Exploring and Improving Drafts in Blockwise Parallel Decoding

Taehyeon Kim [1] **Ananda Theertha Suresh** [2] **Kishore Papineni** [2] **Michael Riley** [2] **Sanjiv Kumar** [2]
Adrian Benton [2]

Abstract

Blockwise parallel decoding (BPD) was proposed in Stern et al. (2018) as a method to improve the inference speed of language models by simultaneously predicting multiple future tokens, termed *block drafts*, which are subsequently verified by the autoregressive model. Block drafts are generated by multiple independent prediction heads of blockwise parallel language models. This paper contributes to the understanding and improvement of block drafts in two ways. First, we analyze the token distributions produced by multiple prediction heads. Secondly, we leverage this analysis to develop algorithms to improve BPD inference speed by refining the block drafts using n-gram and neural language models. Experiments demonstrate that refined block drafts yield a +5-21% increase in block efficiency (i.e., the number of accepted tokens from the block draft) across diverse datasets.

1. Introduction

The landscape of natural language processing has been profoundly reshaped by recent advances in autoregressive language models (Brown et al., 2020; Wei et al., 2022; Radford et al., 2019; Raffel et al., 2020; Team et al., 2023). However, a significant obstacle to their wider application is high inference latency, particularly for extremely deep models with hundreds of billions of parameters (Hoffmann et al., 2022; Rae et al., 2021; Chowdhery et al., 2022). This latency, intrinsic to decoding with autoregressive language models (LMs), imposes considerable computational burdens and limits real-time deployment.

In response to these latency challenges, the field has seen a shift towards decoding methods aimed at reducing the inference latency in large language models (LLM). One promising development is the concept of blockwise parallel decoding (BPD) (Stern et al., 2018; Monea et al., 2023; Cai et al., 2024). Unlike autoregressive decoding, which generates one token at a time, blockwise parallel LMs are outfitted with a set of prediction heads, which propose and verify a draft, a block of subsequent tokens, in parallel. While BPD offers one solution to accelerated text generation, it also poses a challenge in ensuring that the proposed drafts are fluent and natural.

BPD inference speed depends on both the time it takes to produce a block draft and the draft's agreement with the base LM's output (Figure 1a). Unlike standard autoregressive LMs that generate tokens sequentially—ensuring consistency with all preceding tokens (e.g., 'Messi' following 'Lionel')—BPD employs a parallel strategy. Here, blockwise parallel LMs simultaneously predict multiple token drafts (e.g., 'Lionel' and 'Ronaldo'), each independently. The primary challenge in BPD is ensuring that these concurrently generated tokens maintain consistency. Effective block drafters should prioritize sequences such as 'Lionel Messi' over incongruous combinations like 'Lionel Ronaldo', which a reasonable LM is unlikely decode. The focus of this research is improving the quality of block drafts without altering the underlying model parameters.

2. Our contributions

This paper first investigates properties made by the prediction heads of blockwise parallel LMs across several tasks; given these observations, we propose rescoring algorithms to produce higher quality block drafts.

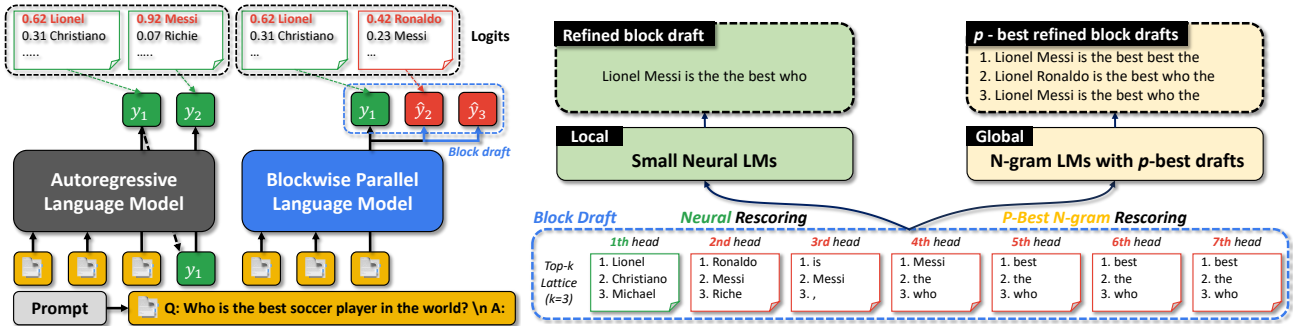
2.1. Observations on block drafts

Consecutive repetitions All heads within a block make predictions independently in a blockwise parallel LM. Unsurprisingly, we observe that this leads to block drafts with significant token repetition across heads. Consecutive repetition is pervasive across tasks, ranging from 20% to 75% of all neighboring draft tokens, depending on the task (**Section 4**).

Oracle top-k block efficiency In the standard BPD algorithm (**Algorithm 1**), the most likely token at each indepen-

[1]KAIST, work done while at Google Research as a student researcher. [2]Google Research, New York. Correspondence to: Adrian Benton <adbenton@google.com>.

Work presented at the ES-FoMo Workshop at ICML 2024. Vienna, Austria.



(a) Example of block drafts

(b) Output of our proposed algorithms

Figure 1. (a) Illustration of the following two tokens decoded by autoregressive greedy decoding vs. two tokens drafted by BPD. (b) Outputs from our proposed algorithms, where the top-k token-level predictions are refined using local neural and global n-gram rescoring, which selects the p most globally probable sequences for batch verification.

dent head is selected as the draft. This approach is prone to two issues: (1) the draft will likely be ungrammatical with artifacts such as word repetition and (2) the model might not be confident about the prediction at some of the heads. We use block efficiency, the average number of draft tokens accepted during decoding, to measure the quality of a given drafter (Leviathan et al., 2023; Sun et al., 2023b). We ask if the block efficiency can be improved by considering the top-k most likely tokens at each head. To measure the potential benefit of considering top-k tokens, we measure the block efficiency of the oracle path through this top-k lattice, *oracle top-k block efficiency*, and show that there is significant headroom for improvement across tasks (**Section 4**).

2.2. New algorithms

Based on these observations, we propose two algorithms to leverage the top-k predictions at each head and improve BPD latency (Figure 1b).

Local rescoring via neural LMs Given the top-k predictions at each head, we refine the block draft by using a small neural, autoregressive LM to greedily rescore these local predictions (**Subsection 5.1**). While the block prediction scores are produced independent of each other, neural rescoring should favor sequences that are fluent, encouraging coherence between the predictions at each head.

Global rescoring via n-gram LMs with multiple drafts If the blockwise parallel LM has h heads and we consider the top-k tokens from each head, then there are k^h candidate drafts of length h that can be formed. We propose to use an n-gram model to efficiently rescore *all* paths, via dynamic programming, and generate the p most probable rescored paths as a batch of draft candidates. These p drafts can then be verified in parallel (**Subsection 5.2**).

There are two critical distinctions between the proposed algorithms. While neural rescoring models are potentially more expressive and can leverage unbounded context, n-gram LMs can be used to efficiently find the *globally* most likely rescored drafts from the entire set of k^h possible draft candidates. Figure 2 shows that our proposed methods enhance block efficiency, increasing it by **+21.30%** relative. This same method also optimizes resource usage, reducing key-value (KV) cache I/O by **-2.54%** and increasing FLOPs per token by **+4.04%** (**Figure 2; Appendix L**).

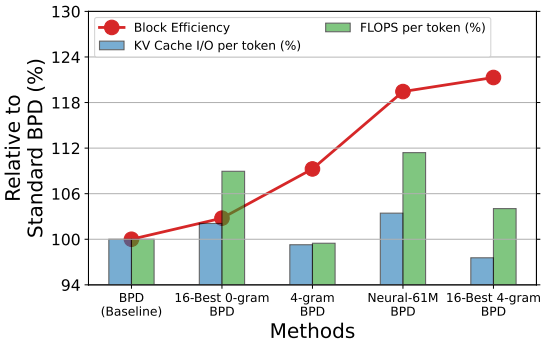


Figure 2. Relative performance of our methods to standard BPD with a 1.5B parameter blockwise parallel LM on NewsRoom dataset (Grusky et al., 2018). Details are described in **Appendix L**.

3. Preliminaries

This section introduces notation and concepts, including algorithms for standard autoregressive decoding and BPD.

Autoregressive decoding Let $\mathcal{M}_\theta$ be an autoregressive LM parameterized by θ. The objective is to generate an output sequence $y_{\leq T} = (y_1, \ldots, y_T)$ conditioned on an input sequence $\bar{x}$. $z_t = \mathcal{M}_\theta(y_t|\bar{x}, y_{<t})$ is a vector of logits, $z_t \in \mathbb{R}^{|\mathcal{V}|}$, where $\mathcal{V}$ is the vocabulary over tokens. These logits define a conditional probability distribution at each time step $p_\theta(y_{t+1}|\bar{x}, y_{\leq t}) = \frac{e^{z_t}}{\sum e^{z_t}}$, which by the chain rule yields $p_\theta(y_{\leq T}|\bar{x}) = \prod_{t=1}^{T} p_\theta(y_t|\bar{x}, y_{<t})$.

Blockwise parallel decoding Let $\mathcal{M}_\theta^h$ be a blockwise parallel LM with block size h. This model employs h distinct feedforward neural (FFN) layer with a single hidden layer,

Algorithm 1 Blockwise parallel decoding (BPD)

input : Blockwise parallel LM $\mathcal{M}_\theta^h$, initial prompt sequence $\bar{x}$ and target sequence length T.

Initialize $t \leftarrow 1$
while $t < T$ **do**
  /* Stage 1: Predict */
  $z_{t,1}^1, \ldots, z_{t,h}^h \leftarrow \mathcal{M}_\theta^h(y_{t+1}, \ldots, y_{t+h} | \bar{x}, y_{\leq t})$
  $y_{t+1}, \hat{y}_{t+2}, ..., \hat{y}_{t+h} \leftarrow \arg\max(z_{t,1}^1, z_{t,2}^2, ..., z_{t,h}^h)$
  /* Stage 2: Verify */
  for $j \leftarrow 2, \ldots, h$ in parallel **do**
    $z_{t,j}^1, \cdots \leftarrow \mathcal{M}_\theta^h(y_{t+j}, \ldots | \bar{x}, y_{\leq t+1}, \hat{y}_{t+2}, \cdots, \hat{y}_{t+j-1})$
  end for
  /* Stage 3: Accept */
  $n \leftarrow \max\{n : \hat{y}_{t+j} = \arg\max z_{t,j}^1, 2 \leq j \leq n\}$
  $t \leftarrow t + n$
end while

atop the target LM's final hidden layer. The output of each FFN is followed by a softmax layer over the vocabulary to predict each of the h subsequent tokens in the block. In our experiments, the parameters of the FFNs are learned jointly with the base LM during training, and the weights of all softmax layers are tied to the input embedding table. **Algorithm 1** describes the BPD greedy decoding procedure:

1. **Predict** $\mathcal{M}_\theta^h$ is used to generate a draft of h token predictions $y_{t+1}, \hat{y}_{t+2} \cdots, \hat{y}_{t+h}$, conditioned on the prompt, $\bar{x}$, and existing generated text, $y_{\leq t}$. y_{t+1} is identical to the target LM greedy decode.

2. **Verify** At this stage, the target LM greedily generates next-token logits $\{z_{t,2}^1, \cdots, z_{t,h}^1\}$ conditioned on the existing prefix and block draft $\{\bar{x}, y_{\leq t+1}, \hat{y}_{t+2}, \cdots, \hat{y}_{t+h}\}$. Verification amounts to checking which block draft tokens match the autoregressive greedy decode from the target LM. Verification of all positions can be performed in parallel if the target LM is a decoder-only transformer.

3. **Accept** Finally, the length of the longest contiguous prefix of draft tokens that match the target LM greedy decode is identified: n. The decoded sequence is extended by $n + 1$ tokens and we iterate.[1] Note that in general, not all h tokens are accepted, and many of the draft tokens in each block are discarded. Since the additional time required to generate a block of tokens is fast relative to the time it takes for the forward pass of the target LM, a modest gain in accepted prefix length justifies the cost of draft generation.

[1]The decoded sequence is extended by $n + 1$ tokens since during verification we generate the token from the target LM, $\arg\max z_{t,n+1}{}^1$, at the first position where the draft differs from the target LM greedy decode.

Table 1. Consecutive token repetition in block drafts before and after C4-trained 2-gram rescoring of the top-16 lattice. "% Consec" is the percentage of consecutive identical draft tokens out of all pairs of consecutive tokens. "Max run" is the average maximum repeated subsequence length in tokens.

Task	Dataset	% Consec		Max run	
		Vanilla	2-gram	Vanilla	2-gram
LM	LAMBADA	20.0	**10.7**	2.2	**1.8**
QA	SQuAD V1	75.5	**67.6**	6.6	**6.1**
S-SUM	CNN/Daily	46.4	**21.9**	3.8	**2.5**
	SAMSUM	29.9	**20.0**	3.1	**2.5**
L-SUM	MultiNews	33.6	**14.7**	3.1	**2.1**
	XSUM	24.0	**9.4**	2.6	**1.7**
	NewsRoom	47.2	**32.1**	4.1	**3.3**

4. Exploration of BPD drafts

Consecutive repetitions We observe that vanilla block drafts are prone to significant token repetition. This is due to the fact that each head's prediction is independent of the others, and is a limitation shared with non-autoregressive generation in general (Gu et al., 2017). Table 1 shows the proportion of consecutive tokens in block drafts that are identical to each other, along with the average length of the longest run per block draft.

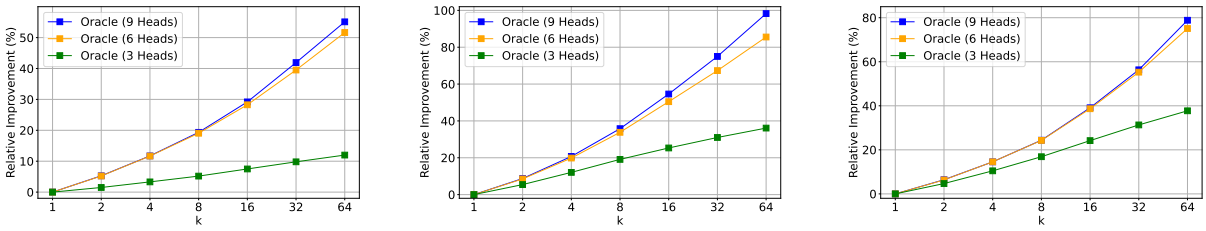
We compare these statistics before and after rescoring with a 2-gram LM: a trivial rescorer, but one that can encourage local consistency between consecutive draft tokens. Runs of repeated tokens are unnatural, and unlikely to be generated by a strong base language model. Rescoring the top-k block draft lattice with a 2-gram LM reduces the percentage of consecutive repeated tokens from between **9.9%** to **24.5%**, depending on the task.

Oracle efficiency The concept of oracle block efficiency in BPD serves as a theoretical benchmark, illustrating the headroom available from improving the quality of the block draft. To compute oracle block efficiency, we consider the top-k most probable tokens at each head, and form a "sausage" lattice from these.

Generating drafts via oracle is not practical, but rather a reference point. Analyzing the gap between actual BPD performance and the oracle upper bound (Figure 3) helps us to understand the limitations of the original block drafts and potential areas for improvement. Additionally, exploring oracle efficiency as a function of k in the top-k lattice, demonstrates how "close" the block draft was to producing a stronger draft.

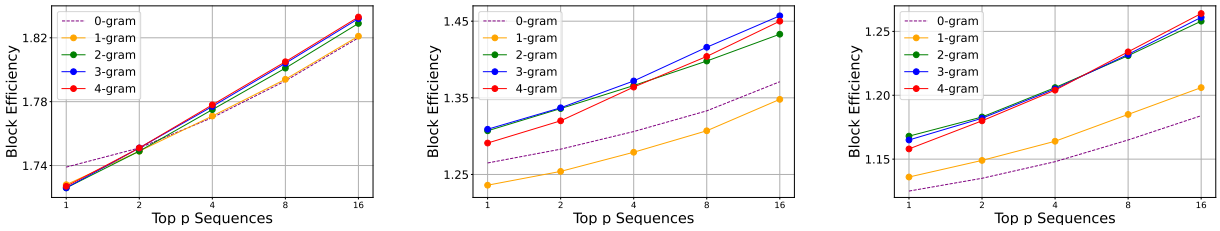
5. Lattice rescoring for improved block efficiency

Each of these algorithms is a modification of the block drafted in **Stage 1** in Algorithm 1. Instead of using the most



(a) CNN/Daily (b) SAMSUM (c) XSUM

Figure 3. Oracle block efficiency over the top-k lattice as a function k. Each plot (a-c) represents a different task, demonstrating the relative improvement in block efficiency of the oracle draft with respect to the vanilla block draft.



(a) CNN/Daily (b) SAMSUM (c) XSUM

Figure 4. Block efficiency of p-best n-gram BPD methods as a function of the number of top p sequences verified in parallel. The block efficiency of the methods is evaluated with the the same number of paths extracted from the top-16 lattice.

Table 2. Block efficiency of our methods over the top-16 lattice. '16-best 0-gram BPD' indicates performance of 16-best verification over the original lattice without n-gram rescoring. Relative percent improvement over BPD (Baseline) is indicated in parentheses. Green circles (●) indicate improvement over the Baseline, while red circles (●) denote no improvement.

Task	Dataset	Baseline BPD	Local rescoring neural-61M BPD	Global rescoring 4-gram BPD	Global rescoring 16-best 0-gram BPD	Global rescoring 16-best 4-gram BPD	Oracle (k=16)
LM	LAMBADA	3.12	3.08 (-1.28%) ●	3.05 (-2.24%) ●	3.23 (+3.53%) ●	**3.29 (+5.45%)** ●	3.67
QA	SQuAD V1	2.08	2.10 (+0.96%) ●	2.07 (-0.48%) ●	2.18 (+4.85%) ●	**2.22 (+6.87%)** ●	2.45
S-SUM	CNN/Daily	1.74	1.73 (-0.57%) ●	1.73 (-0.57%) ●	1.82 (+4.66%) ●	**1.83 (+5.41%)** ●	2.26
	SAMSUM	1.27	1.39 (+9.45%) ●	1.29 (+1.57%) ●	1.37 (+7.87%) ●	**1.45 (+14.17%)** ●	1.95
L-SUM	MultiNews	1.10	**1.25 (+13.64%)** ●	1.12 (+1.82%) ●	1.13 (+2.73%) ●	1.22 (+10.91%) ●	1.43
	XSUM	1.13	1.23 (+8.85%) ●	1.16 (+2.65%) ●	1.18 (+4.42%) ●	**1.26 (+11.50%)** ●	1.55
	NewsRoom	1.08	1.29 (+19.44%) ●	1.18 (+9.26%) ●	1.11 (+2.78%) ●	**1.31 (+21.30%)** ●	1.50

likely token at each head as the prediction, we construct the top-k sausage lattice of likely drafts from each head, where the set of top-k tokens is denoted as S_i for head i. This approach allows any token within S_i to be chosen for position i, yielding a total possible combinations of k^h.

In this lattice, any path from the start to final state represents a viable draft. we propose two algorithms to select a small number of h-length drafts from this lattice, which are then verified. The first algorithm uses neural autoregressive transformers (**Subsection 5.1**) for rescoring, while the second uses n-gram language models (**Subsection 5.2**).

5.1. Local rescoring via neural models

A simple approach uses a small neural rescorer, interpolating between the logits of the rescorer LM and vanilla block logits with an interpolation weight. The rescored prediction is given by: $z^j_{t,j}[S_j] \leftarrow z^j_{t,j}[S_j] + \alpha \cdot r_{t+j}[S_j]$

$z^j_{t,j}$ represents the logit of the block draft at head j, and r_{t+j} is the corresponding logit predicted by the small neural rescoring model, which is conditioned on the sequence $y_{\leq t}, \cdots, \hat{y}_{t+2}, \cdots, \hat{y}_{t+j-1}$. The parameter α is the weight placed on the rescorer's prediction. We experiment with decoder-only transformers with 32-94 million parameters (**Subsection D.2**). We use greedy rescoring in our experiments.

5.2. Global *n*-gram rescoring

We also evaluate the quality of drafts generated by rescoring with an n-gram LM. Recall that blockwise parallel LMs can be used to compute a lattice representing k^h possible

sequences. We rescore all of these sequences with an n-gram model, select the top p sequences and pass them to the verification stage. When $p = 1$, we refer to this as *n-gram BPD* and when $p > 1$, *p-best n-gram BPD*.

While global rescoring typically yields better results compared to local rescoring, explicitly rescoring k^h sequences with a neural LM and selecting the most likely sequence would take time $O(k^h)$, which is computationally prohibitive. Hence, we take advantage of n-gram models, which are unique in that we can efficiently select the most likely sequence in time poly(k, h) using dynamic programming. We use the OpenFST library (Allauzen et al., 2007) to represent each n-gram model as a weighted finite state automaton and apply finite state composition with the top-k lattice followed by extraction of the p most likely draft sequences. Training details for the n-gram models are given in **Appendix D.3**.

5.3. Empirical evaluation

Block efficiency Table 2 and Figure 4 demonstrate the impact of lattice rescoring on block efficiency across various language modeling, extractive question answering, and summarization tasks. Autoregressive neural, n-gram LM, and p-best n-gram BPD rescoring all demonstrate improvements in block efficiency, although gains are task-dependent.

For tasks with high initial BPD block efficiency (LAMBADA, CNN/Daily), both of the rescoring methods show little to no improvement, suggesting that vanilla BPD already produces high quality drafts. However, when block efficiency is initially low, we find that lattice rescoring leads to block efficiency gains, with greedy neural rescoring achieving the best performance in some cases. This suggests that rescoring most helps refine predictions when the initial drafts are particularly poor.

6. Conclusion

This paper presents a comprehensive analysis of BPD, highlighting its predictive dynamics and proposing methods to refine the generation of block drafts. Our study offers insights into BPD's behavior, particularly the tendency for drafts to contain consecutive repetitions and its heads to exhibit varying confidence levels in predictions. We introduce a novel measure, oracle top-k block efficiency, to explore potential improvements in block efficiency. Two algorithms are proposed: local rescoring with small neural models, and global rescoring of multiple drafts with n-gram LMs. These algorithms leverage the strengths of both blockwise parallel LMs and small rescoring models to streamline the decoding process, pushing the boundaries of efficient text generation with BPD.

References

Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., and Mohri, M. Openfst: A general and efficient weighted finite-state transducer library: (extended abstract of an invited talk). In *Implementation and Application of Automata: 12th International Conference, CIAA 2007, Praque, Czech Republic, July 16-18, 2007, Revised Selected Papers 12*, pp. 11–23. Springer, 2007.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Cai, T., Li, Y., Geng, Z., Peng, H., Lee, J. D., Chen, D., and Dao, T. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*, 2024.

Chen, C., Borgeaud, S., Irving, G., Lespiau, J.-B., Sifre, L., and Jumper, J. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

Chi, E. A., Salazar, J., and Kirchhoff, K. Align-refine: Non-autoregressive speech recognition via iterative realignment. *arXiv preprint arXiv:2010.14233*, 2020.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35:30318–30332, 2022.

Elbayad, M., Gu, J., Grave, E., and Auli, M. Depth-adaptive transformer. *arXiv preprint arXiv:1910.10073*, 2019.

Fabbri, A. R., Li, I., She, T., Li, S., and Radev, D. R. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. *arXiv preprint arXiv:1906.01749*, 2019.

Gliwa, B., Mochol, I., Biesek, M., and Wawer, A. Samsum corpus: A human-annotated dialogue dataset for abstractive summarization. *arXiv preprint arXiv:1911.12237*, 2019.

Grusky, M., Naaman, M., and Artzi, Y. Newsroom: A dataset of 1.3 million summaries with diverse extractive strategies. In Walker, M., Ji, H., and Stent, A. (eds.), *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 708–719, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1065. URL https://aclanthology.org/N18-1065.

Gu, J., Bradbury, J., Xiong, C., Li, V. O., and Socher, R. Non-autoregressive neural machine translation. *arXiv preprint arXiv:1711.02281*, 2017.

Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. Teaching machines to read and comprehend. *Advances in neural information processing systems*, 28, 2015.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., et al. An empirical analysis of compute-optimal large language model training. *Advances in Neural Information Processing Systems*, 35: 30016–30030, 2022.

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pp. 1–12, 2017.

Katz, S. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE transactions on acoustics, speech, and signal processing*, 35(3):400–401, 1987.

Kim, S., Mangalam, K., Moon, S., Malik, J., Mahoney, M. W., Gholami, A., and Keutzer, K. Speculative decoding with big little decoder. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Kim, T., Kim, J., Lee, G., and Yun, S.-Y. Instructive decoding: Instruction-tuned large language models are self-refiner from noisy instructions. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=LebzzClHYw.

Kitaev, N., Kaiser, Ł., and Levskaya, A. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

Leviathan, Y., Kalman, M., and Matias, Y. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

Li, X. L., Holtzman, A., Fried, D., Liang, P., Eisner, J., Hashimoto, T., Zettlemoyer, L., and Lewis, M. Contrastive decoding: Open-ended text generation as optimization. *arXiv preprint arXiv:2210.15097*, 2022.

Ma, X., Fang, G., and Wang, X. Llm-pruner: On the structural pruning of large language models. *arXiv preprint arXiv:2305.11627*, 2023.

Monea, G., Joulin, A., and Grave, E. Pass: Parallel speculative sampling. *arXiv preprint arXiv:2311.13581*, 2023.

Narayan, S., Cohen, S. B., and Lapata, M. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *arXiv preprint arXiv:1808.08745*, 2018.

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, Q. N., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., and Fernández, R. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Rae, J. W., Borgeaud, S., Cai, T., Millican, K., Hoffmann, J., Song, F., Aslanides, J., Henderson, S., Ring, R., Young, S., et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv e-prints*, 2019.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

Schuster, T., Fisch, A., Gupta, J., Dehghani, M., Bahri, D., Tran, V., Tay, Y., and Metzler, D. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472, 2022.

Spector, B. and Re, C. Accelerating llm inference with staged speculative decoding. *arXiv preprint arXiv:2308.04623*, 2023.

Stern, M., Shazeer, N., and Uszkoreit, J. Blockwise parallel decoding for deep autoregressive models. *Advances in Neural Information Processing Systems*, 31, 2018.

Stolcke, A. Entropy-based pruning of backoff language models. *arXiv preprint cs/0006025*, 2000.

Sun, M., Liu, Z., Bair, A., and Kolter, J. Z. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023a.

Sun, Z., Suresh, A. T., Ro, J. H., Beirami, A., Jain, H., and Yu, F. Spectr: Fast speculative decoding via optimal transport. *arXiv preprint arXiv:2310.15141*, 2023b.

Team, G., Anil, R., Borgeaud, S., Wu, Y., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pp. 38087–38099. PMLR, 2023.

Yao, Z., Yazdani Aminabadi, R., Zhang, M., Wu, X., Li, C., and He, Y. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems*, 35: 27168–27183, 2022.

Zhu, C., Ping, W., Xiao, C., Shoeybi, M., Goldstein, T., Anandkumar, A., and Catanzaro, B. Long-short transformer: Efficient transformers for language and vision. *Advances in neural information processing systems*, 34: 17723–17736, 2021.

Appendices

A. Broader impact

Our research in advancing blockwise parallel decoding (BPD) for language models (LMs) paves the way for substantial improvements in language processing. This section outlines the broader implications of our work, focusing on key areas such as efficiency, scalability, and systematic impact.

- **Inference efficiency:** We've achieved further acceleration in inference with minimal increases in CPU overhead and memory bandwidth utilization. In particular, we show that our methods significantly boost the inference efficiency compared to the standard autoregressive inference.
- **Scalability:** The introduction for selecting top p sequences from top k predictions demonstrates notable scalability, hinting at accelerated inference with growing model sizes. This scalability is further explored through diverse experiments involving various n-gram and neural LM architectures.
- **Systematic impact:** The advancement of BPD promises identical performance to traditional autoregressive methods, but with significantly improved efficiency in decoding times, which can be helpful in latency-critical applications.

B. Limitation & future work

B.1. Limitation

Extending beyond greedy decoding In this work, we focus on refining block drafts for greedy decoding, the decoding scheme considered in the original BPD paper (Stern et al., 2018). Extending this methodology to non-greedy sampling strategies is an area for future work. Leviathan et al. (Leviathan et al., 2023) illuminates the feasibility of incorporating speculative decoding strategies to expedite non-greedy sampling processes. Their approach underscores the universality of the verification process across diverse model architectures, contingent on the availability of draft token-level predictions and probabilities — a requirement met by (rescored) block candidates.

Scaling up to larger models We show that lattice rescoring can improve the block efficiency for a 1.5 billion parameter blockwise parallel LM. It remains an open question as to whether one would observe similar improvements in block efficiency with larger LMs.

Memory overhead during parallel verification A notable challenge in our framework is the memory overhead encountered during parallel verification. The simultaneous processing of multiple token predictions, particularly for larger models and draft batches when coupled with lattice rescoring, can intensify memory demands. That said, this can be mitigated by the application of tree attention during verification (Spector & Re, 2023).

Naive drafting heads and training recipes The implementation of drafting heads within our BPD framework, while foundational, leaves room for advanced development. This aspect is particularly crucial for refining the model's accuracy and efficiency. Future iterations could benefit from more sophisticated training methods, enhancing these heads' ability to navigate complex linguistic contexts and improving overall predictive performance.

B.2. Future work

The evolution of the BPD framework will be pivotal in addressing its current constraints and broadening its utility. Key areas for future research include:

- The intrinsic compatibility of our lattice rescoring method with alternative sampling strategies presents a fertile ground for future inquiries. This delineation not only enriches the discourse within efficient language model inference but also sets a trajectory for subsequent empirical endeavors.
- Scaling the blockwise parallel LM for compatibility with larger-scale LLMs, ensuring it remains effective and efficient as language models evolve.

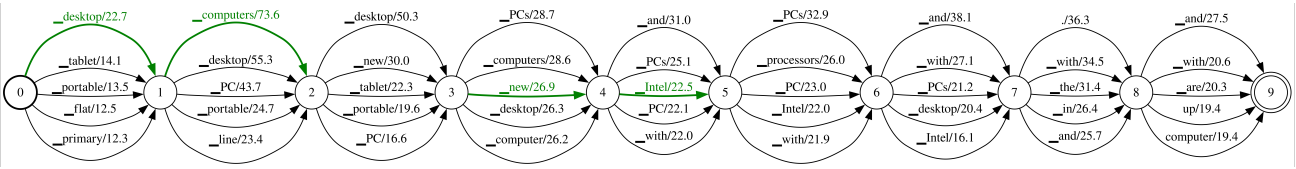


Figure 5. An example of a top-5 sausage lattice generated on a NewsRoom example. Edge weights correspond to (rescored) logits. Edges at each time step are ordered in descending weight and green, bolded edges correspond to candidates matching the greedy decode over the next nine tokens: *"... desktop computers with new Intel Corp processors that it ..."*. The initial node in this graph is state 0 and the final node is 9.

- Advancing training methodologies for drafting heads, to bolster their predictive accuracy and contribution to the BPD process, thereby optimizing the framework for more complex language modeling tasks.

- Using the sequential entropy ordering of heads (Figure 6b) as a possible halting condition during block draft head training, or to inform how a rescoring LM should be interpolated with the block lattice weights.

C. Related work

C.1. Efficient transformer inference

Works on improving transformer efficiency encompass both optimization of an existing set of model weights, or a fundamental change to the model architecture. Examples of the former include techniques such as quantization (Xiao et al., 2023; Yao et al., 2022; Dettmers et al., 2022) and model pruning (Sun et al., 2023a; Ma et al., 2023). In parallel, neural architecture search has played a crucial role in identifying network structures that balance performance with efficiency (Kitaev et al., 2020; Zhu et al., 2021). Relatedly, Elbayad et al. (Elbayad et al., 2019) propose early-exiting at intermediate layers for faster inference, while Schuster et al. (Schuster et al., 2022) explore confidence thresholding for balancing speed and accuracy. These methods offer insights into optimizing decoding under resource constraints.

One important line of work has focused on modifying the decoding method in LMs. The adoption of non-autoregressive (parallel) decoding strategies (Stern et al., 2018; Gu et al., 2017) marks a pivotal shift in this domain, addressing inference latency by simultaneously generating multiple tokens. Subsequent innovations have sought to refine this approach by incorporating additional context (Chi et al., 2020), iterative refinement (Kim et al., 2023), and tree-based attention mechanism (Cai et al., 2024). However, these refinements often require complex training or additional inference data.

C.2. Efficient and effective decoding

There are several recent works that improve the speed of LLM decoding, including pioneering works like BPD and speculative decoding. Speculative decoding leverages a smaller 'draft' model to anticipate the outputs of a larger target model, improving average decode latency without loss in generation quality (Leviathan et al., 2023; Chen et al., 2023; Kim et al., 2023). The draft model is typically trained on the same corpus as the LLM, thus autoregressively generates similar drafts as the target model with reduced latency. Speculative decoding is most successful when a long sequence of speculated tokens are accepted by the target LM during verification, avoiding multiple serial calls to the target LM to generate the same sequence.

On the surface, contrastive decoding algorithms share some similarities with our proposed draft rescoring approach, insofar as a weaker model is used to modify the predictions of the target LM (Li et al., 2022; Kim et al., 2024). However, in this work, we refine block drafts solely to improve latency. Like speculative decoding, our proposals have no effect on the quality of the target LM's generated text.

D. Experimental setup

In this paper, we use ≈ 1.5 billion (B) parameter decoder-only transformer LMs with up to 9 blockwise heads.[2] The 1.5B model and all other LMs were pretrained using the C4 English mixture (Raffel et al., 2019) with the causal next token

[2]This study is based on the original BPD framework, with a modification: we use decoder-only models instead of the T5 encoder-decoder architecture. Other setups are consistent with the approach in (Stern et al., 2018).

Table 3. Per-task test performance of each finetuned model and block efficiency over language modeling (LM), extractive question answering (QA), and both long and short summarization (L-Sum & S-Sum).

Task	Dataset	Performance	Block Efficiency
LM	LAMBADA (Paperno et al., 2016)	7.88	3.12
QA	SQuAD V1 (Rajpurkar et al., 2016)	57.60	2.08
S-SUM	CNN/Daily (Hermann et al., 2015)	39.85	1.74
	SAMSUM (Gliwa et al., 2019)	37.66	1.27
L-SUM	MultiNews (Fabbri et al., 2019)	23.08	1.10
	XSUM (Narayan et al., 2018)	52.15	1.13
	NewsRoom (Grusky et al., 2018)	39.85	1.08

Table 4. Outputs from BPD frameworks. Black indicates standard decoded output, blue indicates accepted draft tokens, and brown is the prompt.

LAMBADA

it's nothing more than a faceless, formless brown blob to me, but I take his word for the resemblance to our genetic makeup. ... {*Skip*}...

SQuAD V1

Question: Who was announced as the LEM contractor in November 1962? context: Wiesner kept up the pressure, even making the disagreement public ... {*Skip*}...
Answer: Grumman

XSUM

Summarize: ... {*Skip*}...
Millions of small businesses will benefit from a reduction of business rate from the Budget Osborne, Chancellor George Osborne has announced.

prediction objective tokenized with the GPT3 subword vocabulary (Brown et al., 2020). For the 1.5B blockwise parallel LMs, all heads were trained jointly to predict the following h tokens at each iteration. During pretraining, we use batches of 2048 subword sequences, each 512 tokens in length, amounting to $\approx$ 200B input tokens in total on TPUv3/TPUv4 (Jouppi et al., 2017) with Jax (Bradbury et al., 2018).

We evaluate the potential latency improvement of block drafts by *block efficiency* (Leviathan et al., 2023; Sun et al., 2023b). In this context, block efficiency represents the theoretical speedup compared to standard greedy decoding. It is defined as the average number of tokens decoded per serial call to the blockwise parallel LMs. The formula for block efficiency is given by $B := \frac{\text{Total number of decoded tokens}}{\text{Total number of serial calls to } \mathcal{M}_\theta^h}$.

In this definition, the total number of decoded tokens is the sum of the number of accepted tokens across decoding steps, not necessarily all h predicted tokens in each block. Only the tokens that pass the 'Verify' stage and align with the base model's predictions are accepted and integrated into the final sequence. This ensures that generated text is identical to the target LM, while achieving speedup. The total number of serial calls to $\mathcal{M}_\theta^h$ is the number of times the model processes a block of tokens. A block efficiency of 1 means that one is achieving no speedup relative to standard decoding.

In addition to a standard language modeling dataset, LAMBADA (Paperno et al., 2016), we conduct experiments across several classes of downstream tasks. In the realm of text summarization, we evaluate models on the XSUM (Narayan et al., 2018), MultiNews (Fabbri et al., 2019), SAMSum (Gliwa et al., 2019), NewsRoom (Grusky et al., 2018) and CNN/DailyMail (Hermann et al., 2015) datasets. Each of these datasets is characterized by distinct summary lengths and styles. For extractive QA, the SQuAD V1 dataset (Rajpurkar et al., 2016) serves as our testbed. For each task aside from language modeling, we finetune the blockwise parallel LM for that task.[3] Table 3[4] shows that block efficiency varies dramatically across tasks and as a function of the number of block draft heads. We use all 9 block draft heads for subsequent experiments as this acts as an upper bound on possible block efficiency.

Table 4 sketches how BPD acts on three examples from each class of tasks.

- **LM:** BPD excels at generating common multi-word expressions in a single step. For example, (no) 'thing more than', and (take) 'his word for the' are each drafted and accepted in a single step.
- **QA:** BPD also attains high block efficiency in extractive QA, where it correctly drafts multi-token entities copied from the input sequence. In SQuAD V1, it accurately completes the answer 'Grumman' from 'Gru' by adding 'mman', highlighting its ability to process multiple tokens at once and quickly extend answers.
- **SUM:** BPD's effectiveness in SUM tasks varies by dataset. For formulaic summaries like CNN/DailyMail, it performs well, reflecting its alignment with LM and QA tasks. However, in narrative-driven datasets like SAMSum and XSUM, where concise summaries are required, the block efficiency of BPD is little better than standard decoding.

As an example of rescoring methods, Figure 5 gives an example of a top-5 BPD sausage lattice. Observe that any path from the state 0 to state 9 is a path possible by the algorithm.

[3]Details are given in **Subsection D.2**.

[4]The performance metric for LM is perplexity, for QA is exact match, and for the remaining summarization tasks, the metric is ROUGE-L.

D.1. Training objective for blockwise parallel LMs

We minimized the following loss function to train blockwise parallel LMs:

$$\mathcal{L}_{BPD} = \sum_{h=1}^{H} \lambda_h \mathcal{L}_h$$

where H is the number of heads, λ_h is a non-negative scalar that weights the loss from head h, and $\mathcal{L}_h$ denotes the loss for each individual head:

$$\mathcal{L}_h = - \sum_{x_{1\ldots i}, y_{i+h}} \log p(y_{i+h}|x_{1\ldots i})$$

where $x_{1\ldots i}$ is the token sequence up to position i, y_{i+h} is the ground truth token at position $i+h$, and $p(y_{i+h}|x_{1\ldots i})$ is the probability of observing token y_{i+h} given the sequence $x_{1\ldots i}$ under the blockwise parallel LM. We trained all models in this work with $\lambda_h = 1$. We leave tuning these hyperparameters, improving the block efficiency and quality of the blockwise parallel LM, as future work.

D.2. Neural Model Details

Table 5. Architecture hyperparameters for each of the transformer-based neural language models.

Type	Model	# Layers	Embedding Dim	Hidden Dim
Blockwise Parallel Decoder	1.5B	18	1,536	12,288
Autoregressive Decoder	32M	2	384	1,536
	61M	12	384	1,536
	94M	6	768	3,072

Each neural rescoring LM is a decoder-only transformer with learned absolute positional embeddings and twelve self-attention heads at each layer. The key architecture hyperparameters are given in Table 5. Aside from scale, the only difference between the blockwise parallel LM and neural rescoring models is the addition of the feedforward neural networks and eight additional block prediction heads. Note that the number of parameters for each of these models also includes the embedding table.

Each model was pretrained on the English C4 corpus for 200K iterations with a batch size of $2^{20} \approx 1M$ tokens per batch. Dropout was not applied. For the blockwise parallel LM, all heads were trained jointly. The pretraining for the blockwise parallel LMs took about 47 hours on 128 TPUv3 units.

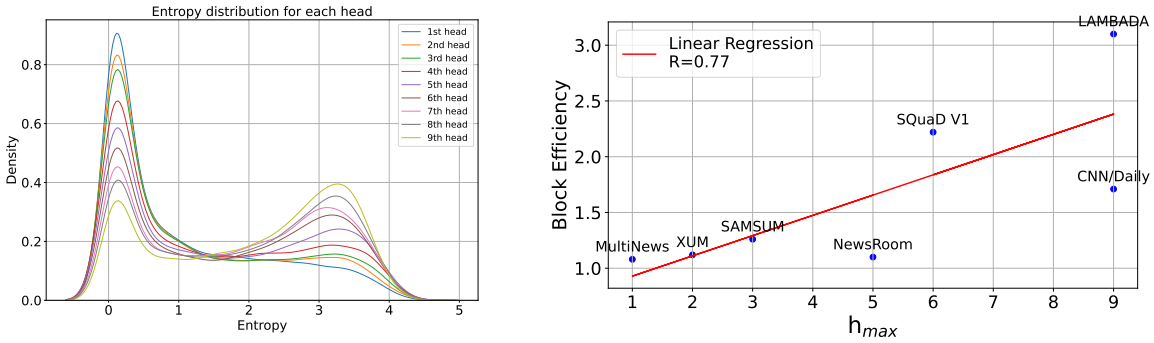
For downstream tasks, models were finetuned for a maximum 100K iterations with a batch size of two examples with maximum sequence length of 2048. Maximum learning rate was fixed to 10^{-4} for all runs, with a cosine learning rate schedule. Checkpoints were selected based on heldout set model performance. Interpolation weight for all rescoring models was tuned for block efficiency on 100 randomly selected examples from the evaluation set for each task, and performance was reported on the remainder of the evaluation set. Figure 2 describes the decoding process of local neural rescoring.

D.3. *n*-gram Details

All *n*-gram LMs in this work are Katz backoff *n*-gram LMs (Katz, 1987) fit on the train split of the GPT3 subword-tokenized English C4 corpus with *n*-gram order $\in \{2, 4\}$. We apply entropy pruning (Stolcke, 2000) to reduce model size to a maximum of 100 million *n*-grams per model, and ensure that each trigram is observed at least twice and each 4-gram is observed at least four times. Preprocessing of the text is identical to that used to train neural LMs.

D.4. Datasets

- **LAMBADA (LAnguage Modeling Broadened to Account for Discourse Aspects)**: A collection of narrative passages designed to test the understanding of long-range dependencies in language models, where the task involves



(a) Entropy distributions across block draft heads

(b) Correlation between block efficiency and $h_{\max}$

Figure 6. (a) Entropy distributions across block draft heads on LAMBADA. The density plots illustrate the entropy distribution for each head in the model. (b) Correlation between block efficiency and $h_{\max}$, the head until which the average entropy in a task increases monotonically.

predicting the last word of a passage based on the full context (Paperno et al., 2016).

- **SQuAD V1 (Stanford Question Answering Dataset)**: A reading comprehension dataset that features questions based on Wikipedia articles, with answers located within the text (Rajpurkar et al., 2016).
- **CNN/DailyMail**: This dataset includes news articles paired with human-written summaries, mainly used to evaluate the summarization capabilities of language models, particularly in abstractive summarization (Hermann et al., 2015).
- **SAMSum (Semi-Automatic Machine Summarization)**: Focuses on abstractive summarization using news articles and machine-generated summaries, testing models' abilities to refine and improve existing summaries (Gliwa et al., 2019).
- **MultiNews**: Comprises news articles from diverse sources for abstractive summarization tasks, evaluating models on handling different writing styles and topics (Fabbri et al., 2019).
- **XSUM**: Contains scientific documents and summaries, challenging language models to process complex scientific information and language (Narayan et al., 2018).
- **NewsRoom**: A dataset of news articles aimed at assessing the factual accuracy and information extraction capabilities of models in generating summaries (Grusky et al., 2018).

All datasets were tokenized using the 50,257 GPT3 subword vocabulary (Brown et al., 2020).

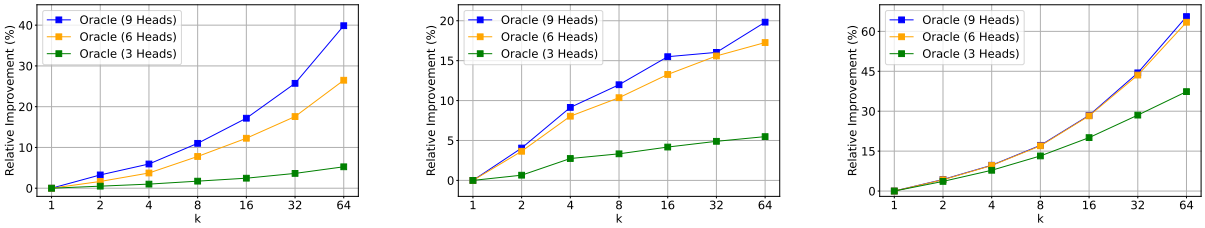
Templates We used the following prompts during model finetuning and inference.

- **SQuAD**: "question: [question] context: [context]"
- **CNN/DailyMail**: "summarize: [text]"
- **SAMSum**: "Here is a dialogue: [text]\nWrite a short summary!"
- **MultiNews**: "Write a summary based on this article: [text]"
- **XSUM**: "Summarize: [text]"
- **NewsRoom**: "Please write a short summary for the following article: [title] [text]"

Algorithm 2 Local rescoring via neural models

input : Blockwise parallel LM $\mathcal{M}_\theta^h$, top-k indices selection function TOP-$k(\cdot)$, rescoring model $\mathcal{M}_{\theta_r}$, interpolation weight $\alpha > 0$.

$z_{t+1}^1, \ldots, z_{t+h}^h \leftarrow \mathcal{M}_\theta^h(y_{t+1}, \ldots, y_{t+h} | \bar{x}, y_{\leq t})$
$S_1, \ldots, S_h \leftarrow$ TOP-$k(z_{t+1}^1, \ldots, z_{t+h}^h)$
/* Lattice Rescoring (Subsection 5.1)*/
for $j \leftarrow 2, \ldots, h$ in parallel **do**
  $r_{t+j} \leftarrow \mathcal{M}_{\theta_r}(y_{t+j} | x, y_{<t+j})$
  $z_{t,j}^j[S_j] \leftarrow z_{t,j}^j[S_j] + \alpha \cdot r_{t+j}[S_j]$
end for



(a) LAMBADA (b) SQuAD V1 (c) MultiNews

Figure 7. Oracle block efficiency over the top-k lattice as a function k. Each plot (a-c) represents a different task, demonstrating the relative improvement in block efficiency of the oracle draft with respect to the vanilla block draft.

E. Confidence across multiple heads

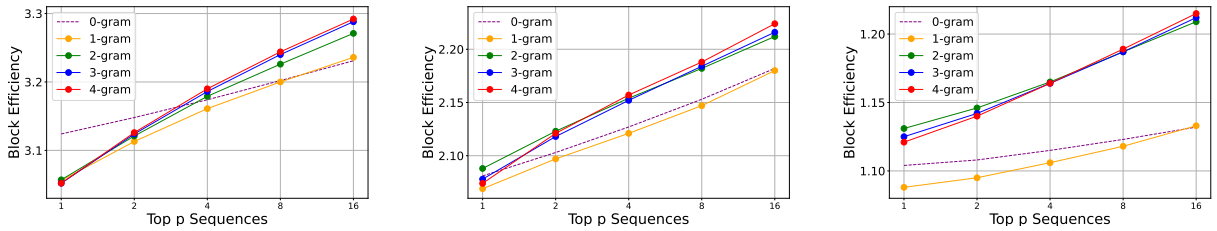
Intuitively, predicting the identity of the i^{th} future token becomes harder as i increases. To better understand this phenomenon, we measure the confidence of the predictions by the entropy of the probability distribution. In Figure 6a, we plot the normalized histogram of entropy of each head on the LAMBADA task. From the normalized histogram, it is clear that the entropy increases as we move from first head to the last head, which agrees with our intuition that hardness of predictions increases as i increases.

However, we observed that the entropy of heads does not increase monotonically for all tasks. Let $\overline{\mathbb{H}}[i]$ be the average entropy of head i on a particular corpus, and let $h_{\max} = \max_k \{k : \forall i < k, \overline{\mathbb{H}}[i] \leq \overline{\mathbb{H}}[i+1]\}$, be the index of the largest head such that the average entropy of each head increases monotonically to that point. We observed a strong correlation between $h_{\max}$ and block efficiency (Figure 6b). Heads with lower entropy (indicating more confident predictions) intuitively contribute more to efficiency. Nonetheless, simply maximizing the number of low-entropy heads is not optimal, but rather incorporating progressively higher entropy heads, up to a certain point, can benefit decoding efficiency. A linear regression confirms this with an R-value of $\mathbf{0.77}$. This analysis suggests that BPD head entropy could be used as a proxy for block efficiency, and thus inference latency.

F. Repairing repetitions

In **Section 4**, we note that vanilla block drafts are prone to token-level repetition and that rescoring with a simple language model reduces the incidence of this. Although rescoring reduces repetition overall in drafts, is this driving improvements in block efficiency? To answer this, we compared the drafts generated by greedy rescoring with the 61M parameter neural rescorer against vanilla drafts. Time step instances were considered wins/ties/losses based on the accepted prefix length of the rescored draft vs. vanilla draft. Table 6 displays the win frequency across tasks along with the percentage of wins/losses attributed to introducing/eliminating repetition.

Note that in the tasks where rescoring improves block efficiency the most, NewsRoom and MultiNews, a high percentage of those repaired instances are driven by fixing erroneously repeated tokens. In fact, for MultiNews, 66.23% of block drafts are improved through repetition repair. We also evaluated the performance of rescoring with in-domain trained rescoring LMs, but found that they tended to perform no better than C4-trained LMs (**Appendix H**).



(a) LAMBADA (b) SQuAD V1 (c) MultiNews

Figure 8. Block efficiency of p-best n-gram BPD methods as a function of the number of top p sequences verified in parallel. The block efficiency of the methods is evaluated with the the same number of paths extracted from the top-16 lattice.

Table 6. Wins, ties, and losses of 61M parameter neural-rescored drafts and vanilla drafts. "% Repair" corresponds to instances where the rescored draft eliminates repetition and "% Regress" corresponds to instances where the rescored draft introduces repetition.

Dataset	Ties	Win			Loss		
		Total	% Repair	% Regress	Total	% Repair	% Regress
LAMBADA	631.5K	5804	27.95	0.05	9466	2.01	0.06
SQuAD V1	104.4K	1624	12.68	8.13	6325	2.53	12.28
CNN/Daily	965.0K	5928	23.20	0.67	17863	3.19	0.48
SAMSum	12.1K	2462	17.91	23.56	867	18.57	16.72
MultiNews	1.45M	294856	44.41	7.45	50209	22.21	5.37
XSUM	262.0K	36010	29.87	0.77	6826	4.19	10.99
NewsRoom	251.3K	79710	66.23	0.60	6492	2.85	7.39

G. Further experimental evaluation

Figure 7 provides additional oracle block efficiency over various tasks. Moreover, Figure 8 further supports the impact of lattice rescoring on block efficiency across various tasks.

H. Rescoring with in-domain language models

Table 7. Block efficiency from rescoring with in-domain trained rescoring models for 2-gram and 61M parameter neural rescorer.

Dataset	2-gram		neural-61M	
	C4	In-domain	C4	In-domain
SQuAD V1	2.09	2.04	2.10	2.06
CNN/Daily	1.73	1.73	1.73	1.72
SAMSUM	1.31	1.22	1.39	1.24
MultiNews	1.13	1.14	1.25	1.16
XSUM	1.17	1.18	1.23	1.14
NewsRoom	1.20	1.22	1.29	1.11

We found that in-domain rescorers performed no better than rescorers only trained on C4. We suspect this is due to a lack of sufficient finetuning data and that the main benefit of rescoring comes from discouraging unnatural artifacts such as repetition from the original BPD draft. Table 7 shows block efficiency after rescoring using in-domain models for all tasks besides language modeling.

Neural rescorers were finetuned from C4-pretrained checkpoints. n-gram models were trained from scratch, and unseen vocabulary was added as unigram arcs with trivial weight (negative log probability of 1000.0). This was done to ensure that all paths through the lattice were assigned non-zero probability by the n-gram model. In previous experiments, we also tried interpolating the in-domain n-gram model with a unigram model trained on C4, and observed similar performance as simply adding unseen unigrams.

I. Interpolation weights tuned per task

We tuned the interpolation weight, α for the 94M parameter neural and 4-gram LM rescorers, and then used this weight to rescore with all other models of that same class. 100 examples from each task's heldout set were set aside for tuning, to maximize block efficiency. The remainder of examples were used for evaluation. We swept over $\alpha \in \{0.1, 0.5, 0.75, 0.9, 1.0, 1.1, 1.5, 2.0, 5.0, 10.0\}$.

Note that for tasks where lattice rescoring was unhelpful, the interpolation weight, α is tuned to place much higher weight on the block draft logits (Table 8). This is a signal that the rescorer does not provide additional information over the original block heads.

Table 8. Tuned interpolation weight per task for neural and *n*-gram rescoring.

Dataset	**Neural**	***n*-gram**
LAMBADA	0.1	0.1
SQuAD V1	1.0	0.75
SAMSum	5.0	1.5
CNN/Daily	0.1	0.1
MultiNews	5.0	2.0
XSUM	1.5	1.1
NewsRoom	5.0	2.0

J. Local rescoring impact on block efficiency

Table 9 reveals the impact of different rescoring methods on the block efficiency of the block lattice, offering valuable insights into their effectiveness across diverse tasks and models, supporting the investigations in **Figure 2**.

- **Limited improvement for high baselines**: For tasks with already high initial block efficiency (LAMBADA, CNN/DailyMail), rescoring offers minimal or even negative changes in block efficiency compared to the baseline BPD system. This suggests that for well-calibrated models with hierarchical confidence ordering, the standard BPD already achieves significant speed improvements, leaving limited room for further gains through rescoring.

- **Efficacy for poor baselines**: In tasks with less calibrated predictions (SQuAD V1, XSUM, NewsRoom), rescoring using both *n*-gram and neural language models demonstrably improves block efficiency. Notably, neural rescoring with larger models (61M and 94M parameters) achieves the highest efficiency gains in these tasks, reaching up to 19.44% improvement in NewsRoom. These results highlight the potential of rescoring to refine predictions and enhance efficiency for models exhibiting calibration issues.

- **Task-specific effectiveness**: The level of improvement from rescoring varies across different summarization tasks (MultiNews, XSUM, NewsRoom). While all show positive responses, NewsRoom exhibits the most significant gains, suggesting that the effectiveness of rescoring can be task-dependent.

- **Comparison with oracle efficiency**: The 'Oracle' columns present the upper bound achievable if only the most likely token at each step is chosen with perfect hindsight (k=2 and k=16). While significant gaps remain between current results and the oracle, the observed improvements from rescoring demonstrate progress towards closing this efficiency gap.

Overall, these findings suggest that local rescoring methods can be a valuable tool for enhancing BPD efficiency, particularly for models with less calibrated predictions. Further exploration of advanced rescoring strategies, especially in conjunction with larger neural language models, holds promise for achieving even closer-to-oracle efficiency levels.

K. Ablation on the number of heads in the blockwise parallel LM

Table 10 summarizes the block efficiency for different head configurations across various language tasks with the same settings discussed in Figure 3.

Table 9. Block efficiency after rescoring of the block lattice. Green circles (●) indicate improvement over the Baseline (BPD), with the percentage changes in block efficiency shown in brackets relative to the Baseline. Red circles (●) denote no improvement.

Task	Dataset	Baseline BPD	Global rescoring 2-gram BPD	Global rescoring 3-gram BPD	Global rescoring 4-gram BPD	Local rescoring neural-32M BPD	Local rescoring neural-61M BPD	Local rescoring neural-94M BPD	Oracle (k=2)	Oracle (k=16)
LM	LAMBADA	**3.12**	3.06 (-1.92%) ●	3.05 (-2.24%) ●	3.05 (-2.24%) ●	3.08 (-1.28%) ●	3.10 (-0.64%) ●	3.05 (-2.24%) ●	3.22	3.67
QA	SQuAD V1	2.08	2.09 (+0.48%) ●	2.08 (0.00%) ●	2.07 (-0.48%) ●	**2.10 (+0.96%)** ●	**2.10 (+0.96%)** ●	2.07 (-0.48%) ●	2.16	2.45
S-SUM	CNN/Daily	**1.74**	1.73 (-0.57%) ●	1.73 (-0.57%) ●	1.73 (-0.57%) ●	1.73 (-0.57%) ●	1.73 (-0.57%) ●	1.73 (-0.57%) ●	1.84	2.26
	SAMSum	1.27	1.31 (+3.15%) ●	1.31 (+3.15%) ●	1.29 (+1.57%) ●	1.33 (+4.72%) ●	**1.39 (+9.45%)** ●	1.21 (-4.72%) ●	1.23	1.95
L-SUM	MultiNews	1.10	1.13 (+2.73%) ●	1.13 (+2.73%) ●	1.12 (+1.82%) ●	**1.25 (+13.64%)** ●	**1.25 (+13.64%)** ●	1.20 (+9.09%) ●	1.13	1.43
	XSUM	1.13	1.17 (+3.54%) ●	1.17 (+3.54%) ●	1.16 (+2.65%) ●	1.18 (+4.42%) ●	**1.23 (+8.85%)** ●	1.17 (+3.54%) ●	1.17	1.55
	NewsRoom	1.08	1.20 (+11.11%) ●	1.18 (+9.26%) ●	1.18 (+9.26%) ●	**1.29 (+19.44%)** ●	**1.29 (+19.44%)** ●	1.17 (+8.33%) ●	1.15	1.50

- **General trend**: Both performance and block efficiency tend to increase with the number of heads, up to a point. This suggests that using more heads allows the model to capture richer contextual information and make more accurate predictions.

- **Efficiency trade-off**: While increasing heads generally improves block efficiency, it also increases the memory for verification stages. Therefore, the optimal number of heads depends on the balance between desired block efficiency and available resources.

Table 10. Test performance per task. Test performance of each finetuned model and block efficiency are shown as a function of heads ($h \in 3, 6, 9$). Tasks inclue Language Modeling (LM), extractive Question Answering (QA), and both Long and Short Summarization (L-Sum & S-Sum). The metric for LM is perplexity, for QA is exact match, and for all the remaining (summarization) tasks, the metric is ROUGE-L.

Task	Dataset	Performance	# of Heads (h) 3	6	9
LM	LAMBADA	7.88	1.79	2.84	3.12
QA	SQuAD V1	57.60	1.53	2.03	2.08
S-SUM	CNN/Daily	39.85	1.60	1.71	1.74
	SAMSUM	37.66	1.18	1.25	1.27
L-SUM	MultiNews	23.08	1.08	1.08	1.10
	XSUM	52.15	1.11	1.12	1.13
	NewsRoom	39.85	1.07	1.08	1.08

L. Efficiency of rescoring block drafts

To enhance our understanding of block rescoring within the realm of contemporary deep learning hardware environments, we present an in-depth examination focused on TPU/GPU utilization and the overhead incurred by n-gram rescoring. This analysis is divided into two parts: (1) an analysis of block rescoring through the lens of TPU/GPU utilization, and (2) empirical benchmarks of n-gram lattice rescoring. The major takeaways are as follows.

Memory bandwidth (HBM ⇔ SRAM) A critical factor in the performance of deep learning applications is the efficient management of memory bandwidth between High Bandwidth Memory (HBM) and Static Random Access Memory (SRAM) (Dao et al., 2022). Increasing the block efficiency via the block lattice rescoring reduces the average per token parameter and key-value cache I/O that needs to be communicated from HBM to SRAM.

Overhead in n-gram rescoring n-gram rescoring is actually quite efficient. For the size of lattices we consider in this work, moving the lattice from HBM to DRAM, performing n-best n-gram rescoring, and moving the n-best paths back to HBM requires no more than 2 ms per lattice.

L.1. Hardware utilization

We compare our approach against traditional Autoregressive LMs across several metrics (Table 11):

Table 11. Comparative analysis of per decoded token efficiency metrics across block rescoring methods and the standard autoregressive LM (batch size=1). This table shows the average block efficiency, parameter I/O, key-value (KV) cache I/O at varying sequence lengths, and FLOPS—evaluated on a per-token basis under the condition of batch size 1.

Component	Autoregressive	Base BPD	4-gram BPD	Neural-61M BPD	16-best 0-gram BPD	16-best 4-gram BPD
Avg. Block Efficiency	1.000	1.646	1.657	1.724	1.717	1.797
Parameter I/O (GB)	3.000	1.823	1.811	1.811	1.747	1.669
KV Cache I/O (GB) - Seq_len 128	0.113	0.074	0.073	0.076	0.140	0.134
KV Cache I/O (GB) - Seq_len 512	0.453	0.280	0.278	0.290	0.338	0.323
KV Cache I/O (GB) - Seq_len 1024	0.906	0.555	0.552	0.574	0.602	0.575
KV Cache I/O (GB) - Seq_len 2048	1.812	1.106	1.098	1.144	1.129	1.079
FLOPS (T)	0.931	0.57	0.567	0.635	0.621	0.593

Memory bandwidth and compute efficiency The block rescoring variants achieve significant reductions in Parameter I/O and KV Cache I/O compared to autoregressive decoding, suggesting BPD methods' ability to reducing inference times by mitigating the primary latency bottleneck—memory bandwidth. Advances in TPU/GPU architecture ensure that an increase in FLOPS per token has a minimal effect on latency, confirming our strategy's capacity to navigate the complexities of memory bandwidth efficiently.

Comparative latency impact A consistent decrease in memory bandwidth utilization across blockwise parallel LMs, including those leveraging LM rescoring and parallel processing strategies, illustrates our approach's contribution to accelerating inference speed. This underscores the practicality and applicability of our enhancements in promoting more efficient language model inference within state-of-the-art computational frameworks.

L.2. Overhead of n-gram rescoring

While the majority of computational efforts in block rescoring are dedicated to TPU/GPU utilization, the implementation of n-gram rescoring introduces additional overheads. These are primarily attributed to CPU computations and the data transfer between the CPU and HBM. This section provides a comprehensive examination of these overheads, drawing on benchmarks from rescoring experiments with a 4-gram C4 LM.

Benchmarks for 4-gram C4 LM rescoring We conducted benchmarks on rescoring lattices with a 4-gram C4 LM of $\approx$100M n-grams. The average latency observed across 10 runs for different numbers of the shortest paths is summarized in the Table 12:

Table 12. Average latency for N-best rescoring an 8-time step lattice with 16 arcs per time step. N, the number of shortest paths, is varied from 1 to 16.

# Shortest Paths	N-best Rescoring Latency (ms)
1	1.630
2	1.751
4	1.878
8	1.871
16	1.983

Notably, rescoring with a large 4-gram LM averages less than 2 milliseconds for extracting up to 16 globally-best paths, despite the lattice containing approximately 4.29 billion possible paths. In our initial experiments, increasing the size of the *n*-gram LM had little effect on n-best rescoring latency, indicating that enhancements leading to higher block efficiency will incur little additional latency, provided the LM fits within DRAM.

Latency is predominantly influenced by lattice size, particularly the number of top-k tokens per time step and the number of time steps, as depicted in Table 13:

The benchmarks highlight the fact that the additional overhead introduced by *n*-gram rescoring, though present, should not significantly impact overall latency.

Table 13. 1-best rescoring latency by the 4-gram C4 LM for varying lattice sizes.

Number of time steps	Top-k per time step	1-best rescoring latency (ms)
4	2	1.038
4	4	1.050
4	8	1.130
4	16	1.237
8	2	1.061
8	4	1.144
8	8	1.234
8	16	1.630
16	2	1.102
16	4	1.206
16	8	1.558
16	16	2.174