Paste Language Models, which complement the LM head with a lightweight pointer over past positions. Although self-attention can already retrieve any previous token, the copied token must still be decoded through the low-rank LM head, and the CP head adds a second, less compressed prediction and gradient pathway. At negligible cost, CPLMs train faster and improve perplexity, downstream, fine-tuning and long-context results on every backbone we tested, and let sliding-window models match full attention with a fraction of the memory. The CP head only relaxes the LM head bottleneck for tokens that appear in the context, and our experiments stop at 1.7B parameters. Extending the copy mechanism to spans or to external corpora and studying how its benefits evolve at larger scales are natural next steps.

AI USE STATEMENT

In this work, we used generative AI tools for literature search, for drafting and editing parts of the paper, including the related work section, proofreading and formatting, and for implementing standard methods such as evaluation and plotting scripts. We have not used generative AI tools to generate research ideas, design experiments, or produce experimental results: all reported numbers come from our own training and evaluation runs. We have reviewed all AI-assisted work: AI-generated code was verified and tested by the authors, AI-drafted or AI-edited text was checked and revised by the authors against our experimental results, and all references were verified manually. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not involve human subjects or personal data collection. All models are pretrained on FineWeb-Edu (Lozhkov et al., 2024), a publicly available and filtered web corpus, fine-tuned on the publicly available Tulu-3 data (Lambert et al., 2025) and HotpotQA (Yang et al., 2018b), and evaluated on public benchmarks. As with any language model trained on web data, our models may reproduce biases or factual errors present in their training data. They are small research models that have not undergone safety alignment, and should not be deployed in user-facing applications without further evaluation. Because CPLMs copy from their context more easily, they may also reproduce sensitive or copyrighted content present in their inputs more faithfully. Conversely, sliding-window CPLMs reach the performance of full-attention models with a much smaller memory footprint, which can lower the cost of inference.

REPRODUCIBILITY STATEMENT

The CP head, its gating and its sink mechanism are fully specified in Section 2.2, and the two-stage decoding procedure in Appendix C.6. Architectures, data, optimizer and schedules for all pretraining runs are given in Appendix A, and multi-seed results at the 135M scale in Appendix C.1. The output-mechanism baselines we compare to are described in Appendix C.6. The supervised fine-tuning evaluations are detailed in Appendix D, the needle-in-a-haystack protocol in Appendix E, and the long-context benchmarks in Appendix F. The HotpotQA fine-tuning recipe and the two document-access regimes are described in Section 3.4. All datasets used in this work are publicly available, and we will release our code and trained checkpoints upon publication.

REFERENCES

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martin Blazquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Agustín Piqueres Lajarín, Hynek Kydlíček, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan Son NGUYEN, Ben Burtenshaw, Clémentine Fourrier, Haojun Zhao, Hugo Larcher, Mathieu Morlon, Cyril Zakka, Colin Raffel, Leandro Von Werra, and Thomas Wolf. SmolLM2: When smol goes big — data-centric training of a fully open small language model. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=3JiCl2A14H.

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Re. Zoology: Measuring and improving recall in efficient language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=LY3ukUANko.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, Bangkok, Thailand, August

2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172/.

Ronen Basri and David Jacobs. The softmax bottleneck does not limit the probabilities of the most likely tokens. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=DgJqQk6y19.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: reasoning about physical commonsense in natural language. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 7432–7439. AAAI Press, 2020. doi: 10.1609/AAAI.V34I05.6239. URL https://doi.org/10.1609/aaai.v34i05.6239.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego De Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack Rae, Erich Elsen, and Laurent Sifre. Improving language models by retrieving from trillions of tokens. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 2206–2240. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/borgeaud22a.html.

Haw-Shiuan Chang and Andrew McCallum. Softmax bottleneck makes language models unable to represent multi-mode word distributions. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8048–8073, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.554. URL https://aclanthology.org/2022.acl-long.554.

Haw-Shiuan Chang, Zonghai Yao, Alolika Gon, Hong Yu, and Andrew McCallum. Revisiting the architectures like pointer networks to efficiently improve the next word distribution, summarization factuality, and beyond. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 12707–12730, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.805. URL https://aclanthology.org/2023.findings-acl.805/.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

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

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Matthew Finlayson, John Hewitt, Alexander Koller, Swabha Swayamdipta, and Ashish Sabharwal. Closing the curious case of neural text degeneration. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=dONpC9GL1o.

Octavian Ganea, Sylvain Gelly, Gary Becigneul, and Aliaksei Severyn. Breaking the softmax bottleneck via learnable monotonic pointwise non-linearities. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2073–2082. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/ganea19a.html.

Nathan Godey and Yoav Artzi. Lost in backpropagation: The LM head is a gradient bottleneck. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=PMDKvhbE4x.

Nathan Godey, Éric Villemonte de la Clergerie, and Benoît Sagot. Why do small language models underperform? studying language model saturation via the softmax bottleneck. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=MoitXWlXcS.

Edouard Grave, Armand Joulin, and Nicolas Usunier. Improving neural language models with a continuous cache. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B184E5qee.

Andreas Grivas, Nikolay Bogoychev, and Adam Lopez. Low-rank softmax can have unargmaxable classes in theory but rarely in practice. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6738–6758, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.465. URL https://aclanthology.org/2022.acl-long.465/.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. Incorporating copying mechanism in sequence-to-sequence learning. In Katrin Erk and Noah A. Smith (eds.), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1631–1640, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1154. URL https://aclanthology.org/P16-1154/.

Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. Pointing the unknown words. In Katrin Erk and Noah A. Smith (eds.), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 140–149, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1014. URL https://aclanthology.org/P16-1014/.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=kIoBbc76Sy.

Samy Jelassi, David Brandfonbrener, Sham M. Kakade, and eran malach. Repeat after me: Transformers are better than state space models at copying. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=duRRoGeoQT.

Sekitoshi Kanai, Yasuhiro Fujiwara, Yuki Yamanaka, and Shuichi Adachi. Sigsoftmax: Reanalysis of the softmax bottleneck. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/9dcb88e0137649590b755372b040afad-Paper.pdf.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HklBjCEKvH.

Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. Babilong: testing the limits of llms with long context reasoning-in-a-haystack. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James Validad Miranda, Alisa Liu, Nouha Dziri, Xinxi Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Christopher Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=i1uGbfHHpH.

Tian Lan, Deng Cai, Yan Wang, Heyan Huang, and Xian-Ling Mao. Copy is all you need. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=CROlOA9Nd8C.

Aitor Lewkowycz, Anders Johan Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Venkatesh Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=IFXTZERXdM7.

Yuping Lin, Zitao Li, Yue Xing, Pengfei He, Yingqian Cui, Yaliang Li, Bolin Ding, Jingren Zhou, and Jiliang Tang. Retrieval heads are dynamic. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15710–15729, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.715. URL https://aclanthology.org/2026.acl-long.715/.

Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. Fineweb-edu: the finest collection of educational content, 2024. URL https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu.

Songshuo Lu, Hua Wang, Yutian Rong, Zhi Chen, and Yaohua Tang. TurboRAG: Accelerating retrieval-augmented generation with precomputed KV caches for chunked text. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 6588–6601, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.334. URL https://aclanthology.org/2025.emnlp-main.334/.

Dongyang Ma, Yan Wang, and Tian Lan. Block-attention for efficient prefilling. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=7zNYY1E2fq.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Byj72udxe.

NVIDIA, :, Aaron Blakeman, Aaron Grattafiori, Aarti Basant, Abhibha Gupta, Abhinav Khattar, Adi Renduchintala, Aditya Vavre, Akanksha Shukla, Akhiad Bercovich, Aleksander Ficek, Aleksandr Shaposhnikov, Alex Kondratenko, Alexander Bukharin, Alexandre Milesi, Ali Taghibakhshi, Alisa Liu, Amelia Barton, Ameya Sunil Mahabaleshwarkar, Amir Klein, Amit Zuker, Amnon Geifman, Amy Shen, Anahita Bhiwandiwalla, Andrew Tao, Anjulie Agrusa, Ankur Verma, Ann Guan, Anubhav Mandarwal, Arham Mehta, Ashwath Aithal, Ashwin Poojary, Asif Ahamed, Asit Mishra, Asma Kuriparambil Thekkumpate, Ayush Dattagupta, Banghua Zhu, Bardiya Sadeghi,

Barnaby Simkin, Ben Lanir, Benedikt Schifferer, Besmira Nushi, Bilal Kartal, Bita Darvish Rouhani, Boris Ginsburg, Brandon Norick, Brandon Soubasis, Branislav Kisacanin, Brian Yu, Bryan Catanzaro, Carlo del Mundo, Chantal Hwang, Charles Wang, Cheng-Ping Hsieh, Chenghao Zhang, Chenhan Yu, Chetan Mungekar, Chintan Patel, Chris Alexiuk, Christopher Parisien, Collin Neale, Cyril Meurillon, Damon Mosk-Aoyama, Dan Su, Dane Corneil, Daniel Afrimi, Daniel Lo, Daniel Rohrer, Daniel Serebrenik, Daria Gitman, Daria Levy, Darko Stosic, David Mosallanezhad, Deepak Narayanan, Dhruv Nathawani, Dima Rekesh, Dina Yared, Divyanshu Kakwani, Dong Ahn, Duncan Riach, Dusan Stosic, Edgar Minasyan, Edward Lin, Eileen Long, Eileen Peters Long, Elad Segal, Elena Lantz, Ellie Evans, Elliott Ning, Eric Chung, Eric Harper, Eric Tramel, Erick Galinkin, Erik Pounds, Evan Briones, Evelina Bakhturina, Evgeny Tsykunov, Faisal Ladhak, Fay Wang, Fei Jia, Felipe Soares, Feng Chen, Ferenc Galko, Frank Sun, Frankie Siino, Gal Hubara Agam, Ganesh Ajjanagadde, Gantavya Bhatt, Gargi Prasad, George Armstrong, Gerald Shen, Gorkem Batmaz, Grigor Nalbandyan, Haifeng Qian, Harsh Sharma, Hayley Ross, Helen Ngo, Herbert Hum, Herman Sahota, Hexin Wang, Himanshu Soni, Hiren Upadhyay, Huizi Mao, Huy C Nguyen, Huy Q Nguyen, Iain Cunningham, Ido Galil, Ido Shahaf, Igor Gitman, Ilya Loshchilov, Itamar Schen, Itay Levy, Ivan Moshkov, Izik Golan, Izzy Putterman, Jan Kautz, Jane Polak Scowcroft, Jared Casper, Jatin Mitra, Jeffrey Glick, Jenny Chen, Jesse Oliver, Jian Zhang, Jiaqi Zeng, Jie Lou, Jimmy Zhang, Jinhang Choi, Jining Huang, Joey Conway, Joey Guman, John Kamalu, Johnny Greco, Jonathan Cohen, Joseph Jennings, Joyjit Daw, Julien Veron Vialard, Junkeun Yi, Jupinder Parmar, Kai Xu, Kan Zhu, Kari Briski, Katherine Cheung, Katherine Luna, Keith Wyss, Keshav Santhanam, Kevin Shih, Kezhi Kong, Khushi Bhardwaj, Kirthi Shankar, Krishna C. Puvvada, Krzysztof Pawelec, Kumar Anik, Lawrence McAfee, Laya Sleiman, Leon Derczynski, Li Ding, Lizzie Wei, Lucas Liebenwein, Luis Vega, Maanu Grover, Maarten Van Segbroeck, Maer Rodrigues de Melo, Mahdi Nazemi, Makesh Narsimhan Sreedhar, Manoj Kilaru, Maor Ashkenazi, Marc Romeijn, Marcin Chochowski, Mark Cai, Markus Kliegl, Maryam Moosaei, Matt Kulka, Matvei Novikov, Mehrzad Samadi, Melissa Corpuz, Mengru Wang, Meredith Price, Michael Andersch, Michael Boone, Michael Evans, Miguel Martinez, Mikail Khona, Mike Chrzanowski, Minseok Lee, Mohammad Dabbah, Mohammad Shoeybi, Mostofa Patwary, Nabin Mulepati, Najeeb Nabwani, Natalie Hereth, Nave Assaf, Negar Habibi, Neta Zmora, Netanel Haber, Nicola Sessions, Nidhi Bhatia, Nikhil Jukar, Nikki Pope, Nikolai Ludwig, Nima Tajbakhsh, Nir Ailon, Nirmal Juluru, Nishant Sharma, Oleksii Hrinchuk, Oleksii Kuchaiev, Olivier Delalleau, Oluwatobi Olabiyi, Omer Ullman Argov, Omri Puny, Oren Tropp, Ouye Xie, Parth Chadha, Pasha Shamis, Paul Gibbons, Pavlo Molchanov, Pawel Morkisz, Peter Dykas, Peter Jin, Pinky Xu, Piotr Januszewski, Pranav Prashant Thombre, Prasoon Varshney, Pritam Gundecha, Przemek Tredak, Qing Miao, Qiyu Wan, Rabeeh Karimi Mahabadi, Rachit Garg, Ran El-Yaniv, Ran Zilberstein, Rasoul Shafipour, Rich Harang, Rick Izzo, Rima Shahbazyan, Rishabh Garg, Ritika Borkar, Ritu Gala, Riyad Islam, Robert Hesse, Roger Waleffe, Rohit Watve, Roi Koren, Ruoxi Zhang, Russell Hewett, Russell J. Hewett, Ryan Prenger, Ryan Timbrook, Sadegh Mahdavi, Sahil Modi, Samuel Kriman, Sangkug Lim, Sanjay Kariyappa, Sanjeev Satheesh, Saori Kaji, Satish Pasumarthi, Saurav Muralidharan, Sean Narentharen, Sean Narenthiran, Seonmyeong Bak, Sergey Kashirsky, Seth Poulos, Shahar Mor, Shanmugam Ramasamy, Shantanu Acharya, Shaona Ghosh, Sharath Turuvekere Sreenivas, Shelby Thomas, Shiqing Fan, Shreya Gopal, Shrimai Prabhumoye, Shubham Pachori, Shubham Toshniwal, Shuoyang Ding, Siddharth Singh, Simeng Sun, Smita Ithape, Somshubra Majumdar, Soumye Singhal, Stas Sergienko, Stefania Alborghetti, Stephen Ge, Sugam Dipak Devare, Sumeet Kumar Barua, Suseella Panguluri, Suyog Gupta, Sweta Priyadarshi, Syeda Nahida Akter, Tan Bui, Teodor-Dumitru Ene, Terry Kong, Thanh Do, Tijmen Blankevoort, Tim Moon, Tom Balough, Tomer Asida, Tomer Bar Natan, Tomer Ronen, Tugrul Konuk, Twinkle Vashishth, Udi Karpas, Ushnish De, Vahid Noorozi, Vahid Noroozi, Venkat Srinivasan, Venmugil Elango, Victor Cui, Vijay Korthikanti, Vinay Rao, Vitaly Kurin, Vitaly Lavrukhin, Vladimir Anisimov, Wanli Jiang, Wasi Uddin Ahmad, Wei Du, Wei Ping, Wenfei Zhou, Will Jennings, William Zhang, Wojciech Prazuch, Xiaowei Ren, Yashaswi Karnati, Yejin Choi, Yev Meyer, Yi-Fu Wu, Yian Zhang, Yigong Qin, Ying Lin, Yonatan Geifman, Yonggan Fu, Yoshi Subara, Yoshi Suhara, Yubo Gao, Zach Moshe, Zhen Dong, Zhongbo Zhu, Zihan Liu, Zijia Chen, and Zijie Yan. Nvidia nemotron 3: Efficient and open intelligence, 2025. URL https://arxiv.org/abs/2512.20856.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane

Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernandez. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534, Berlin, Germany, August 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P16-1144.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 30811–30849. Curran Associates, Inc., 2024. doi: 10.52202/079017-0970. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/370df50ccfdf8bde18f8f9c2d9151bda-Paper-Datasets_and_Benchmarks_Track.pdf.

Guilherme Penedo, Hynek Kydlíček, Vinko Sabolčec, Bettina Messmer, Negar Foroutan, Amir Hossein Kargaran, Colin Raffel, Martin Jaggi, Leandro Von Werra, and Thomas Wolf. Fineweb2: One pipeline to scale them all — adapting pre-training data processing to every language. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=jnRBe6zatP.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. YaRN: Efficient context window extension of large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=wHBfxhZu1u.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: an adversarial winograd schema challenge at scale. *Commun. ACM*, 64(9):99–106, August 2021. ISSN 0001-0782. doi: 10.1145/3474381. URL https://doi.org/10.1145/3474381.

Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. In Regina Barzilay and Min-Yen Kan (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1073–1083, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1099. URL https://aclanthology.org/P17-1099/.

Sho Takase, Jun Suzuki, and Masaaki Nagata. Direct output connection for a high-rank language model. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 4599–4609, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1489. URL https://aclanthology.org/D18-1489/.

Oskar van der Wal, Pietro Lesci, Max Müller-Eberstein, Naomi Saphra, Hailey Schoelkopf, Willem Zuidema, and Stella Biderman. Polypythias: Stability and outliers across fifty language model pre-training runs. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=bmrYu2Ekdz.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper_files/paper/2015/file/29921001f2f04bd3baee84a12e98098f-Paper.pdf.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *NUT@EMNLP*, 2017.

Wenhao Wu, Yizhong Wang, Guangxuan Xiao, Hao Peng, and Yao Fu. Retrieval head mechanistically explains long-context factuality. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=EytBpUGB1Z.

Yuhuai Wu, Markus Norman Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=TrjbxzRcnf-.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule, 2025. URL https://arxiv.org/abs/2412.06464.

Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W. Cohen. Breaking the softmax bottleneck: A high-rank RNN language model. In *International Conference on Learning Representations*, 2018a. URL https://openreview.net/forum?id=HkwZSG-CZ.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, Brussels, Belgium, October-November 2018b. Association for Computational Linguistics. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

Zhilin Yang, Thang Luong, Russ R Salakhutdinov, and Quoc V Le. Mixtape: Breaking the softmax bottleneck efficiently. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/512fc3c5227f637e41437c999a2d3169-Paper.pdf.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. Cacheblend: Fast large language model serving for rag with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, pp. 94–109, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400711961. doi: 10.1145/3689031.3696098. URL https://doi.org/10.1145/3689031.3696098.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472/.

Zexuan Zhong, Tao Lei, and Danqi Chen. Training language models with memory augmentation. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 5657–5673, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.382. URL https://aclanthology.org/2022.emnlp-main.382/.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models, 2023. URL https://arxiv.org/abs/2311.07911.

A Pretraining Hyperparameters

All pretraining runs follow the SmolLM2 recipe (Allal et al., 2025), and vanilla models and CPLMs share the exact same hyperparameters, listed in Table 4. The CP queries and keys use the hidden dimension D unless stated otherwise.

	135M	**360M**	**1.7B**
Architecture			
Layers	30	32	24
Hidden size	576	960	2,048
FFN size	1,536	2,560	8,192
Attention heads	9	15	32
KV heads (GQA)	3	5	32
Head dim	64	64	64
Vocabulary size	49,152	49,152	49,152
Tied embeddings	Yes	Yes	Yes
Optimization & schedule			
Sequence length	2,048	2,048	4,096
Batch size (sequences)	512	512	512
Training steps	19,000	19,000	95,000
Training tokens	20 B	20 B	200 B
Optimizer	AdamW	AdamW	AdamW
Peak learning rate	3×10^{-3}	3×10^{-3}	5×10^{-4}
LR schedule	WSD	WSD	WSD
Warmup steps	500	500	2,000
Cooldown steps	1,900	1,900	4,750
Weight decay	0.01	0.01	0.01
Adam (β_1, β_2)	(0.9, 0.95)	(0.9, 0.95)	(0.9, 0.95)
Gradient clipping	1.0	1.0	1.0
Precision	`bf16-mixed`	`bf16-mixed`	`bf16-mixed`

Table 4: Pretraining hyperparameters for the 135M, 360M and 1.7B runs.

Full-attention and hybrid models use YaRN (Peng et al., 2024) with a $\times 4$ scaling factor from their 4,096-token pretraining context, applied to all layers: at inference only for the base models, and already during fine-tuning for the SFT models. SW-256 models keep plain RoPE, since they never see positions more than 256 tokens back.

B Latency and Memory

The CP head adds two $D \times D$ projections and a sink key to the backbone, i.e. 0.5% more parameters at 1.7B and 9B, and 0.2% at 30B. The 9B and 30B models are untrained architectures, only used to measure time and memory. We measure what this costs in time and memory on a single NVIDIA B300 GPU in `bfloat16`, comparing each CPLM with a vanilla model with the same backbone. We focus on a full-attention model for this analysis.

Training. Table 5 reports training throughput and peak memory with sequences of 4,096 tokens. The CP head distribution and its contribution to the loss are computed with a fused kernel, so that CPLMs never materialize the $T \times T$ copy distribution over the vocabulary. At 1.7B, CPLMs train at 97% of the vanilla throughput, and the gap closes with scale: at 9B and 30B, the difference is within 1%, i.e. at the level of run-to-run variation. This is expected, as the cost of the CP head grows with the hidden dimension and the sequence length but not with depth, so that it becomes a smaller share of each training step as models grow. Peak memory increases by 2.4–3.0 GB at 9B and 30B (2–3%). At 1.7B, computing this contribution over the 32k tokens of the batch at once adds 12.1 GB, and computing it in chunks of 8,192 positions reduces this overhead to 3.4 GB (+2.7%).

	Throughput (tokens/s)			Peak memory (GB)		
Model	Vanilla	CPLM	Ratio	Vanilla	CPLM	Ratio
1.7B	55,862	54,113	×0.969	125.4	137.5	×1.096
9B	20,429	20,533	×1.005	114.0	117.0	×1.026
30B	5,391	5,342	×0.991	139.9	142.3	×1.017

Table 5: Training cost on one B300 GPU with sequences of 4,096 tokens. Ratios are CPLM values divided by vanilla values. At 1.7B, a training step includes the AdamW update (batch of 8 sequences); at 9B and 30B, only the forward and backward passes fit on one GPU.

Inference. At inference time, CPLMs cache one copy key per token, next to the key–value cache. During prefill, the copy keys of the prompt are written to this cache and the copy distribution is only computed for the last position, so that prefill never forms the $T \times T$ copy scores. Each decoding step then adds a single dot product between the copy query and each cached copy key, followed by the mixture with the LM head distribution. We measure generation with a static cache, FlexAttention and a decoding step compiled into a CUDA graph, prefilling prompts of 4k or 16k tokens and greedily generating 256 tokens (Table 6).

Prefill time is unchanged: all differences are within $\pm 1.3\%$ and go in both directions. A decoding step is 1.0–3.8% slower. Since decoding is limited by memory bandwidth, the extra time is essentially that of reading the cached copy keys. It is largest when the key–value cache dominates the decoding step (large batches of long sequences), and smallest when reading the weights does (small batches). As prefill comes at no cost, the overhead per request is even smaller: for a prompt of 4k to 16k tokens followed by 256 generated tokens, CPLMs add 0.8–1.7% to the total time.

The copy cache holds D values per token, against $2\, n_{\text{layers}}\, n_{\text{kv}}\, d_{\text{head}}$ for the key–value cache, i.e. 2.1% of the key–value cache at 1.7B and 6.2% at 9B, and decoding memory grows by exactly this amount.

			Time			Memory (GB)	
Model	T	B	Prefill	Decoding step	Request	KV cache	Copy cache
1.7B	4k	1	×0.993	×1.018	×1.017	0.80	0.02
	4k	128	×0.999	×1.017	×1.010	102.1	2.13
	16k	32	×0.996	×1.018	×1.008	97.5	2.03
9B	4k	1	×0.987	×1.010	×1.008	0.53	0.03
	4k	128	×1.000	×1.038	×1.014	68.1	4.25
	16k	32	×1.001	×1.027	×1.011	65.0	4.06

Table 6: Inference cost on one B300 GPU, for prompts of T tokens and batches of B sequences. Time columns give the ratio of the CPLM time to the vanilla time for prefill, for a decoding step, and for a whole request (prefill and 256 generated tokens), so that values above 1 mean that the CPLM is slower. Memory columns give the size of the key–value cache and of the additional copy cache of the CPLM in `bfloat16`.

Overall, the cost of CPLMs is of the same small order in training and at inference: at most a few percent of throughput, less than 1% at 9B and 30B in training, and memory overheads of a few percent.

C Ablations

C.1 Multi-Seed Robustness

We repeat the 135M/20B runs with three seeds (42, 43, 44) to confirm the copy gains exceed seed variance. Each arm uses the same recipe as described in Appendix A with a window size of 256 where it applies; the only change across runs is the random seed. Table 7 reports final validation perplexity (mean $\pm$ standard deviation over the three seeds) for each backbone, with and without the

CP head. CPLM lowers perplexity for every backbone, and each gain is roughly 15–40$\times$ the seed standard deviation. A one-sided paired t-test gives a p-value $p < 10^{-3}$ in all three cases.

Backbone	Vanilla	CPLM	Δ	p
Full	16.33 ± 0.03	15.83 ± 0.02	-0.50	3.8×10^{-4}
SW	16.89 ± 0.02	16.02 ± 0.02	-0.87	9.5×10^{-5}
Hybrid 3:1	16.37 ± 0.02	15.84 ± 0.04	-0.52	2.5×10^{-4}

Table 7: Final validation perplexity at 135M (mean $\pm$ std over seeds 42/43/44). Δ is CPLM minus Vanilla.

C.2 Copy Dimension

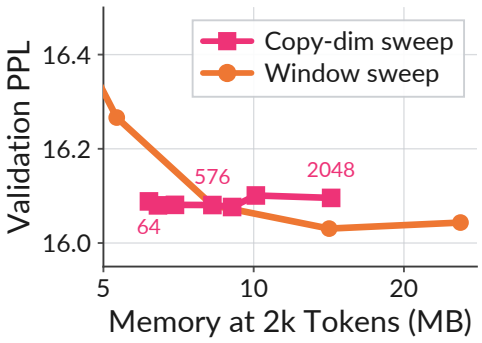


Figure 8: Validation perplexity vs. cache memory at 2k tokens for our 135M sliding-window CPLM ($w = 256$), when varying the dimension of the CP queries and keys from 64 to 2,048 (*Copy-dim sweep*, default $D = 576$), or the backbone window size (*Window sweep*).

By default, the CP queries and keys use the hidden dimension D. Figure 8 shows that this dimension barely matters: with copy dimensions ranging from 64 to 2,048, the validation perplexity of our 135M sliding-window CPLM stays within 0.02 points, while the memory needed to cache the copy keys varies substantially. A copy dimension of 64 thus stores 9$\times$ fewer values per token than the default $D = 576$ for the same performance. This makes the copy dimension a much cheaper knob than the window size: reducing the window to reach the same memory budget increases perplexity by up to 0.2 points. Small copy dimensions would also further reduce the memory of the copy cache in retrieval settings (Section 3.4). For the sake of simplicity, we use the default D for our larger pretraining runs, and leave the exploration of this phenomenon for future work.

C.3 360M Scale

Figure 9 reports the training dynamics of our 360M models, trained with the same data and token budget as our 135M models (Table 4). The results are the same as at 135M and 1.7B scale (Figure 2): CPLMs improve over their vanilla counterparts for every backbone, and the SW CPLM outperforms the vanilla Full and Hybrid models.

C.4 Sink and Window Size

Figure 10 compares CPLMs with and without the sink across backbone window sizes. The sink consistently helps when the window is small, lowering perplexity by about 0.3 points with a 32-token window and by 0.07 points with a 256-token window. The gap vanishes for windows of 512 tokens or more and for full attention. We hypothesize that, with a small window, the backbone has less information about whether the next token can be found in the context, so that the ability to defer back to the LM head after triggering the CP head becomes more valuable.

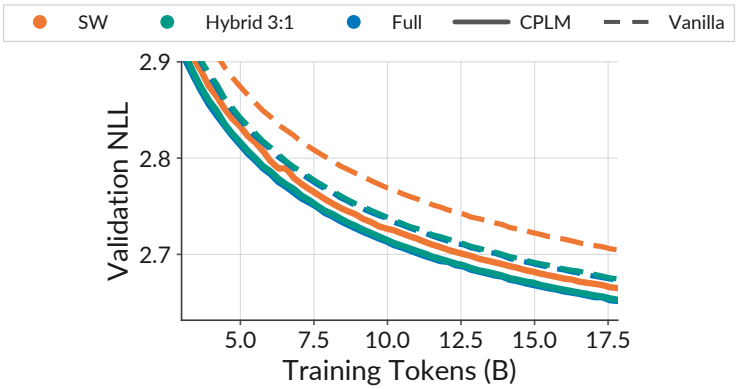


Figure 9: FineWeb-Edu validation loss vs. training tokens (before LR decay) at 360M scale, for $w = 256$. Full and Hybrid training curves are essentially superposed.

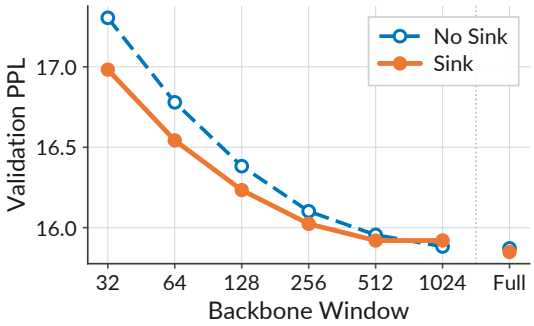


Figure 10: Validation perplexity of 135M CPLMs with and without the sink mechanism, for sliding-window backbones with windows from 32 to 1,024 tokens and for a full-attention backbone.

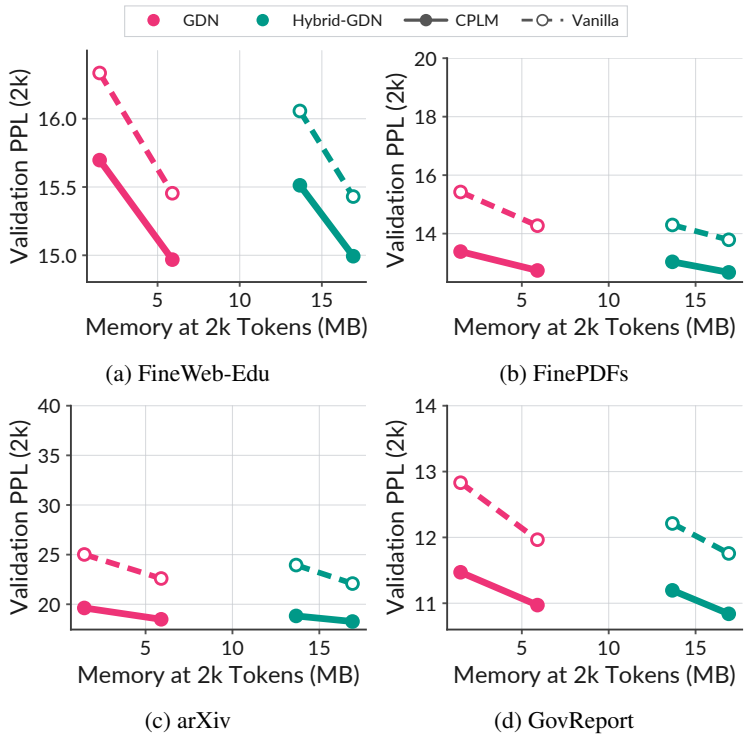
C.5 Gated DeltaNet Pretraining

We replace the softmax-attention mixer with a Gated DeltaNet (GDN) layer (Yang et al., 2025), a linear-attention recurrence whose state is constant in sequence length, keeping the rest of the backbone, the Copy-Paste head, and the training recipe identical to the other 135M experiments. Similarly to sliding-window attention, we train two variants—a fully-recurrent GDN model and a hybrid that keeps one attention layer for every three GDN layers—each with and without the CP head mechanism, for 20B tokens on the same doc-masked data. Even though we match the hyperparameters of the other 135M runs, this comparison is not parameter-matched: the GDN mixer is larger than an attention block (1.78M vs. 0.88M parameters per layer), so the fully-recurrent model carries about 20% more parameters than the corresponding attention arm (161M vs. 134.5M), which would confound direct comparison.

Figure 11 shows that the CP head also benefits recurrent backbones. For both GDN and Hybrid-GDN models, and for both state sizes, CPLMs have a lower perplexity than their vanilla counterparts on every dataset, by about 0.5 points on in-domain FineWeb-Edu and by 1 to 6 points on out-of-distribution data. As with sliding-window attention (Figure 3), a fully-recurrent GDN CPLM outperforms the vanilla Hybrid-GDN models on out-of-distribution data, while using several times less memory.

C.6 CP Head Ablations

We compare ways of combining a pointer over past tokens with the LM head. Unless stated otherwise, every variant uses the same backbone, and the same queries q_t, keys k_τ and copy attention $\alpha_t(\tau)$ over the context positions $\tau \leq t$ as the CP head (Equation (2)), with $p_{\mathrm{CP}}(v \mid x_{\leq t})$ defined



(a) FineWeb-Edu (b) FinePDFs

(c) arXiv (d) GovReport

Figure 11: Validation perplexity vs. memory usage at 135M scale for fully-recurrent (GDN) and hybrid (Hybrid-GDN) Gated DeltaNet backbones, with and without the CP head, on in-domain (FineWeb-Edu) and out-of-distribution (FinePDFs, arXiv, GovReport) data. Each architecture is shown with GDN state sizes of 64 and 256, the larger state using more memory.

as in Equation (3). Variants only differ in how the LM head and the pointer are combined into the next-token distribution $p(v \mid x_{\le t})$.

Sigmoid gate. As in pointer-generator networks (See et al., 2017), the two distributions are mixed with a separate gate $g_t = \sigma(w_g^\top h_t)$:

$$p(v \mid x_{\le t}) \;=\; (1 - g_t)\, p_{\mathrm{LM}}(v \mid x_{\le t}) \;+\; g_t\, p_{\mathrm{CP}}(v \mid x_{\le t}),$$

and the <copy> token is not used.

Shared softmax. Following CopyNet (Gu et al., 2016), the vocabulary logits and the copy scores share a single normalizer:

$$p(v \mid x_{\le t}) \;=\; \frac{1}{Z_t}\Big(\exp\left(w_v^\top h_t\right) + \sum_{\tau \le t\,:\,x_\tau = v} \exp\left(k_\tau^\top q_t/\sqrt{D}\right)\Big),$$

$$Z_t \;=\; \sum_{u \in \mathcal{V}} \exp\left(w_u^\top h_t\right) + \sum_{\tau \le t} \exp\left(k_\tau^\top q_t/\sqrt{D}\right).$$

There is no explicit gate: the share of copy mass is set by the scale of the copy scores relative to the vocabulary logits, and it grows with the number of visible positions. We keep our dot-product copy scores and omit the selective-read state update of CopyNet.

Chang et al. (2023) partitions. Following Chang et al. (2023), the vocabulary is split into partitions, scored with separate projections of the hidden state inside a single softmax. The base logit of token v is $L_V(h_t)^\top w_v$. The context partition (C) rescores the tokens present in the context with $L_C(h_t)^\top w_v$. The pointer partition (P) adds $L_{PD}(h_t)^\top e_v$ for the tokens present in the context, where e_v averages $L_{LD}(h_\tau)$ over the past occurrences τ of v. The reranker (R) rescores the top-20 tokens

Method	Output mechanism	Validation		FinePDFs @2k	
		Full	SW-256	Full	SW-256
Vanilla	LM softmax	16.34	16.88	14.25	16.52
Chang et al. (2023)					
C	Context partition	16.25	16.71	13.88	15.66
P	Pointer partition	16.19	16.54	13.81	14.96
CP	Context + pointer	16.19	16.53	13.64	14.91
R	Top-k reranker	16.58	17.12	14.28	16.61
CPR	Context + pointer + reranker	16.33	16.62	13.75	14.85
Pointer variants					
Sigmoid gate	Mixture, separate gate $\sigma(W_g h)$	15.94	16.20	13.39	14.25
Shared softmax	Single normalizer (CopyNet-style)	15.88	16.09	13.32	**13.82**
CPLM					
w/o sink	`<copy>`-gated mixture	15.87	16.10	13.32	14.13
w/ sink (ours)	`<copy>`-gated mixture + sink	**15.85**	**16.02**	**13.31**	**13.82**

Table 8: Output-mechanism ablations at 135M (20B tokens, seed 42). Perplexity for a full-attention and a sliding-window (256) backbone, on the in-distribution validation set (final) and on held-out FinePDFs documents (2,000 English documents, first 2,048 tokens each, whitespace- and line-break-normalized). All runs share the backbone, data and training recipe. Across three seeds the standard deviation of the main arms on validation is 0.015–0.037 PPL (Appendix C.1).

that are not in the context with $L_R(h_t)^\top w_v$. CP and CPR combine these partitions. All variants are trained with cross-entropy and no copy supervision. The original work fine-tunes a pretrained model with a multi-layer augmented hidden state, pointer projections initialized at $10^{-10}I$, and two rerankers. To pretrain from scratch, we use the final hidden state, initialize the pointer projections at $10^{-3}I$ so that the pointer engages within our training budget, and use a single reranker.

Remark on decoding cost. Our gate is a token of the LM head, so the amount of copying is known before any copy attention is computed. Sampling can therefore proceed in two stages: sample from the LM head over $\mathcal{V} \cup \{\texttt{<copy>}\}$, and only compute the copy attention when `<copy>` is sampled, falling back to p_{LM} if the sink is then sampled. This is exact sampling from the mixture of Equation (7). For greedy decoding, copying adds at most $\lambda_t \le c_t$ to the probability of any token, while scaling the LM head distribution by $1 - \lambda_t$. Hence, if the two largest probabilities $p_1 \ge p_2$ of $p_{\mathrm{LM}}(\cdot \mid x_{\le t})$ satisfy $(1-c_t)(p_1-p_2) \ge c_t$, the argmax of p_{LM} is also the argmax of the mixture, and the copy attention can be skipped. The shared softmax and the partitions of Chang et al. (2023) have no such signal: their normalizer or logits depend on the copy scores, so that the copy computation is needed at every decoding step.

Results. Table 8 reports final perplexity. The copy-gated mixture with a sink matches or slightly improves over the other pointer variants on both backbones and evaluation sets, and ties with the shared softmax on FinePDFs with the sliding-window backbone. The partitions of Chang et al. (2023) remain behind every pointer variant; except for the reranker, which underperforms the vanilla model, they improve over the vanilla model, with CP the strongest. The margins between our method and the other pointer variants are small, within about one to two seed standard deviations (Appendix C.1). Note that the pointer-generator gate (See et al., 2017) and CopyNet (Gu et al., 2016) were designed for encoder-decoder models that copy from a source sequence, and we adapt them here to copying from the past context of a decoder-only LM. These results are consistent with our framing: our contribution is not the specific pointer design, but showing that such a head substantially improves LM pretraining across backbones, mostly by mitigating the LM head gradient bottleneck (Section 4.1 and Appendix H). We use the `<copy>` gate and the sink for convenience rather than performance. By construction, the gate allows exact two-stage sampling that only computes the copy attention when needed (see the remark above), and the gate and sink probabilities expose the decision to copy and the availability of a suitable position, which we use in our analyses (Figures 4 and 14).

C.7 Comparison with a Neural Cache

A natural question is whether the gains of CPLMs could be obtained without training, by adding a context cache to a vanilla LM. We compare against the neural cache of Grave et al. (2017), applied to each vanilla backbone. At position t, the cache scores every earlier position τ by the similarity between the final hidden states h_t and h_τ, and proposes the token that followed τ:

$$p_{\text{cache}}(v \mid x_{\le t}) \;\propto \sum_{\tau < t \,:\, x_{\tau+1} = v} \exp\left(\theta\, h_t^\top h_\tau\right).$$

The next-token distribution is then $(1-\gamma)\, p_{\text{LM}} + \gamma\, p_{\text{cache}}$. We tune θ and γ for each backbone on 100 held-out FineWeb-Edu documents, and evaluate on 100 different documents of 2,048 tokens (Table 9).

Attention	Vanilla	Vanilla + Neural cache	CPLM
Full	9.17	9.17	**9.06**
Hybrid 3:1	9.20	9.20	**9.06**
SW-256	9.65	9.63	**9.24**

Table 9: FineWeb-Edu perplexity of our 1.7B models (100 documents of 2,048 tokens). The neural cache (Grave et al., 2017) is added to the vanilla model, with θ and γ tuned on separate FineWeb-Edu documents.

The tuned neural cache barely changes the perplexity of the vanilla models: the best mixture weight is $\gamma \le 0.002$ for all three backbones, i.e. the cache is almost switched off. CPLMs, which learn when and where to copy, lower perplexity by 0.11 (Full), 0.14 (Hybrid) and 0.41 (SW). The gains of CPLMs therefore come from training the copy mechanism jointly with the LM, not from having access to a context cache.

D SFT Evaluation

SFT uses sequences of up to 16k tokens. All SFT evaluations use the chat template seen during fine-tuning, except MATH, which uses its standard plain prompt. Full and Hybrid models use YaRN RoPE scaling (factor 4); SW-256 models use standard RoPE.

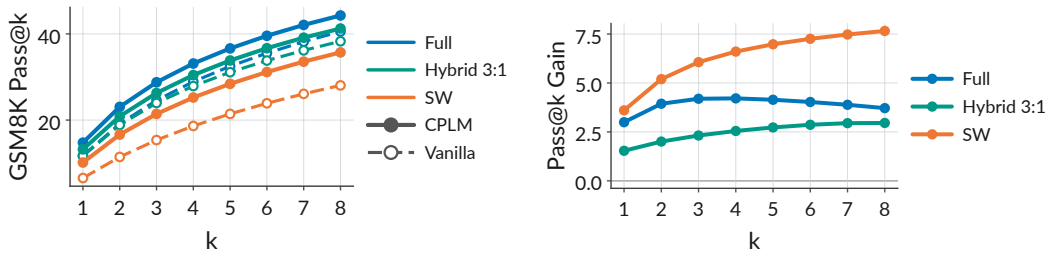
Perplexity. *Tulu-3* is the validation perplexity at the end of SFT. *Nemotron-3* is the perplexity on assistant tokens at positions 7.5k–8.5k, within the 16k-token context used for SFT, of 4,000 randomly sampled conversations from Nemotron-SFT-Math-v4, with reasoning traces included in the assistant turns.

MATH. Level 1–2 problems from four subjects of MATH (Hendrycks et al., 2021) (Algebra, Counting & Probability, Number Theory, Prealgebra; 861 problems), with the Minerva 4-shot prompt (Lewkowycz et al., 2022). We only use levels 1 and 2, as harder problems are out of reach for models of this size. We report pass@4 over 8 samples (temperature 0.7, top-p 0.9).

GSM8K. 5-shot GSM8K (Cobbe et al., 2021), pass@4 over 8 samples (temperature 0.7, top-p 0.9). The answer is the last number in the completion.

HumanEval and MBPP. Zero-shot HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021): the task is given in the user turn and the model writes the full function. We report pass@10 over 20 samples (temperature 0.8, top-p 0.95).

IFEval. Zero-shot IFEval (Zhou et al., 2023) with a repetition penalty of 1.1 and nucleus sampling (temperature 0.9, top-p 0.95). We report the mean of the prompt- and instruction-level accuracies under both strict and loose criteria.



(a) Pass@k (b) Pass@k gain of CPLMs

Figure 12: GSM8K pass@k of our 1.7B SFT models for k from 1 to 8, estimated from 8 samples (temperature 0.7, top-p 0.9). **(a)** Pass@k of vanilla models and CPLMs. **(b)** Difference between each CPLM and its vanilla counterpart, in points.

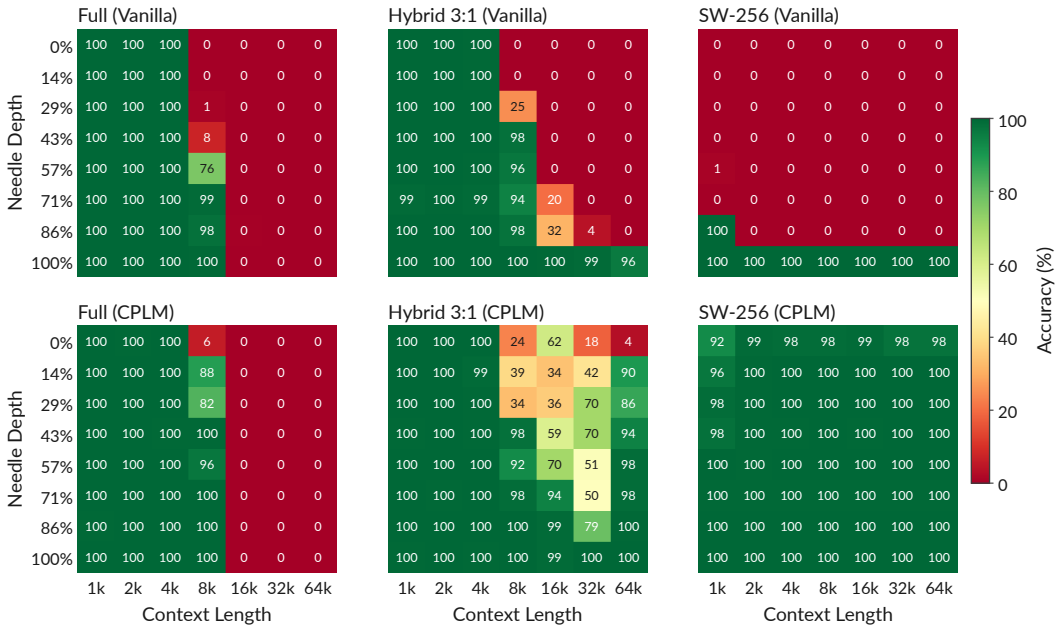


Figure 13: Needle-in-a-haystack accuracy (%) by needle depth (rows) and context length (columns). Top: Vanilla. Bottom: CPLM. Each cell averages 200 examples. Average over the full grid, Vanilla / CPLM: Full 49.7 / 54.8; Hybrid 3:1 58.2 / 83.6; SW-256 14.3 / 99.4.

Pass@k consistency. We find our pass@k results to be consistent across k. For instance, on GSM8K (Figure 12), CPLMs outperform their vanilla counterparts at every k from 1 to 8, including pass@1, where they improve accuracy by 1.6 to 3.6 points. The gain grows with k for the SW and Hybrid CPLMs, reaching 7.7 points at pass@8 for the SW CPLM, while it stays around 4 points for the Full CPLM.

E NEEDLE-IN-A-HAYSTACK

Setup. We build each prompt from 15 neutral filler sentences, repeated to fill the context. A needle ("The special magic number is: N.", with N a random 7-digit integer) is inserted at relative depth $d \in \{0, \frac{1}{7}, \ldots, 1\}$. The prompt ends with the question "What is the special magic number?" We test context lengths $L \in \{1\text{k}, 2\text{k}, 4\text{k}, 8\text{k}, 16\text{k}, 32\text{k}, 64\text{k}\}$ and use 200 examples per (d, L) cell. The model decodes greedily for up to 16 tokens. An answer counts as correct if the output contains N. For the Full and Hybrid 3:1 models, we apply YaRN ($\times 4$) to the full-attention layers at every length, so each model uses a single configuration across the sweep. Sliding-window layers keep the base RoPE, since they never see a relative offset above 255.

Results. Up to the 4k pretraining length, Full and Hybrid models score close to 100% at every depth, with or without CPLM. The architectures differ beyond that point.

SW-256. The Vanilla model only finds the needle when it is inside the last 256 tokens. That is the 100%-depth row, plus the 86% row at 1k. Every other cell is at 0, so accuracy is stuck at $1/8$ of the depths for $L \geq$ 2k. With CPLM, the same backbone gets 98–100% in almost every cell up to 64k. The only drop is a small one at depth 0 and at 1k.

Hybrid 3:1. Past 8k, the Vanilla Hybrid behaves like a sliding-window model. It still finds needles at the end of the context, but accuracy on earlier needles falls to 0. Its average per length drops from 64 at 8k to 12 at 64k. With CPLM, 60–84% accuracy per length remains from 16k to 64k, and the deeper needles are recovered best. The shallowest needles ($d \leq 2/7$) are the hardest.

Full. Both Full models fail completely from 16k on, even with YaRN. Retrieval does not generalize much beyond $2\times$ the training length. CPLM helps in the transition region: at 8k, accuracy rises from 48 to 84, and Vanilla's weak rows at shallow-to-middle depths are mostly recovered. CPLM does not move the failure point.

In summary, CPLM helps most when the architecture cannot attend over the full context. When the backbone's own attention fails beyond the window or past the training length, the copy pathway keeps long-range retrieval working.

F Long-Context Evaluation

Table 10 details the LongBench results of our base 1.7B models. The largest gains come from TREC, a few-shot classification task where the answer labels can be copied from the in-context demonstrations: CPLMs improve it by $+16.0$, $+29.0$ and $+24.5$ points for Full, SW and Hybrid models, respectively. However, CPLMs also improve on tasks where copying is not crucial. On the three summarization tasks, whose outputs are abstractive, SW and Hybrid CPLMs improve over their vanilla counterparts on every task, by $+3.6$ to $+6.2$ points, although the Full CPLM is behind its vanilla counterpart on GovReport and QMSum.

LongBench after SFT. Table 11 reports the same benchmark after supervised fine-tuning. CPLMs still improve the average score for every backbone ($+0.6$, $+2.7$ and $+1.9$ points for Full, SW and Hybrid models), but the gaps are smaller than for base models and less uniform across tasks. SW and Hybrid CPLMs keep large gains on TREC, while the Full CPLM is behind its vanilla counterpart on this task. Summarization gains are more consistent: CPLMs improve on GovReport and MultiNews for every backbone. Conversely, the SW CPLM loses ground on SAMSum and on code completion after fine-tuning.

RULER and BABILong by context length. Table 12 extends the synthetic evaluations of Table 3 to 64k tokens. For Full and Hybrid models, CPLMs are better at almost every length up to 16k, and the Hybrid CPLM keeps an advantage on RULER at 32k. Beyond 16k, both Full models fail completely, which shows that the CP head does not compensate for the length generalization failure of the backbone. The SW CPLM improves RULER accuracy at every length, from 10.2 to 36.1 at 4k and from 5.0 to 18.9 at 64k, while both SW models remain close to the floor on BABILong beyond 2k tokens.

Per-task results at 4k. Tables 13 and 14 break down RULER and BABILong at 4k tokens. For the SW CPLM, the RULER gain mostly comes from single-needle retrieval, which it solves almost perfectly (S1 and S2) while its vanilla counterpart cannot see beyond its window, and from word extraction tasks (CWE and FWE). For Full and Hybrid models, CPLMs improve on single and multi-key retrieval, multi-value retrieval and QA, but not on every task: the Full CPLM is behind on FWE and SQuAD, and the Hybrid CPLM on CWE and FWE. The hardest multi-key tasks (MK2 and MK3) remain near the floor for all models. On BABILong, the Full CPLM improves on all five tasks ($18.6 \rightarrow 32.6$ on average), while the Hybrid CPLM is on par with its vanilla counterpart on average, and both SW models are close to the floor.

	Single-Doc QA			Multi-Doc QA			Summarization			Few-shot			Synthetic		Code		
	NarrativeQA	*Qasper*	*MultiFieldQA*	*HotpotQA*	*2WikiMQA*	*MuSiQue*	*GovReport*	*QMSum*	*MultiNews*	*TREC*	*TriviaQA*	*SAMSum*	*PassageCount*	*PassageRet*	*LCC*	*RepoBench-P*	**Avg.**
Full	1.0	3.7	12.4	6.3	9.0	3.1	**16.6**	**7.8**	11.1	39.0	20.8	2.2	0.1	**3.3**	15.7	15.0	10.4
+ CP	**2.6**	**5.1**	**12.9**	**6.4**	**9.6**	**4.1**	11.0	6.2	**13.2**	**55.0**	**26.0**	**4.0**	**1.0**	0.6	**20.0**	**17.1**	**12.2** +1.7
SW-256	0.7	1.4	6.9	1.6	6.0	1.0	5.5	5.6	8.1	25.5	**24.0**	0.3	0.4	2.4	14.5	15.1	7.4
+ CP	**2.7**	**5.8**	**14.7**	**4.0**	**6.2**	**3.0**	**9.1**	**11.1**	**12.1**	**54.5**	22.0	**14.5**	**0.7**	**3.4**	**27.2**	**22.7**	**13.4** +5.9
Hybrid 3:1	0.9	2.1	**10.8**	**4.5**	**8.4**	**3.1**	10.5	4.7	10.0	36.0	25.2	0.2	0.5	0.9	17.2	17.7	9.5
+ CP	**1.9**	**5.6**	8.6	3.1	5.3	1.7	**15.9**	**10.9**	**15.1**	**60.5**	**27.7**	**17.2**	**0.9**	**2.5**	**24.3**	**27.0**	**14.3** +4.7

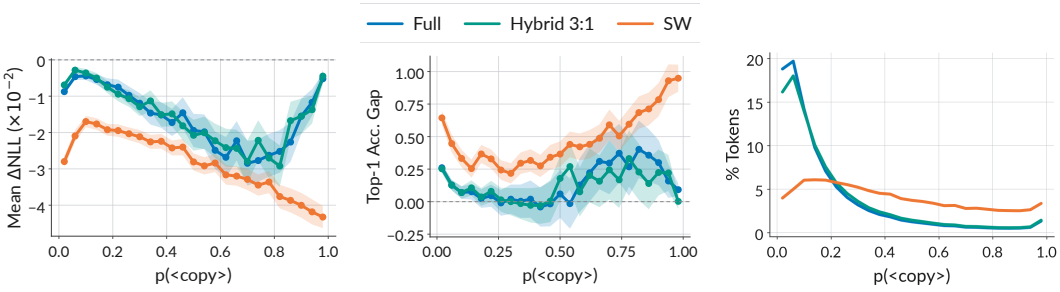
Table 10: **LongBench (English, 16 tasks), base models.** All 1.7B models, full-length contexts, no instruction tuning. RoPE: YaRN×4 for Full and Hybrid, base RoPE for SW-256 (YaRN degrades the purely-local arms). **Bold** marks the better of each vanilla/CPLM pair; green is the CP gain in average score.

	Single-Doc QA			Multi-Doc QA			Summarization			Few-shot			Synthetic		Code		
	NarrativeQA	*Qasper*	*MultiFieldQA*	*HotpotQA*	*2WikiMQA*	*MuSiQue*	*GovReport*	*QMSum*	*MultiNews*	*TREC*	*TriviaQA*	*SAMSum*	*PassageCount*	*PassageRet*	*LCC*	*RepoBench-P*	**Avg.**
Full	12.8	20.7	24.5	**27.1**	**23.6**	6.5	22.2	**20.5**	19.0	**15.5**	66.7	27.7	0.0	4.5	19.5	21.3	20.7
+ CP	**14.8**	**23.6**	**27.4**	23.7	18.0	**10.7**	**24.7**	19.8	**22.5**	8.0	**68.1**	**30.4**	**1.5**	4.5	**20.9**	**23.7**	**21.4** +0.6
SW-256	10.0	9.8	16.5	14.8	**21.2**	**8.1**	13.7	19.3	12.4	22.0	40.2	**17.2**	0.5	**5.5**	**16.9**	**18.9**	15.4
+ CP	**10.8**	**11.9**	**20.2**	**22.5**	18.0	5.7	**18.6**	**20.5**	**14.7**	**59.0**	**46.5**	6.2	**2.4**	4.2	14.5	14.3	**18.1** +2.7
Hybrid 3:1	**14.7**	18.3	25.3	24.7	24.0	**10.1**	22.1	**20.2**	14.8	1.0	**64.1**	23.8	0.1	5.0	17.2	**22.9**	19.3
+ CP	13.0	**21.5**	**28.4**	**24.9**	24.0	8.1	**24.8**	19.0	**18.0**	**20.5**	60.1	**29.1**	**2.4**	5.0	**22.0**	17.1	**21.1** +1.9

Table 11: **LongBench (English, 16 tasks), after SFT.** All 1.7B models, instruction-tuned at lr 1×10^{-4}, 16k contexts. **Bold** marks the better of each vanilla/CPLM pair; green is the CP gain in average score.

G Analyzing CPLMs

G.1 Copy Probability and Prediction Quality



(a) NLL gap (b) Top-1 accuracy gap (c) Distribution of c_t

Figure 14: Prediction quality as a function of the copy gate $c_t = p(\texttt{<copy>})$ of our 1.7B CPLMs, on the FineWeb-Edu validation set. Shaded areas show the standard deviation within each bin. **(a)** Mean NLL difference between each CPLM and its vanilla counterpart, computed as CPLM minus vanilla (negative values favor CPLMs) and **(b)** top-1 accuracy gap (positive values favor CPLMs), computed on the tokens of each bin of c_t. **(c)** Share of tokens in each bin.

We now study whether the gains of CPLMs are limited to the tokens that they copy. We bin tokens according to the probability c_t that the CPLM assigns to <copy>, and compare the NLL and top-1 accuracy of CPLMs and vanilla models within each bin (Figure 14). In every bin and for every backbone, CPLMs reach a lower NLL than their vanilla counterparts (Figure 14a). This includes

		RULER						BABILong								
Attention	Model	4k	8k	16k	32k	64k	**Avg**	0k	1k	2k	4k	8k	16k	32k	64k	**Avg**
Full	Vanilla	50.1	36.3	14.5	0.0	0.0	20.2	**55.0**	44.6	35.0	18.6	4.4	0.6	0.0	0.0	19.8
	CPLM	**54.9**	**38.6**	**25.2**	0.0	0.0	**23.7**	52.6	**49.6**	**40.8**	**32.6**	**12.6**	**4.0**	0.0	0.0	**24.0**
Hybrid 3:1	Vanilla	50.6	32.1	16.9	7.1	**3.6**	22.1	56.4	38.8	29.0	26.8	11.4	3.6	0.6	0.2	20.8
	CPLM	**53.5**	**40.8**	**19.2**	**11.7**	2.8	**25.6**	**57.6**	**47.4**	**42.8**	**27.4**	**16.2**	**5.8**	**1.4**	0.2	**24.8**
SW-256	Vanilla	10.2	7.5	5.9	5.5	5.0	6.8	46.2	18.2	10.2	5.2	1.8	**1.0**	0.4	0.6	10.5
	CPLM	**36.1**	**27.6**	**23.9**	**21.2**	**18.9**	**25.5**	**49.6**	**22.6**	**13.0**	**8.4**	**2.2**	0.6	**0.8**	0.6	**12.2**

Table 12: RULER and BABILong accuracy (%) by context length for the 1.7B base models. RULER: mean over its 13 tasks; BABILong: mean over qa1–qa5. Avg is the mean over all lengths shown (4k–64k for RULER, 0k–64k for BABILong). Full and Hybrid use YaRN (×4 from 4096); SW-256 uses plain RoPE. **Bold** marks the better model of each pair.

		Retrieval (NIAH)								Tracing	Aggregation		QA		
Attention	Model	S1	S2	S3	MK1	MK2	MK3	MQ	MV	VT	CWE	FWE	SQuAD	HotpotQA	**Avg**
Full	Vanilla	99.0	100.0	68.0	62.0	**6.0**	**9.0**	**83.0**	64.8	0.0	34.7	**53.3**	**43.0**	28.0	50.1
	CPLM	**100.0**	100.0	**85.0**	**67.0**	4.0	8.0	80.8	**72.2**	**26.6**	**60.8**	40.3	39.0	**30.0**	**54.9**
Hybrid 3:1	Vanilla	**100.0**	97.0	62.0	62.0	6.0	5.0	**82.8**	73.2	**9.0**	**51.6**	**42.3**	38.4	29.0	50.6
	CPLM	99.0	**100.0**	**70.0**	**90.0**	**8.0**	**8.0**	82.5	**79.0**	7.0	38.9	38.0	**42.3**	**33.0**	**53.5**
SW-256	Vanilla	8.0	9.0	3.0	8.0	3.0	1.0	5.2	3.2	**0.4**	16.0	26.0	31.3	19.0	10.2
	CPLM	**100.0**	**99.0**	**36.0**	**39.0**	3.0	**7.0**	**16.5**	**20.2**	0.0	**44.4**	**48.0**	**32.0**	**24.0**	**36.1**

Table 13: RULER per-task accuracy (%) at 4k context for the 1.7B base models. NIAH: single-needle (S1–S3), multi-key (MK1–MK3), multi-query (MQ), multi-value (MV); VT: variable tracking; CWE/FWE: common/frequent word extraction. Avg is the mean over the 13 tasks. Full and Hybrid use YaRN (×4 from 4096); SW-256 uses plain RoPE. **Bold** marks the better model of each pair.

tokens for which c_t is close to 0, where the prediction relies almost entirely on the LM head: the CP head also improves the predictions of the LM head itself, which is consistent with a better-trained backbone. The gains mostly grow with c_t, as the CP head contributes more to the prediction, and are largest for the SW CPLM, which cannot rely on its backbone to reach tokens beyond its window. Top-1 accuracy gains are also largest for tokens that are likely to be copied, especially for the SW CPLM, while Full and Hybrid CPLMs are on par with their vanilla counterparts for intermediate values of c_t (Figure 14b). Finally, Full and Hybrid CPLMs assign a low copy probability to most tokens, while the SW CPLM uses the CP head more uniformly (Figure 14c).

G.2 Copy Distance

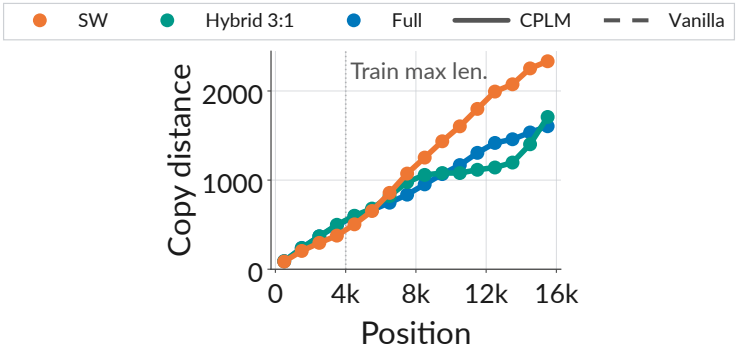


Figure 15: Average distance between the current position and the positions copied by the CP head, as a function of the position, for our 1.7B CPLMs on 5,000 FinePDFs documents of 16k tokens.

Figure 15 reports the average distance between the current position and the positions copied by the CP head, on the same FinePDFs documents as Figure 4. For all backbones, the copy distance grows almost linearly with position, including beyond the 4k-token pretraining context: CPLMs do not

Attention	Model	qa1	qa2	qa3	qa4	qa5	**Avg**
Full	Vanilla	19.0	9.0	17.0	33.0	15.0	18.6
	CPLM	**31.0**	**21.0**	**34.0**	**34.0**	**43.0**	**32.6**
Hybrid 3:1	Vanilla	20.0	**12.0**	23.0	**43.0**	36.0	26.8
	CPLM	**30.0**	6.0	**29.0**	30.0	**42.0**	**27.4**
SW-256	Vanilla	4.0	0.0	5.0	5.0	12.0	5.2
	CPLM	**6.0**	**2.0**	**10.0**	**6.0**	**18.0**	**8.4**

Table 14: BABILong per-task accuracy (%) at 4k context for the 1.7B base models. qa1–qa3: one, two and three supporting facts; qa4: two-argument relations; qa5: three-argument relations. Full and Hybrid use YaRN (×4 from 4096); SW-256 uses plain RoPE. **Bold** marks the better model of each pair.

only copy from the recent context, but make use of the whole document. The SW CPLM copies from the furthest positions, more than 2,000 tokens back at 16k tokens, even though its backbone only attends to the last 256 tokens.

G.3 Other Languages

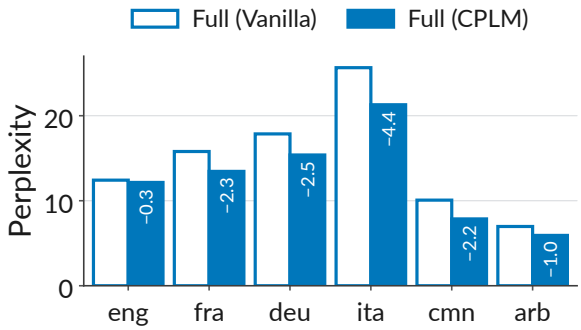


Figure 16: Perplexity of our 135M full-attention vanilla model and CPLM, trained on English FineWeb-Edu, on English data from FineWeb (Penedo et al., 2024) and on French, German, Italian, Mandarin Chinese and Arabic data from FineWeb2 (Penedo et al., 2025). Numbers indicate the perplexity difference between the CPLM and the vanilla model.

Our models are pretrained on English data only. Figure 16 evaluates our 135M full-attention models on other languages, which are largely out of their training distribution. On English FineWeb, the CPLM improves perplexity by 0.3 points, i.e. about 2% in relative terms. On every other language, the gains are much larger, ranging from 14% (Arabic) to 22% (Mandarin Chinese) in relative terms, and reaching 4.4 points for Italian. This is in line with the larger gains observed on out-of-distribution English data (Figure 3): when the model is less able to predict the next token from its parametric knowledge, copying tokens from the context becomes relatively more useful.

H WHERE DOES THE GAIN COME FROM?

In Section 4.1, we argue that CPLMs work mostly because the CP head provides a gradient pathway that bypasses the LM head bottleneck. This section tests this argument mechanically, by removing each ingredient in turn. First, we keep the CP head as an output layer but remove its gradient benefits, by stopping its gradient into the backbone (Appendix H.1). Second, we keep the full CPLM but remove the LM head bottleneck itself, by training byte-level models whose vocabulary is smaller than their hidden dimension (Appendix H.2). If our argument holds, both interventions should remove most of the gains of CPLMs.

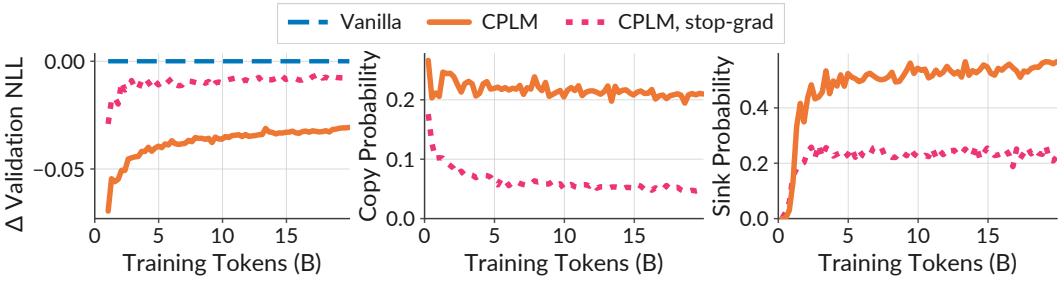
All runs in this section use the 135M full-attention backbone and the training recipe of Table 4 (19,000 steps of 512 sequences of 2,048 units), with a single seed. As in Figure 4, we report the mean copy probability c_t and the mean sink probability s_t over validation positions.

H.1 STOPPING THE COPY GRADIENT INTO THE BACKBONE

Setup. In a CPLM, the next-token loss reaches the backbone through two paths: the LM head, which also produces the copy gate c_t through the <copy> token, and the CP head, through the queries $q_t = W_Q h_t$, the keys $k_\tau = W_K h_\tau$, and the sink score. We only cut the second path, by computing the queries and keys from a detached copy of the hidden states:

$$q_t = W_Q \,\mathrm{sg}(h_t), \qquad k_\tau = W_K \,\mathrm{sg}(h_\tau),$$

where sg is the stop-gradient operator. W_Q, W_K and the sink key are still trained with the full mixture loss of Equation (7), but they must now work with the representations that the backbone learns for the LM head. Everything else is identical to the CPLM. We verified that, with the stop-gradient in place, a loss computed from the CP head distribution alone sends no gradient into the backbone.



(a) Validation NLL gap (b) Copy probability c_t (c) Sink probability s_t

Figure 17: Stopping the gradient of the CP head into the backbone (135M, full attention, 20B tokens). **(a)** Validation NLL relative to the vanilla model at the same step. **(b–c)** Mean copy and sink probabilities. With the gradient stopped, the CPLM keeps about a quarter of its improvement over the vanilla model and copies much less.

	Val. NLL	PPL	ΔNLL vs. Vanilla	c_t	s_t
Vanilla	2.794	16.34	—	—	—
CPLM	2.763	15.85	−0.031	0.210	0.567
CPLM, stop-gradient	2.786	16.22	−0.008	0.046	0.224

Table 15: Final validation NLL (nats per token) and perplexity of the stop-gradient ablation, with the mean copy and sink probabilities.

Results. Stopping the gradient removes most of the benefit (Figure 17 and table 15). The CPLM improves on the vanilla model by 0.031 nats per token (0.49 PPL), against 0.008 nats (0.12 PPL) with the stop-gradient, i.e. about 25% of the full effect. This remaining gain is still several times larger than the seed-to-seed standard deviation of our main arms (0.015–0.037 PPL, Appendix C.1):

the CP head alone, reading representations that were never shaped for it, gives a small but real improvement.

The dynamics of the gap are informative (Figure 17a). Early in training, the stop-gradient model is well ahead of the vanilla model (-0.029 nats at 1B tokens): the CP head can already exploit easy repetitions that a young backbone represents crudely. This advantage shrinks to about -0.010 nats by 5B tokens and to -0.008 nats at the end of training. The gap of the full CPLM starts larger (-0.070), narrows to -0.040 at 5B and -0.035 at 10B tokens, and ends at -0.031 nats. The stop-gradient model also uses the CP head much less (Figures 17b and 17c). Its copy probability falls from 0.18 after 250 steps to 0.046, against a stable 0.21 for the CPLM, so that its effective gate $\lambda_t = c_t(1 - s_t)$ is 0.036, against 0.091 for the CPLM. Its sink probability is also lower (0.22 vs. 0.57), as less copy mass needs to be routed back to the LM head. The model learns that a CP head that it cannot shape is worth little, and mostly turns it off.

Interpretation. Most of the gain of CPLMs comes from what the CP head gradient does to the backbone, rather than from the CP head as an additional output layer. CPLMs have no auxiliary loss: the backbone only receives the gradient of the next-token loss, but through two paths instead of one. The path through the CP head is much less compressed than the one through the LM head (Section 4.1), so that the backbone receives a richer training signal, which in particular helps it represent the context features needed to copy, consistently with CPLM backbones needing fewer induction heads (Section 4.2). With the gradient path closed, W_Q and W_K can only linearly re-read features learned for the LM head, which are a poor fit for position-level matching. This also explains why CPLMs improve the NLL of tokens that they do not copy (Appendix G.1): the backbone they share with the LM head is better trained.

H.2 Byte-Level Language Modeling

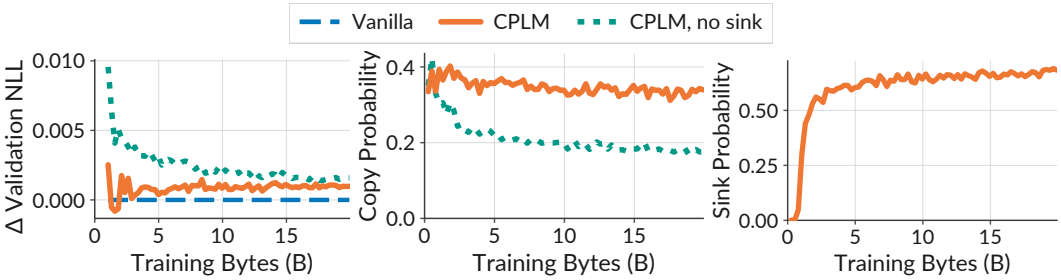
Motivation. If the CP head mainly helped by escaping the LM head bottleneck, its benefits should vanish when there is no bottleneck. Byte-level models provide exactly this setting. Their vocabulary has $V = 257$ entries (256 byte values and a document separator, plus `<copy>` for CPLMs), fewer than the hidden dimension $D = 576$. The LM head matrix $W \in \mathbb{R}^{V \times D}$ then has full row rank: every distribution over the vocabulary can be expressed, and no component of the logit gradient is lost to the nullspace of $W^\top$, unlike for the token-level LM head of Figure 6. This setting is the limit of our SpamLang experiment (Section 4.1), where vanilla models only failed once the vocabulary became large compared to the hidden dimension.

Setup. We train three byte-level models on the same data stream as our token-level models: a vanilla model, a CPLM, and a CPLM without the sink. Each block of SmolLM2 tokens is decoded back to its raw UTF-8 bytes, which is lossless for byte-level BPE, and the first 2,049 bytes are kept as the training sequence, with byte b mapped to id $b + 1$ and the document separator to id 0. The backbone, sequence length (2,048, now in bytes), batch size, optimizer and schedule are unchanged, so that each model sees about 20B bytes, i.e. about 5B tokens worth of text. Only the embedding and LM head shrink, for about 107M parameters in total. Losses are in nats per byte.

	Val. NLL (nats/byte)	Bits/byte	c_t	s_t
Vanilla	0.6908	0.997	—	—
CPLM	0.6918	0.998	0.339	0.683
CPLM, no sink	0.6924	0.999	0.176	—

Table 16: Final validation loss of byte-level models, with the mean copy and sink probabilities.

Results. At the byte level, the CP head brings no gain (Figure 18 and table 16). Both CPLMs end within 0.002 nats per byte of the vanilla model ($+0.0010$ with the sink and $+0.0016$ without), which is comparable to the seed-to-seed standard deviation of our token-level models (Appendix C.1), and the training curves closely track each other throughout training. As for small backbone windows (Appendix C.4), the sink helps early in training ($+0.0025$ vs. $+0.0096$ nats at 1B bytes), consistent with its role as a fallback while the CP head is still unreliable, but neither variant is ever meaningfully ahead of the vanilla model.



(a) Validation NLL gap (b) Copy probability c_t (c) Sink probability s_t

Figure 18: Byte-level language modeling (135M backbone, full attention, 20B bytes). **(a)** Validation NLL (nats per byte) relative to the vanilla model at the same step. **(b–c)** Mean copy and sink probabilities (sink for the CPLM with sink only). All three models reach the same loss within 0.002 nats per byte, even though the CPLM keeps a high copy probability throughout training.

The CP head is not idle, however. The CPLM keeps a copy probability of about 0.34 throughout training, more than at the token level, and relies heavily on the sink ($s_t \approx 0.68$), which leaves an effective gate of 0.11, comparable to the 0.09 of the token-level CPLM. Without the sink, the copy probability settles at 0.18. In both cases, the model copies at least as much as at the token level, yet its loss does not improve: bytes repeat constantly, so copying is easy, but a full-rank LM head predicts them just as well.

Interpretation. The byte-level results support the view that the token-level gains of CPLMs are tied to the LM head bottleneck. Once the LM head is full-rank and gradients reach the backbone without loss, pointing at the context gives the model nothing that it cannot express through the LM head. Together with the stop-gradient ablation, this draws a consistent picture: with a large vocabulary, the next-token loss reaches the backbone through the CP head as an additional training signal, bypassing the low-rank LM head (Section 4.1). This signal accounts for most of the gains of CPLMs, and it has nothing left to contribute once the LM head is no longer a bottleneck. Both interventions thus remove most of the gains of CPLMs, as predicted by our argument in Section 4.1.