Accelerating Blockwise Parallel Language Models with Draft Refinement

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

Abstract

Autoregressive language models have achieved remarkable advancements, yet their potential is often limited by the slow inference speeds associated with sequential token generation. Blockwise parallel decoding (BPD) was proposed by Stern et al. [42] as a method to improve inference speed of language models by simultaneously predicting multiple future tokens, termed *block drafts*, which are subsequently verified by the autoregressive model. This paper advances the understanding and improvement of block drafts in two ways. First, we analyze token distributions generated across multiple prediction heads. Second, leveraging these insights, we propose algorithms to improve BPD inference speed by refining the block drafts using task-independent *n*-gram and neural language models as lightweight rescorers. Experiments demonstrate that by refining block drafts of open-sourced Vicuna and Medusa LLMs, the mean accepted token length are increased by 5-25% relative. This results in over a 3x speedup in wall clock time compared to standard autoregressive decoding in open-source 7B and 13B LLMs.

1 Introduction

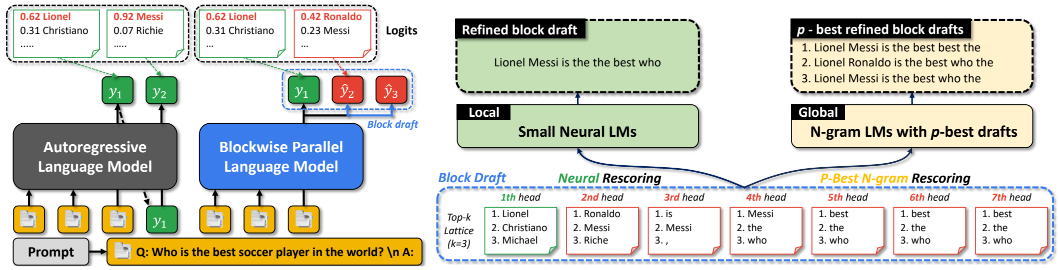
The landscape of natural language processing has been profoundly reshaped by recent advances in autoregressive language models [3, 48, 34, 37, 47]. These models have shown remarkable proficiency across a range of text generation tasks, including applications like question answering [38] and summarization [17]. However, a significant obstacle to their wider application is high inference latency, particularly for extremely deep models with hundreds of billions of parameters [18, 35, 7]. This latency, intrinsic to decoding with autoregressive language models (LMs), imposes considerable computational burdens and limits real-time deployment.

In response to these challenges, the field has seen a shift towards decoding methods aimed at reducing the inference latency in large language models (LLMs). One promising development is the concept of blockwise parallel decoding (BPD) [42, 31, 4]. Unlike autoregressive decoding, which generates one token at a time, blockwise parallel LMs are outfitted with a set of prediction heads that 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 both on the time it takes to generate a block draft and verification of the draft's agreement with the original LM's output (referred to as *base LM* from here on) (Figure 1a). Unlike standard autoregressive LMs that generate tokens sequentially — ensuring consistency with all preceding tokens (e.g., 'Messi' following 'Lionel') — BPD employs a non-autoregressive drafting strategy. Here, blockwise parallel LMs simultaneously predict multiple token drafts (e.g., 'Lionel' and 'Ronaldo'), each position produced independently. The primary challenge in BPD drafting is ensuring that these concurrently-generated tokens are consistent with each other. An effective block drafter should prefer coherent sequences, such as 'Lionel Messi' over less coherent combinations like

[*]Work done while at Google Research as a student researcher. Corresponding authors: <kimtaehyeon610@gmail.com> and <adbenton@google.com>.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).



(a) Example of block drafts

(b) Output of our proposed rescoring algorithms

Figure 1: (a) Illustration of two tokens that are decoded by autoregressive 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 or global n-gram rescoring, which selects the p most probable sequences by dynamic programming, for batched verification.

'Lionel Ronaldo', which would be improbable under a reasonable LM. The focus of this paper is on improving the quality of block drafts without altering the underlying model parameters.

2 Our contributions

This paper first investigates properties of the drafts from blockwise parallel LMs across seven tasks. These analyses are based on modest, 1.5 billion (B) parameter LMs. Given our observations, we propose lattice rescoring algorithms to produce higher quality block drafts. Finally, we apply these lattice rescoring algorithms to improve the drafts from large (7B/13B parameter) open-source LLMs, reducing mean per-token latency relative to both standard BPD and Medusa decoding across tasks.

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 5.1**).

Confidence of different heads We analyze the distribution of probabilities within each block head. Our empirical analysis reveals an interesting property of BPD: the block drafter tends to be more confident with initial tokens, and becomes progressively less confident for subsequent tokens. We find that the confidence of block heads correlates strongly with the quality of the block drafter (**Section 5.2**).

Oracle top-k block efficiency In the standard BPD algorithm (**Algorithm 1**), the most likely token at each head is generated as the draft. As mentioned above, this is prone to two issues: (1) this sequence might contain unnatural, consecutive repetitions and (2) the model might not be confident of 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 [28, 46]. We ask whether 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 define 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 5.3**).

2.2 New block draft algorithms with lightweight rescoring

Based on these observations, we propose two algorithms to leverage the top-k predictions at each head and improve average latency for open-source LLMs (Figure 1b). We show that these algorithms can also reduce the average latency in Medusa decoding [4], a recent popular extension of BPD (**Section 7**). Neither of these algorithms requires changes to the underlying blockwise parallel LMs.

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 (**Section 6.1**). 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 multi-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

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 */
  $z_t^i \leftarrow \mathcal{M}_{\theta,i}^h(\cdot|\bar{x}, y_{\leq t}), \forall i \leq h.$
  $\hat{y}_{t+1}, \hat{y}_{t+2}, \ldots, \hat{y}_{t+h} \leftarrow \arg\max_{y \in \mathcal{V}} z_t^1[y], \arg\max_{y \in \mathcal{V}} z_t^2[y], \ldots, \arg\max_{y \in \mathcal{V}} z_t^h[y]$
  /* Stage 2: Verify */
  for $j \leftarrow 0, \ldots, h$ in parallel **do**
    $\hat{z}_{t+j} \leftarrow \mathcal{M}_\theta(\cdot|\bar{x}, y_{\leq t}, \hat{y}_{t+1}, \hat{y}_{t+2}, \cdots, \hat{y}_{t+j})$
  end for
  /* Stage 3: Accept */
  $n \leftarrow \max\{n : \hat{y}_{t+j} = \arg\max_{y \in \mathcal{V}} \hat{z}_{t+j-1}[y], 1 \leq j \leq n\}$
  $t \leftarrow t + n + 1$, $y_{t+j} \leftarrow \hat{y}_{t+j}, \forall 1 \leq j \leq n$ and $y_{t+n+1} = \arg\max_{y \in \mathcal{V}} \hat{z}_{t+n}[y]$
end while

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 by the blockwise parallel LM (**Section 6.2**).

There are two critical distinctions between the proposed algorithms: the amount of context/expressive power available to each class of rescoring model, and fundamental limitations of decoding with each class. 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 exponentially-sized set of possible draft candidates. Detailed algorithms are given in **Section 6.1**.

3 Preliminaries

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 $\bar{x}$. $z_t = \mathcal{M}_\theta(\cdot|\bar{x}, y_{<t})$ is a vector of logits, $z_t \in \mathbb{R}^{|\mathcal{V}|}$, where $\mathcal{V}$ is the vocabulary over tokens. Let $z_t[y]$ denote the logit of symbol y. These logits define a conditional probability distribution at each time step $p_\theta(y|\bar{x}, y_{\leq t}) = \frac{e^{z_t[y]}}{\sum_{y' \in \mathcal{V}} e^{z_t[y']}}$, which by the chain rule yields $p_\theta(y_{\leq T}|\bar{x}) = \prod_{t=1}^T p_\theta(y_t|\bar{x}, y_{<t})$. Sequences are generated autoregressively, either through ancestral sampling from some form of the conditional next token distribution [19], or by a beam search through the space of possible sequences to return a probable sequence. For simplicity, in this paper we focus on greedy decoding, where at each step the next token is predicted as $y_{t+1} = \arg\max_{y \in \mathcal{V}} p_\theta(y|\bar{x}, y_{\leq t})$. The goal of BPD is to predict the same tokens as the base model, albeit efficiently.

Blockwise parallel decoding Let $\mathcal{M}_\theta^h$ be a blockwise parallel LM with block size h and let $z_t^i = \mathcal{M}_{\theta,i}^h(\cdot|\bar{x}, y_{\leq t})$ be the vector of logits corresponding to the i^{th} block given context $\bar{x}, y_{\leq t}$. This model employs h distinct feedforward neural (FFN) layers, each with a single hidden layer, atop the base 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 initial analyses, 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. Similar to [42], the first head is the same as the base LM, i.e., $z_t^1 = \mathcal{M}_{\theta,1}^h(\cdot|\bar{x}, y_{\leq t}) = \mathcal{M}_\theta(\cdot|\bar{x}, y_{\leq t}) = z_t$ and the hope is that for subsequent heads $i \geq 2$, $\mathcal{M}_{\theta,i}^h(\cdot|\bar{x}, y_{\leq t}) \approx \mathcal{M}_\theta(\cdot|\bar{x}, y_{\leq t+i-1})$.

Algorithm 1 describes the BPD greedy decoding procedure. We outline the algorithm below and refer readers to [42] for additional details.

1. **Predict:** $\mathcal{M}_\theta^h$ generates a draft of h token predictions $\hat{y}_{t+1}, \hat{y}_{t+2}, \ldots, \hat{y}_{t+h}$, conditioned on the prompt, $\bar{x}$, and existing generated text, $y_{\leq t}$ (i.e., $\hat{y}_{t+i} = \arg\max_{y \in \mathcal{V}} z_t^i[y]\ \forall i \leq h$). Since the first head is same as the base LM, $\hat{y}_{t+1}$ is identical to y_{t+1}, the output of the base LM with greedy decoding.

2. **Verify:** In order to verify the predicted drafts, the base LM greedily generates next-token logits $\{\hat{z}_t, \ldots, \hat{z}_{t+h}\}$ conditioned on the existing prefix and block draft i.e., $\hat{z}_{t+i} = \mathcal{M}_\theta(\bar{x}, y_{\leq t}, \hat{y}_{t+1}, \hat{y}_{t+2}, \ldots, \hat{y}_{t+i})$ for $i \in \{0, 1, \ldots, h\}$. Verification amounts to check-

Table 1: 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 [33]	7.88	3.12
QA	SQuAD V1 [38]	57.60	2.08
S-SUM	CNN/Daily [17]	39.85	1.74
	SAMSUM [14]	37.66	1.27
L-SUM	MultiNews [12]	23.08	1.10
	XSUM [32]	52.15	1.13
	NewsRoom [15]	39.85	1.08

Table 2: Sample outputs from blockwise parallel LMs finetuned per task. 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.

ing which block draft tokens match the autoregressive greedy decode from the base LM: $(\arg\max_{y\in\mathcal{V}} \hat{z}_{t+i}[y]) == \hat{y}_{t+i+1}$. Note that the verification of all positions can be performed in parallel under the assumption that the base LM is a decoder-only transformer.

3. **Accept:** Finally, the length of the longest contiguous prefix n where draft tokens match the base LM's greedy decode is identified. Since the first head is the same as the base LM, the first token $\hat{y}_{t+1}$ is always accepted. After accepting the tokens, one *free token* can be obtained as the conditional probability of the base LM based on accepted tokens have already been calculated. Thus, the decoded sequence is extended by $n+1$ tokens and we iterate. Typically, not all h tokens are accepted, with some draft tokens discarded. As the block generation has minimal overhead compared to the base LM's forward pass, even modest gains in accepted prefix length justify the cost of block draft generation.

4 Analysis setup

We train a ≈ 1.5 billion (B) parameter decoder-only transformer LM with 9 heads, and investigate the drafts produced by this modest blockwise parallel LM.[2] The 1.5B model and all auxiliary LMs were pretrained on (English) C4 [36] with the causal next token prediction objective tokenized with the GPT3 subword vocabulary [3]. For the 1.5B blockwise parallel LM, 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 ≈ 200B input tokens in total. Model training/inference was run on TPUv3/TPUv4 [20], and implemented in Jax [2].

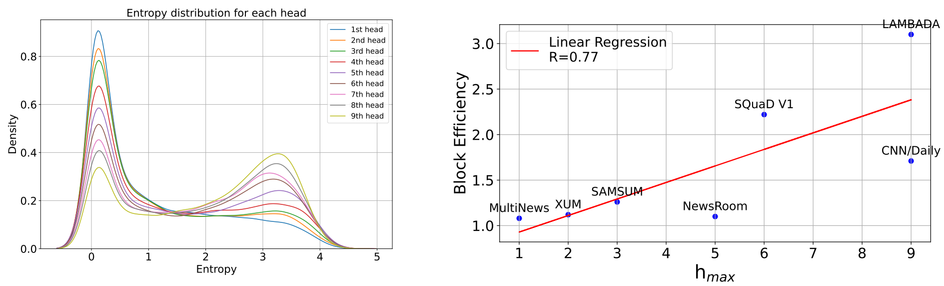
We evaluate the potential latency improvement of block drafts by *block efficiency* [28, 46]. In this context, block efficiency represents the theoretical speedup compared to standard greedy decoding. It is defined as the average number of tokens decoded per serial call to the blockwise parallel LM. 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 LM's predictions are accepted and integrated into the final sequence. This ensures that generated text is identical to the base 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.

We investigate the drafts produced by this 1.5B blockwise parallel LM on LAMBADA [33] (language modeling), SQuAD V1 [38] (extractive QA), along with five summarization tasks: XSUM [32], MultiNews [12], SAMSum [14], NewsRoom [15] and CNN/DailyMail [17]. For each task other than language modeling, we finetune the blockwise parallel LM for that task.[3]

[2] This model follows the original blockwise parallel LM, with a modification: we use decoder-only models instead of the T5 encoder-decoder architecture. Other than this, the LM architecture is consistent with that proposed in Stern et al. [42].

[3] Details are given in **Appendix D**.



(a) Entropy distributions across block draft heads (b) Correlation between block efficiency and $h_{\max}$

Figure 2: (a) Entropy distributions across block draft heads on LAMBADA [33]. 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 nearly monotonically.

Table 1 shows that block efficiency varies dramatically across task.[4] Language modeling, most closely matching the pretraining objective, achieves the highest block efficiency followed by the context-constrained task of extractive question answering. Table 2 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.

5 Exploration of block drafts

5.1 Consecutive repetition

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 [16]. Table 3 shows the proportion of consecutive tokens in block drafts that are identical to each other, along with the average maximum length of repeated sequences in block drafts across all decode time steps. 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. Strings of repeated tokens are unnatural, and unlikely to be generated by a strong base language model. Rescoring the top-k lattice with even a simple language model eliminates a significant amount of repetition, reducing the percentage of consecutive repeated tokens from between **9.9%** to **24.5%**, depending on the task.

Table 3: 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 (upper bound of 9, the number of block draft heads). Higher values correspond to more egregious repetition in drafts.

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**

5.2 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

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

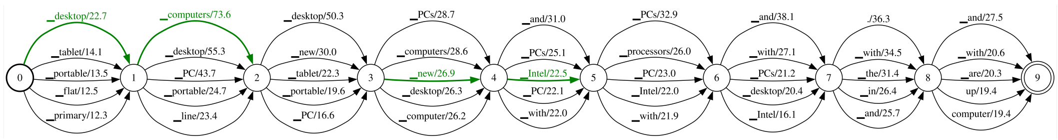


Figure 3: An example of a top-5 sausage lattice on a NewsRoom example. Edge weights correspond to logits. Edges at each time step are ordered in descending weight and green, bolded edges correspond to candidates matching the greedy decode over the next nine tokens: *"... desktop computers with new Intel Corp processors that it ..."*. The initial node in this graph is state 0 and the final node is 9.

token-level probability distribution for each head. In Figure 2a, we plot the normalized histogram of entropy of each head on the LAMBADA dataset. 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 token prediction becomes more difficult as i increases.

However, we observed that the head entropy does not increase monotonically for all tasks as a function of i. 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 2b). Heads with lower entropy (indicating more confident predictions) intuitively contribute more to efficiency. A linear regression confirms this with an R-value of $\mathbf{0.77}$. This analysis suggests that the entropies of block heads could be used as a proxy for block efficiency, and thus inference latency.

5.3 Oracle top-k block efficiency

Oracle efficiency The concept of oracle block efficiency 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. This data structure is a weighted directed graph, which succinctly represents all possible drafts (and their score under the blockwise parallel LM) that could be formed from selecting one of k tokens from each of the h heads (Figure 3). In the automatic speech recognition and machine translation communities, it is known as a "confusion network" [26, 41].

Question: Who is the best soccer player in the world? Answer:

Lionel	*Ronaldo*	*is*	*Messi*	*best*	~~*best*~~	~~*best*~~
Christiano	*Messi*	*Messi*	*the*	*the*	~~*world*~~	~~*in*~~
Michael	*Riche*	*,*	*who*	*who*	~~*the*~~	~~*world*~~

Accepted *Rejected*

Figure 4: Illustration of the output through oracle selection. For a given top k tokens of 3, if we can choose the oracle path successfully, the block efficiency can be improved from 1 to 5.

Given the top-k lattice at each decoding step, we identify an *oracle path* that represents the path through the lattice that maximizes the length of the accepted prefix. This exercise, as shown in Figure 4, gives us insight into how much headroom exists in improving block drafts.

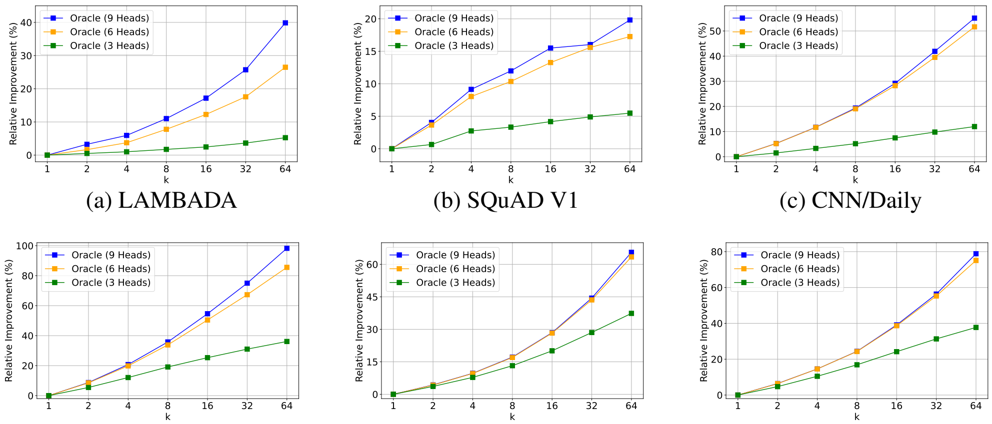
Potential headroom from oracle selection Oracle drafting is not practical, but rather a reference point. Analyzing the gap between actual BPD performance and the oracle upper bound (Figure 5) helps us to understand the limitations of the original block drafts and potential areas for improvement. Additionally, exploring oracle efficiency as a function of the k in the top-k lattice, demonstrates how "close" the block draft was to producing a stronger draft.

6 Lattice rescoring with lightweight rescorers

Having explored the properties of block draft predictions, we propose two drafting algorithms to improve block efficiency through rescoring of the top-k lattice with lightweight auxiliary LMs. This section presents techniques for rescoring the top-k lattice along with empirical results.

Each of these algorithms is a modification of the block drafted in **Stage 1** in Algorithm 1. Instead of using the most 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: $|S_1| \times |S_2| \times \ldots |S_h| = k^h$. [5]

[5]The number of combinations can be reduced to k^{h-1} by using the fact that the first head is the same as the base LM and hence we can set $|S_1| = 1$.



(a) LAMBADA (b) SQuAD V1 (c) CNN/Daily

(d) SAMSUM (e) MultiNews (f) XSUM

Figure 5: Oracle block efficiency over the top-k lattice as a function k. Each plot (a-f) represents a different task, demonstrating the relative improvement in block efficiency of the oracle draft with respect to the standard block draft as a function of the number of block draft heads used.

In this lattice, any path from the start to final state represents a viable draft. Two algorithms are proposed to select a small number of h-length drafts from this lattice, which are then passed to the verification step. The first algorithm employs neural autoregressive transformers (**Section 6.1**), while the second utilizes n-gram language models (**Section 6.2**).

6.1 Local rescoring via neural models

A simple approach uses a small neural rescorer LM, interpolating between the logits of the rescorer LM and vanilla block draft logits with an interpolation weight (**Algorithm 2**). Recall that z_t^j is the vector of logits corresponding to the j^{th} block. Let S_j denote the set of symbols with top-k values in the logits vector z_t^j. The rescored prediction for head j is given by:

$$z_t^j[S_j] \leftarrow z_t^j[S_j] + \alpha \cdot r_{t+j}[S_j],$$

where α is the weight placed on the rescorer's prediction and r_{t+j} are the corresponding logits predicted by the small neural rescoring model, when conditioned on the sequence $y_{\le t}, \hat{y}_{t+1}, \ldots, \hat{y}_{t+2}, \ldots, \hat{y}_{t+j-1}$. We also set logits for symbols outside set S_j (S_j^c) to be negative infinity, which corresponds to zero probability. Note that we do not rescore the first head as it is the same as the base LM. We then run **Algorithm 1**, where instead of using logits directly from the BPD model, we use the rescored logits to generate the draft. We experiment with decoder-only transformers having 32, 61, and 94 million (M) weight parameters (**Appendix D**).

Algorithm 2: Local rescoring via neural models

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

$z_t^i \leftarrow \mathcal{M}_{\theta,i}^h(\cdot|\bar{x}, y_{\le t}), \forall i \le h$
$S_i \leftarrow \text{TOP-}k(z_t^i), \forall i \le h$
$\hat{y}_{t+i} \leftarrow \arg\max_{y \in \mathcal{V}} z_t^i[y], \forall i \le h$
/* Local lattice rescoring */
for $j \leftarrow 2, \ldots, h$ in parallel **do**
  $r_{t+j} \leftarrow \mathcal{M}_{\theta_r}(\cdot|x, y_{\le t}, \hat{y}_{t+1}, \ldots, \hat{y}_{t+j-1})$
  $z_t^j[S_j] \leftarrow z_t^j[S_j] + \alpha \cdot r_{t+j}[S_j]$
  $z_t^j[S_j^c] \leftarrow -\infty$
end for

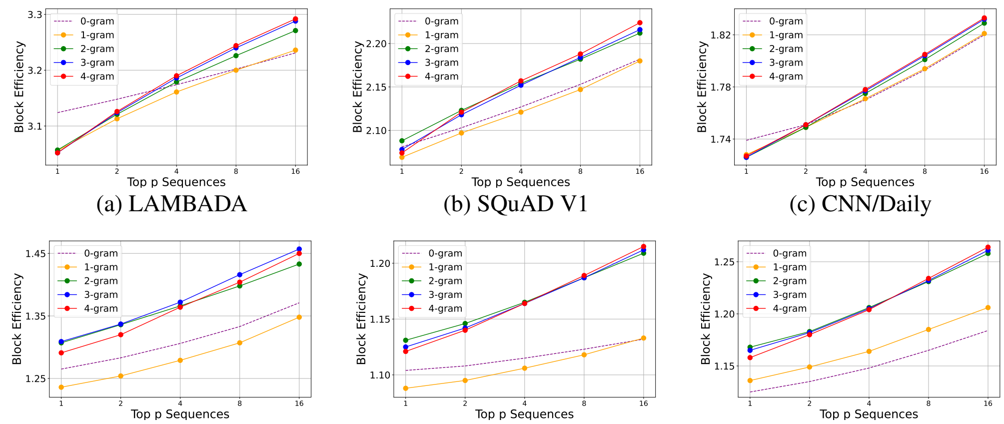
6.2 Global n-gram rescoring

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 except the first position token with an n-gram model, select the top p most likely sequences and pass them to the verification stage. When $p = 1$, we refer to this as *n-gram rescoring* and when $p > 1$, we refer to this as *p-best n-gram BPD*.

While global rescoring typically yields better results compared to local rescoring, rescoring k^h sequences with a neural LM and selecting the most likely sequence would take time $O(k^h)$, which is computationally prohibitive in most cases. Hence, we take advantage of n-gram LMs, which are unique in that one can efficiently select the most likely rescored sequence in time $\text{poly}(k, h)$, using

Table 4: Block efficiency of rescoring methods over the top-16 lattice. '16-best 0-gram BPD' indicates performance of 16-best draft 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



(a) LAMBADA (b) SQuAD V1 (c) CNN/Daily

(d) SAMSUM (e) MultiNews (f) XSUM

Figure 6: 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 same number of paths extracted from the top-16 lattice.

dynamic programming. We use the OpenFST library [1] to represent each n-gram LM 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 n-gram LMs are in **Appendix D.3**.

6.3 Empirical evaluation

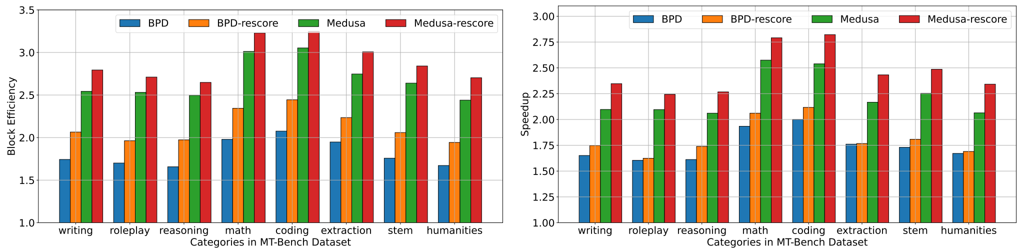
Block efficiency Table 4 and Figure 6 demonstrate the impact of lattice rescoring on block efficiency across various tasks. Autoregressive neural, n-gram LM, and p-best n-gram BPD rescoring all demonstrate improvements in block efficiency, although gains are task-dependent.

- **High initial block efficiency (LAMBADA, CNN/Daily)**: Both rescoring methods show little to no improvement, suggesting that vanilla BPD already produces high quality drafts.
- **Low initial block efficiency (SQuAD V1, SAMSUM, XSUM, NewsRoom)**: Both neural and n-gram augmentatiaons lead to block efficiency gains, particularly with neural LMs achieving the best performance in some cases.

Repairing repetitions In **Section 5.1**, 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

Table 5: Wins, ties, and losses of 61M neural-rescored 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 Total	Win % Repair	Win % Regress	Loss Total	Loss % Repair	Loss % Regress
LAMBADA	631.5K	5.8K	27.95	0.05	9.5K	2.01	0.06
SQuAD V1	104.4K	1.6K	12.68	8.13	6.3K	2.53	12.28
CNN/Daily	965.0K	5.9K	23.20	0.67	17.8K	3.19	0.48
SAMSum	12.1K	2.5K	17.91	23.56	0.9K	18.57	16.72
MultiNews	1.45M	294.9K	44.41	7.45	50.2K	22.21	5.37
XSUM	262.0K	36.0K	29.87	0.77	6.8K	4.19	10.99
NewsRoom	251.3K	79.7K	66.23	0.60	6.5K	2.85	7.39



(a) Block efficiency (b) Speedup ratio

Figure 7: Block efficiency and speedup ratio relative to the standard autoregressive decoding on sub-categories of MT-Bench dataset [53] when greedily decoding with Vicuna 13B.

accepted prefix length of the rescored draft vs. vanilla draft. Table 5 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 E**).

7 Lattice rescoring on open-source blockwise parallel LLMs

Medusa decoding [4] extends BPD by verifying a set of plausible candidates in parallel. Verification is performed efficiently through a tree-attention mechanism, requiring only a single forward pass. Other aspects not explicitly mentioned remain the same as described in Algorithm 1. While Medusa employs tree-attention during decoding to efficiently verify a subset of likely drafts, our approach focuses on rescoring these draft candidates, making them potentially complementary techniques. We explore this synergy by integrating neural rescoring method into Medusa decoding. In this section, we apply rescoring to large open-source LLMs, using Vicuna 7B-v1.3 and Vicuna 13B-v1.3 as base models. We report both block efficiency and speedup ratio achieved relative to standard autoregressive decoding using the SpecBench benchmark [49]. To ensure rigorous verification, we expand our experiments to include a wider range of datasets. We use existing pretrained Medusa heads as the block drafter[6]. Although these base LMs were not trained jointly with the block drafter, this corresponds to the Medusa-1 configuration, which has been shown to result in comparable speedups to jointly trained Medusa models [4]. For lattice neural rescoring, we set k to be the full vocabulary size, using 5 heads with the next-word-prediction LM head as one of the heads, following Algorithm 2. All timings were evaluated on a single NVIDIA A100 80GB GPU with batch size 1.

Figure 7 demonstrates the block efficiency and speedup ratio on MT-Bench [53], comparing greedy BPD and Medusa with and without local rescoring for Vicuna 13B models by setting the interpolation weight α to 1.0. The same analysis on Vicuna 7B is described in the Figure 9, which is detailed in Appendix G. A key observation is that even after increasing model size from 7B to 13B, a relatively small neural model (68M) can effectively serve as the rescoring drafter, showcasing the robustness of our approach. The rescoring model used in these experiments is a decoder-only LM trained on the C4 and ShareGPT datasets[7]. Furthermore, we observe consistent performance improvements across both the original BPD and its extension, Medusa, further validating the efficacy of our local rescoring method. While the speedup gains might not always directly correlate with the increase in block efficiency, we consistently observe performance improvements across all categories. This difference suggests that block efficiency does not always translate into equivalent speedup, likely due to system-level factors. However, there remains potential for further acceleration through additional system-level optimizations.

Table 6 further presents speedup ratios across diverse datasets for Vicuna 7B and 13B models, respectively. We evaluate not only under greedy decoding (Temperature=0.0) but also under temperature sampling (Temperature=0.7, 1.0), employing typical acceptance for verification [4]. Both BPD and Medusa, enhanced with our local rescoring, consistently yield speedup improvements across all

[6] https://huggingface.co/FasterDecoding/medusa-vicuna-7b-v1.3
[7] https://huggingface.co/double7/vicuna-68m

Table 6: Speedup ratio relative to the standard autoregressive decoding for Vicuna models (7B and 13B) on various datasets: MT-bench [53], S-Sum (CNN/Daily), QA [27], GSM8K [8], and RAG [21].

Model	Method	Temperature=0.0					Temperature=0.7					Temperature=1.0				
		MT-bench	S-Sum	QA	GSM8K	RAG	MT-bench	S-Sum	QA	GSM8K	RAG	MT-bench	S-Sum	QA	GSM8K	RAG
Vicuna 7B	BPD	1.780	1.509	1.489	1.696	1.409	1.781	1.523	1.512	1.790	1.496	1.858	1.528	1.613	1.890	1.525
	+ Local rescoring	**1.843** 🟢	**1.534** 🟢	**1.555** 🟢	**1.780** 🟢	**1.501** 🟢	**1.903** 🟢	**1.561** 🟢	**1.666** 🟢	**1.842** 🟢	**1.544** 🟢	**1.998** 🟢	**1.579** 🟢	**1.656** 🟢	**1.954** 🟢	**1.622** 🟢
	Medusa [4]	2.430	2.002	2.045	2.317	1.833	2.425	2.094	2.121	2.563	1.998	2.511	2.096	2.334	2.650	2.010
	+ Local rescoring	**2.482** 🟢	**2.076** 🟢	**2.114** 🟢	**2.357** 🟢	**2.000** 🟢	**2.597** 🟢	**2.228** 🟢	**2.279** 🟢	**2.630** 🟢	**2.139** 🟢	**2.657** 🟢	**2.281** 🟢	**2.386** 🟢	**2.655** 🟢	**2.173** 🟢
Vicuna 13B	BPD	1.745	**1.530**	1.488	1.794	1.483	1.881	1.555	1.559	1.875	1.558	2.043	1.684	1.675	2.078	1.664
	+ Local rescoring	**1.819** 🟢	1.522 🔴	**1.519** 🟢	**1.819** 🟢	**1.501** 🟢	**1.990** 🟢	**1.643** 🟢	**1.761** 🟢	**1.998** 🟢	**1.679** 🟢	**2.188** 🟢	**1.731** 🟢	**1.805** 🟢	**2.206** 🟢	**1.779** 🟢
	Medusa [4]	2.383	2.000	1.986	2.507	1.945	2.559	2.080	2.272	2.700	2.069	2.844	2.278	2.501	2.942	2.256
	+ Local rescoring	**2.467** 🟢	**2.136** 🟢	**2.154** 🟢	**2.519** 🟢	**2.068** 🟢	**2.738** 🟢	**2.211** 🟢	**2.368** 🟢	**2.700** 🟢	**2.293** 🟢	**2.981** 🟢	**2.392** 🟢	**2.574** 🟢	**3.010** 🟢	**2.447** 🟢

Table 7: Speedup ratio of efficient LLM inference methods during greedy decoding.

Method	Vicuna 7B					Vicuna 13B				
	MT-Bench	S-Sum	QA	GSM8K	RAG	MT-Bench	S-Sum	QA	GSM8K	RAG
Sps [5]	1.432	1.394	1.417	1.364	1.568	1.417	1.424	1.362	1.448	1.606
Lookahead [13]	1.818	1.645	1.503	1.865	1.475	1.118	1.007	1.011	1.324	0.963
PLD [39]	1.676	**2.707**	1.162	1.605	1.909	1.528	**2.384**	1.050	1.646	1.876
BPD	1.780	1.509	1.489	1.696	1.409	1.745	1.530	1.488	1.794	1.483
+ Local rescoring	**1.922** 🟢	1.534 🟢	**1.555** 🟢	**1.780** 🟢	**1.501** 🟢	**1.819** 🟢	1.522 🔴	**1.519** 🟢	**1.819** 🟢	**1.501** 🟢
Medusa	2.430	2.002	2.045	2.317	1.833	2.383	2.000	1.986	2.507	1.945
+ Local rescoring	**2.482** 🟢	2.076 🟢	**2.114** 🟢	**2.357** 🟢	**2.000** 🟢	**2.467** 🟢	**2.136** 🟢	**2.154** 🟢	**2.519** 🟢	**2.068** 🟢

settings. Green circles (🟢) indicate further improvements from local rescoring, while red circles (🔴) denote no improvement. Notably, even with larger models, our method delivers consistent gains in latency. Table 7 compares the speedup ratio of various efficient LLM inference methods. While other methods offer speedup in certain scenarios, their performance is inconsistent across task and decoding setting. Overall, we find that local neural rescoring consistently provides additional speedup over both BPD and Medusa decoding.

***n*-gram rescoring** While C4 n-gram lattice rescoring yielded 1-best candidates with improved block efficiency for many tasks, the gains were not as stark as locally rescoring with the Vicuna-68m model (Figure 8). The discrepancy is partly due to domain mismatch, since the C4 data used to train the n-gram rescorer differs from the data used to train the base Vicuna LLMs. Unsurprisingly, we fail to see a large improvement in block efficiency for specialized tasks such as math reasoning (GSM8K), but significant gains for tasks where generic English grammaticality is important (QA and summarization). The neural rescorer may also benefit from access to increased context.

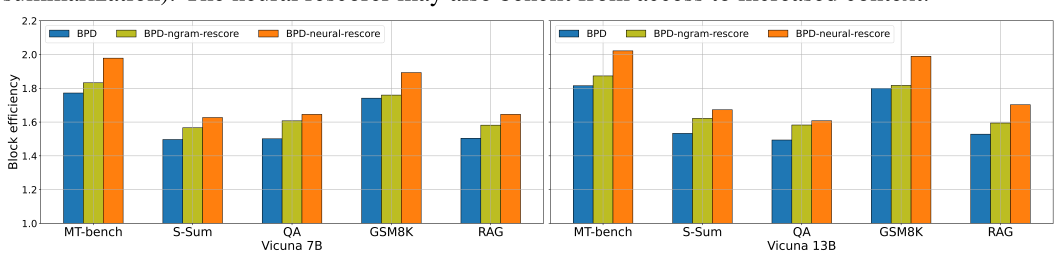


Figure 8: Block efficiency for greedy BPD with n-gram top-10 lattice rescoring. An interpolation weight of 0.2 was placed on the n-gram LM before interpolating with blockwise parallel logits.

8 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. Two algorithms are proposed for generating higher quality drafts: one for local rescoring with small neural models (i.e., neural BPD) and another for global rescoring with an n-gram LM and generating multiple drafts (i.e., p-best n-gram BPD). These algorithms leverage the strengths of both blockwise parallel LMs and small rescoring models to reduce average decoding latency, pushing the boundaries of efficient text generation with BPD. We show that BPD lattice rescoring even complements Medusa decoding, a recent extension of BPD, demonstrating further latency reduction for open-source LLMs. We believe this work points to the value in incorporating smaller LMs in improving LLM decoding speed.

Acknowledgments

We would like to thank Cyril Allauzen for discussions on finite state lattice rescoring throughout the development of this work.

References

[1] Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. 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*, pages 11–23. Springer, 2007.

[2] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[4] Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*, 2024.

[5] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

[6] Ethan A Chi, Julian Salazar, and Katrin Kirchhoff. Align-refine: Non-autoregressive speech recognition via iterative realignment. *arXiv preprint arXiv:2010.14233*, 2020.

[7] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[8] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[9] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[10] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35:30318–30332, 2022.

[11] Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. *arXiv preprint arXiv:1910.10073*, 2019.

[12] Alexander R Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir R Radev. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. *arXiv preprint arXiv:1906.01749*, 2019.

[13] Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of llm inference using lookahead decoding. *arXiv preprint arXiv:2402.02057*, 2024.

[14] Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. Samsum corpus: A human-annotated dialogue dataset for abstractive summarization. *arXiv preprint arXiv:1911.12237*, 2019.

[15] Max Grusky, Mor Naaman, and Yoav Artzi. Newsroom: A dataset of 1.3 million summaries with diverse extractive strategies. In Marilyn Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 708–719, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[16] Jiatao Gu, James Bradbury, Caiming Xiong, Victor OK Li, and Richard Socher. Non-autoregressive neural machine translation. *arXiv preprint arXiv:1711.02281*, 2017.

[17] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. *Advances in neural information processing systems*, 28, 2015.

[18] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. An empirical analysis of compute-optimal large language model training. *Advances in Neural Information Processing Systems*, 35:30016–30030, 2022.

[19] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

[20] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

[21] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*, 2020.

[22] Slava Katz. 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.

[23] Sehoon Kim, Karttikeya Mangalam, Suhong Moon, Jitendra Malik, Michael W Mahoney, Amir Gholami, and Kurt Keutzer. Speculative decoding with big little decoder. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[24] Taehyeon Kim, Joonkee Kim, Gihun Lee, and Se-Young Yun. Instructive decoding: Instruction-tuned large language models are self-refiner from noisy instructions. In *The Twelfth International Conference on Learning Representations*, 2024.

[25] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[26] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the association for computational linguistics companion volume proceedings of the demo and poster sessions*, pages 177–180. Association for Computational Linguistics, 2007.

[27] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

[28] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023.

[29] Xiang Lisa Li, Ari Holtzman, Daniel Fried, Percy Liang, Jason Eisner, Tatsunori Hashimoto, Luke Zettlemoyer, and Mike Lewis. Contrastive decoding: Open-ended text generation as optimization. *arXiv preprint arXiv:2210.15097*, 2022.

[30] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. *arXiv preprint arXiv:2305.11627*, 2023.

[31] Giovanni Monea, Armand Joulin, and Edouard Grave. Pass: Parallel speculative sampling. *arXiv preprint arXiv:2311.13581*, 2023.

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

[33] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

[34] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[35] Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

[36] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv e-prints*, 2019.

[37] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[38] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[39] Apoorv Saxena. Prompt lookup decoding, November 2023.

[40] Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472, 2022.

[41] Wade Shen, Richard Zens, Nicola Bertoldi, and Marcello Federico. The jhu workshop 2006 iwslt system. In *Proceedings of the Third International Workshop on Spoken Language Translation: Evaluation Campaign*, 2006.

[42] Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. *Advances in Neural Information Processing Systems*, 31, 2018.

[43] Andreas Stolcke. Entropy-based pruning of backoff language models. *arXiv preprint cs/0006025*, 2000.

[44] Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

[45] Ziteng Sun, Jae Hun Ro, Ahmad Beirami, and Ananda Theertha Suresh. Optimal block-level draft verification for accelerating speculative decoding. *arXiv preprint arXiv:2403.10444*, 2024.

[46] Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix Yu. Spectr: Fast speculative decoding via optimal transport. *arXiv preprint arXiv:2310.15141*, 2023.

[47] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[48] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

[49] Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. *arXiv preprint arXiv:2401.07851*, 2024.

[50] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR, 2023.

[51] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems*, 35:27168–27183, 2022.

[52] Euiin Yi, Taehyeon Kim, Hongseok Jeung, Du-Seong Chang, and Se-Young Yun. Towards fast multilingual llm inference: Speculative decoding and specialized drafters, 2024.

[53] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36, 2024.

[54] Chen Zhu, Wei Ping, Chaowei Xiao, Mohammad Shoeybi, Tom Goldstein, Anima Anandkumar, and Bryan Catanzaro. Long-short transformer: Efficient transformers for language and vision. *Advances in neural information processing systems*, 34:17723–17736, 2021.

A Broader impact

Our work on BPD for language models has potential applications in latency-sensitive scenarios. Furthermore, this work suggests that LLMs may benefit from the incorporation of faster, lightweight language models, either to reduce latency or potentially to improve the quality of generated text.

B Limitation and future work

B.1 Limitation

Our current drafting heads closely follow the original design of BPD but leave room for architectural improvements. The structure of the block drafter is essential for optimizing gains from rescoring, and advanced training methods may enable the model to understand the block context effectively, bringing better alignment into the target prediction.

B.2 Future work

Our work proposes to augment a small model to improve the quality of the drafts. Possible future directions include (a) combining our lattice rescoring method with alternative sampling strategies (b) scaling the blockwise parallel LM for compatibility with larger-scale LLMs (c) improving training methods for drafting heads (d) using the sequential entropy ordering of heads (Figure 2b) 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 [50, 51, 10] and model pruning [44, 30]. In parallel, neural architecture search has played a crucial role in identifying network structures that balance performance with efficiency [25, 54]. Relatedly, Elbayad et al. [11] propose early-exiting at intermediate layers for faster inference, while Schuster et al. [40] 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 [42, 16] 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 [6], iterative refinement [23], and tree-based attention mechanism [4]. However, these refinements often require complex training or additional inference data.

C.2 Efficient autoregressive 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 [28, 5, 23, 45, 52]. 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 [29, 24]. 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 Experiment details

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 8: 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 8. 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.

D.3 *n*-gram details

All *n*-gram LMs in this work are Katz backoff *n*-gram LMs [22] fit on the train split of the GPT3 subword-tokenized English C4 corpus with *n*-gram order $\in \{2, 4\}$. We apply entropy pruning [43] 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 predicting the last word of a passage based on the full context [33].
- **SQuAD V1 (Stanford Question Answering Dataset)**: A reading comprehension dataset that features questions based on Wikipedia articles, with answers located within the text [38].
- **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 [17].
- **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 [14].
- **MultiNews**: Comprises news articles from diverse sources for abstractive summarization tasks, evaluating models on handling different writing styles and topics [12].
- **XSUM**: Contains scientific documents and summaries, challenging language models to process complex scientific information and language [32].
- **NewsRoom**: A dataset of news articles aimed at assessing the factual accuracy and information extraction capabilities of models in generating summaries [15].

All datasets were tokenized using the 50,257 GPT3 subword vocabulary [3].

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]"

E Rescoring with in-domain language models

Table 9: 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 9 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. 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.

F 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 10). This is a signal that the rescorer does not provide additional information over the original block draft heads.

Table 10: 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

G Local rescoring impact on block efficiency

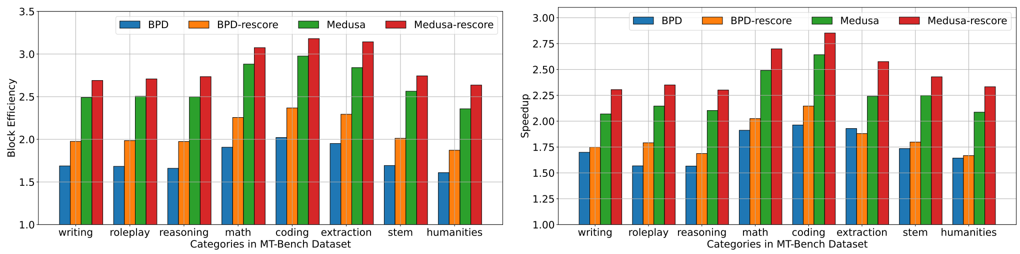
Table 11 reveals the impact of different rescoring methods on the block efficiency of the block lattice, offering insights into their effectiveness across diverse tasks and models, supporting the investigations in **Section 6.1**.

- **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 tasks where standard BPD already achieves significant speed improvements, there is limited room for further gains through rescoring.
- **Efficacy for poor baselines**: In tasks with lower initial block efficiency (SQuAD V1, XSUM, NewsRoom), rescoring using both *n*-gram and neural language models results in increased 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 largest gains, suggesting that the effectiveness of rescoring is 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.

Table 11: 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	**<u>3.12</u>**	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%) 🔴	**<u>2.10 (+0.96%)</u>** 🟢	**<u>2.10 (+0.96%)</u>** 🟢	2.07 (-0.48%) 🔴	2.16	2.45
S-SUM	CNN/Daily	**<u>1.74</u>**	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%) 🟢	**<u>1.39 (+9.45%)</u>** 🟢	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%) 🟢	**<u>1.25 (+13.64%)</u>** 🟢	**<u>1.25 (+13.64%)</u>** 🟢	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%) 🟢	**<u>1.23 (+8.85%)</u>** 🟢	1.17 (+3.54%) 🟢	1.17	1.55
	NewsRoom	1.08	1.20 (+11.11%) 🟢	1.18 (+9.26%) 🟢	1.18 (+9.26%) 🟢	**<u>1.29 (+19.44%)</u>** 🟢	**<u>1.29 (+19.44%)</u>** 🟢	1.17 (+8.33%) 🟢	1.15	1.50



(a) Block efficiency (b) Speedup ratio

Figure 9: Block efficiency and speedup ratio relative to the standard autoregressive decoding on sub-categories of MT-Bench dataset [53] when greedily decoding with Vicuna 7B.

For the evaluation of inference time on open-sourced Vicuan 7B model, we provide Figure 9 for the block efficiency and speedup ratio on MT-Bench dataset which is parallel to Figure 7 in Section 7.

H Ablation on the number of heads in the blockwise parallel LM

Table 12 summarizes the block efficiency for different head configurations across various language tasks with the same settings discussed in Figure 1.

- **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 12: 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

I Practical 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) [9]. 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.

I.1 Hardware utilization

We compare our approach against traditional Autoregressive LMs across several metrics (Table 13).

Table 13: 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 with 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.

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.

I.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 ≈100M n-grams. The average latency observed across 10 runs for different numbers of the shortest paths is summarized in Table 14.

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

Table 14: 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

initial experiments, increasing the size of the n-gram LM had little effect on n-best rescoring latency, indicating that improvements to rescoring LM quality will incur little additional latency, provided that the rescoring 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 15.

Table 15: 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

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

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims are provided in Section 1 and Section 2.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The limitations are described in Appendix B.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

Justification: The paper does not include theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Appendix D describes the experimental details of both the blockwise parallel LMs used in this paper and proposed rescoring methods.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: We have not provided open access to the source code used in our experiments. However, we have detailed the data access, architecture, and training processes in the supplemental material to enable replication of our results.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Appendix D provides clear experimental details.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Our paper does not report error bars or statistical tests, as it focuses on qualitative evaluations and proof-of-concept demonstrations. We provide detailed descriptions of our experiments to support the claims and emphasize practical observations.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Compute resources are detailed in Section 4 and Section 6.1, including TPU types (TPUv3/TPUv4), training time, batch sizes and use of JAX.

   Guidelines

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: Our research does not involve human subjects and primarily presents no direct ethical concerns. The datasets with CC-BY 4.0 are used for evaluation. We discuss potential societal impact in Appendix A.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss societal impacts of our research in Appendix A.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: We believe our models do not present more risk than the public models we compared them to, as they show similar effectiveness but at a lower cost of operation.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The datasets having CC-BY 4.0 license are used for training and evaluation.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: This paper does not release new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: This paper presents no potential risks for IRB approvals or equivalent for research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.