Accelerating Blockwise Parallel Language Models with Draft Refinement

Taehyeon Kim[1] , Ananda Theertha Suresh[2], Kishore A Papineni[2],
Michael Riley[2], Sanjiv Kumar[2], Adrian Benton[2]
[1] KAIST AI (Work done while at Google Research) [2] Google Research

NeurIPS 2024

KAIST

Google Research

The Gist

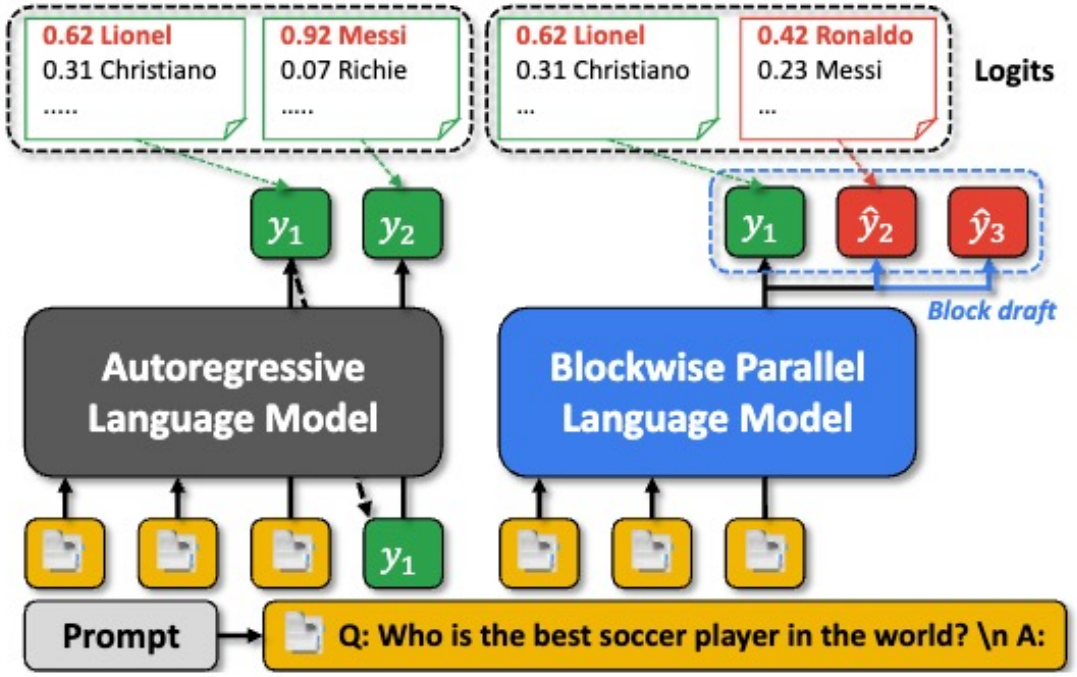
- Decode faster from autoregressive LLMs: **2X – 3X.**
- Lossless quality of generation: no changes on the target model.
- Integrates with Medusa[1] for faster results.

1. Cai, Tianle, et al. "Medusa: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads." *Forty-first International Conference on Machine Learning*
   - *Here, neural rescoring firstly refines the block draft, and then tree-attention is applied over the refined block draft.*

What is Blockwise Parallel Language Model…

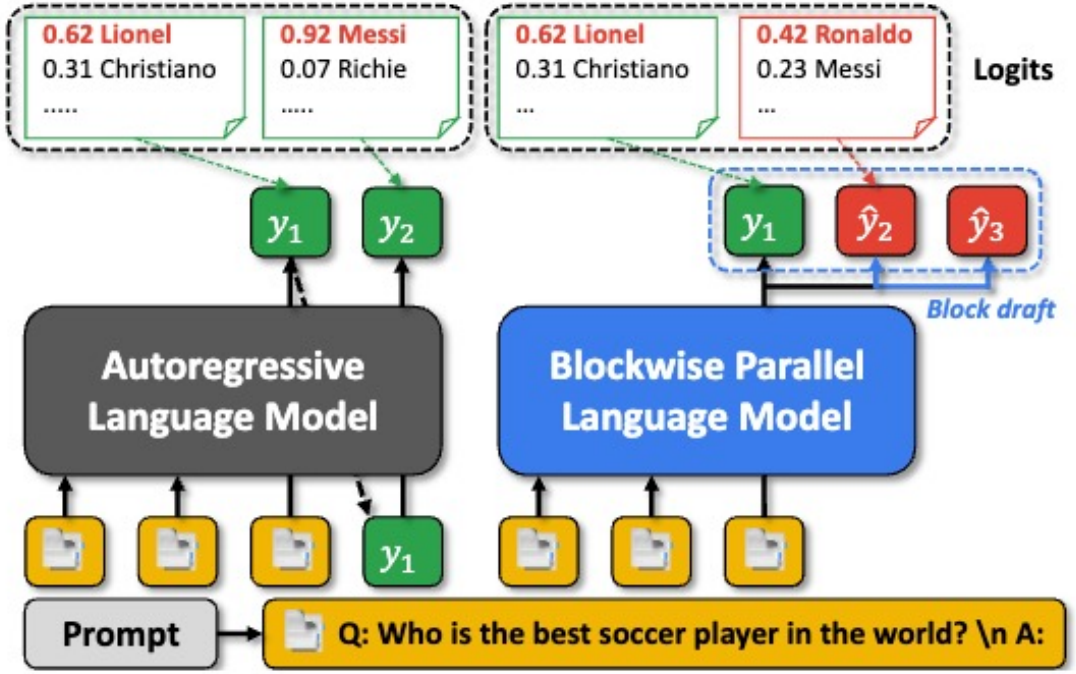
- Blockwise parallel LM
  - Decode K tokens (block) in non-autoregressive manner with a set of prediction heads.
  - Use Blockwise Parallel Decoding (BPD)
    - Acceleration of text generation.
    - Ancestor of speculative decoding.

    (1) Block draft (2) Verify (3) Accept
  - The more coherent block, the higher acceptance rate. Thus, faster generation.



For Blockwise Parallel Language Models...

- Memory-bound scenario.
- Can we somehow gain more from the block?



Observation

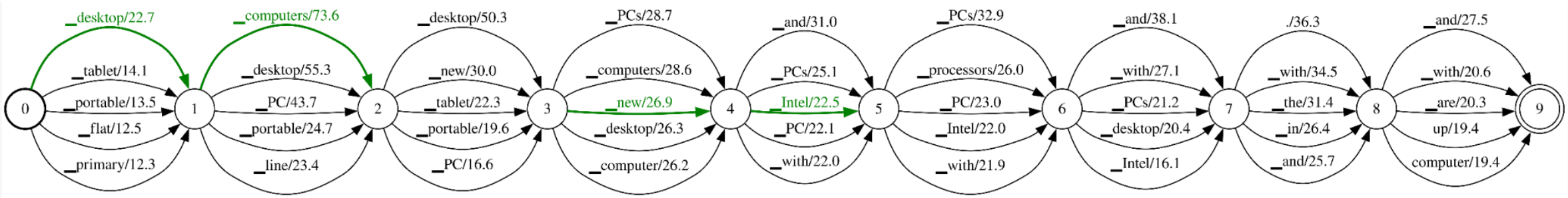
- Some tokens in the block are (1) repetitive or (2) incoherent (e.g., Lionel Ronaldo).

- A blockwise parallel LM produces drafts where 20.0-75.5% of consecutive tokens are repeated.

Observation

- Thought
  1. Represent the top-k tokens from the block drafter as a lattice.
  2. Rescore the lattice using an autoregressive models to improve the draft.

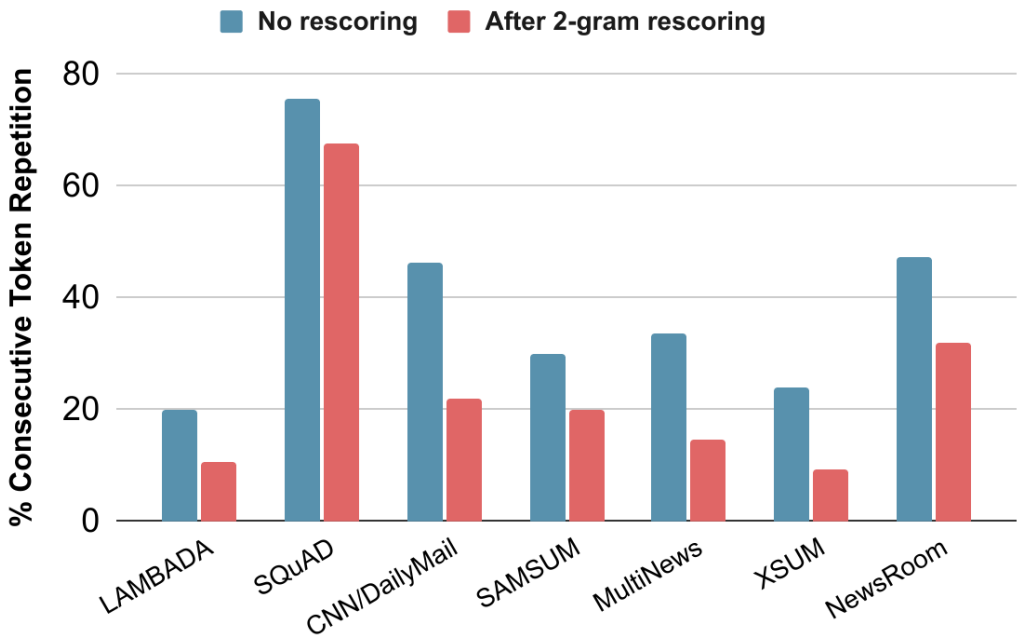
# of prediction heads



top k tokens

Observation

- Thought
  1. Represent the top-k tokens from the block drafter as a lattice.
  2. Rescore the lattice using an autoregressive models to improve the draft.

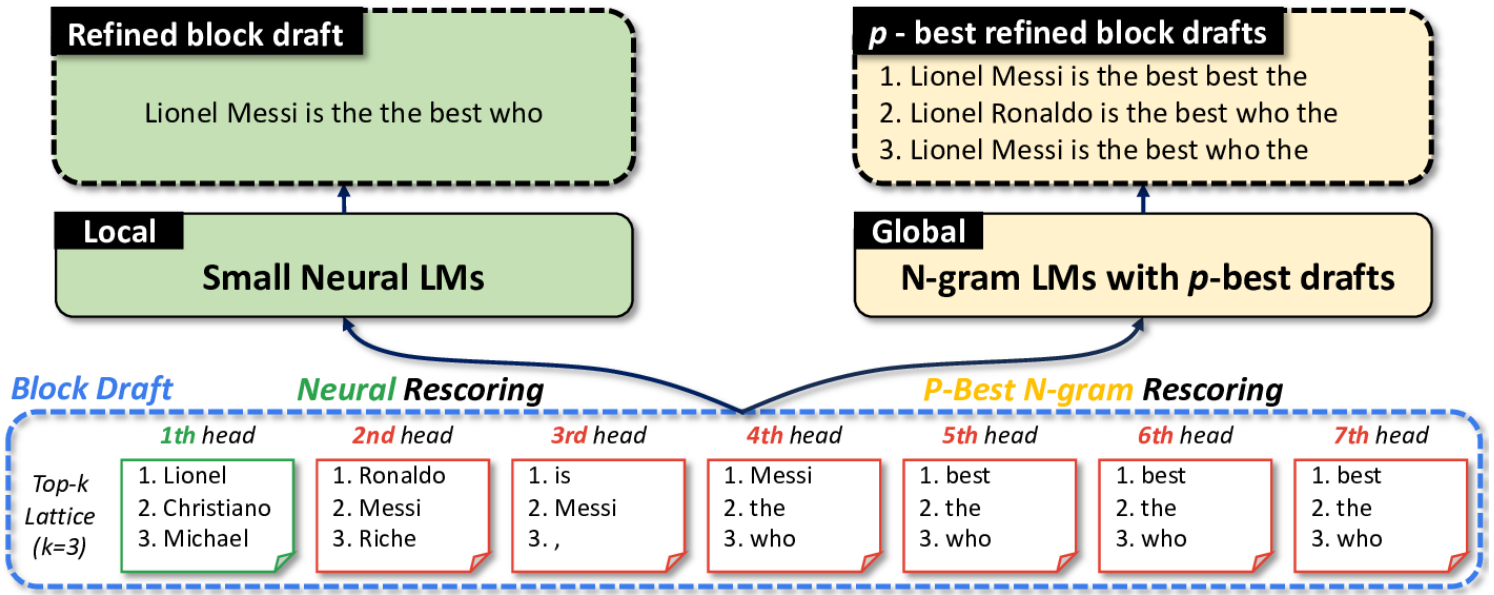


Contribution

- Use of small language models for draft refinement.
- Accelerating BPD with refined block by increasing acceptance rate.

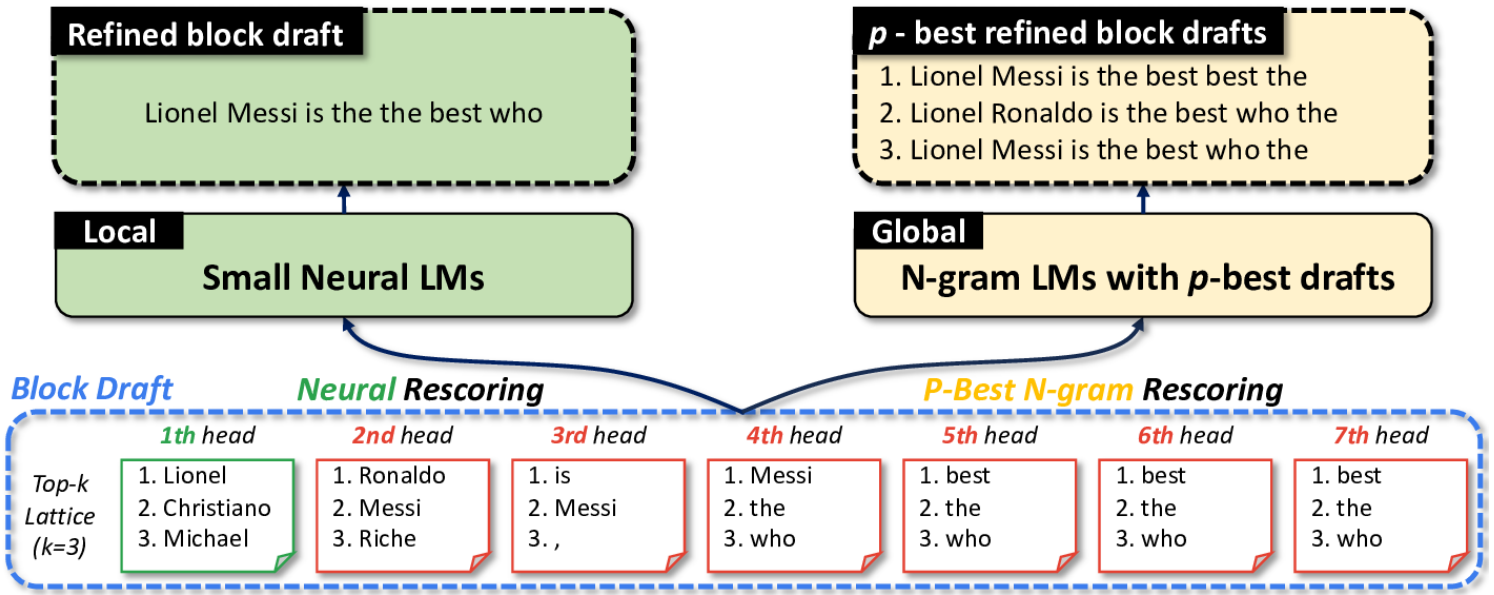
Rescoring Methods

- (Local) Neural Rescoring
  - Greedily rescore the lattice with (61M parameter) decoder-only transformer
  - Pros: Ensure local fluency / Cons: Latency scales with the number of draft tokens.



Rescoring Methods

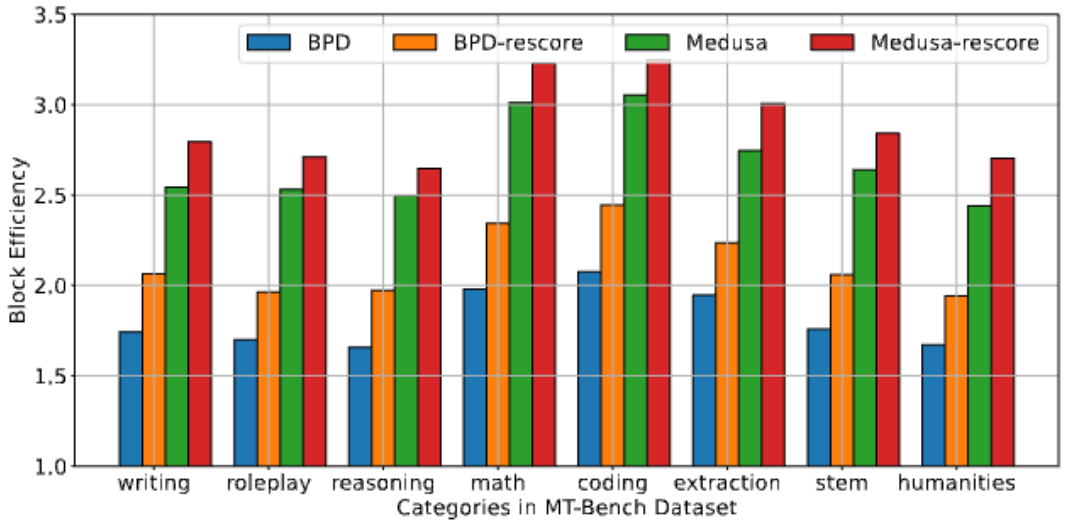
- (Global) N-gram Rescoring
  - Greedily rescore the lattice with C4-trained n-gram model (~100M subword n-grams)
  - Pros: Extract and rescore multiple drafts quickly / Cons: Model itself is weak.



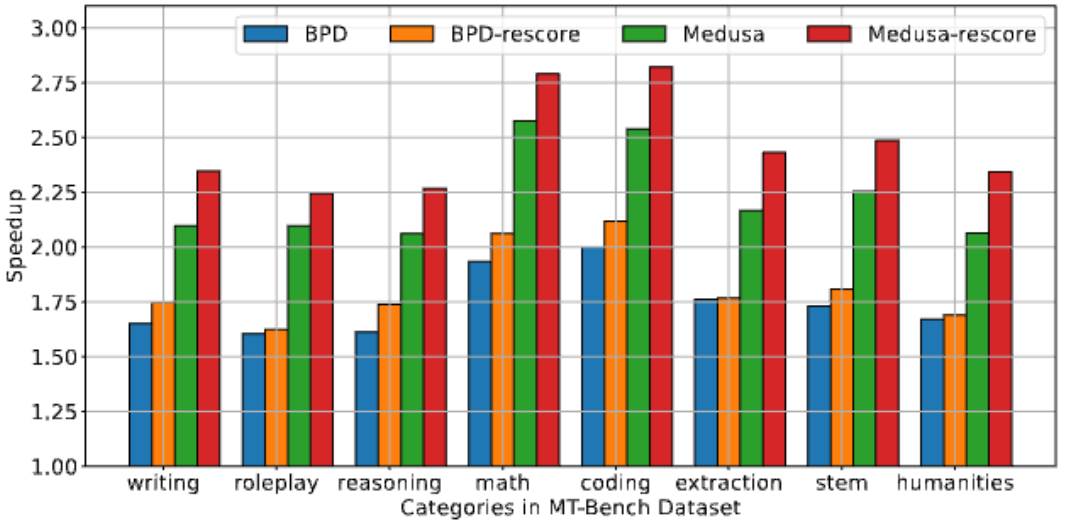
- Block efficiency represents the average number of tokens decoded per serial call.
- Simple rescoring works well on 1.5B LM, but depends on the task.

Task	Dataset	Baseline BPD	Local rescoring neural-61M BPD	4-gram BPD	Global rescoring 16-best 0-gram BPD	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

- [Open LLM 13B] Speedup ratio relative to the standard autoregressive decoding on MT-Bench dataset when greedily decoding with Vicuna 13B.
- Simple neural rescoring further improves Medusa[1] as well as BPD on 13B LLM.



(a) Block efficiency



(b) Speedup ratio

1. Cai, Tianle, et al. "Medusa: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads." *Forty-first International Conference on Machine Learning*
   - *Here, neural rescoring firstly refines the block draft, and then tree-attention is applied over the refined block draft.*

Thank you!

Taehyeon Kim[1] , Ananda Theertha Suresh[2], Kishore A Papineni[2],
Michael Riley[2], Sanjiv Kumar[2], Adrian Benton[2]
[1] KAIST AI (Work done while at Google Research) [2] Google Research

kimtaehyeon610@gmail.com