A Streaming Approach For Efficient Batched Beam Search

Kevin Yang, Violet Yao, John DeNero, Dan Klein

UC Berkeley

{yangk,violetyao,denero,klein}@berkeley.edu

Abstract

We propose an efficient batching strategy for variable-length decoding on GPU architectures. During decoding, when candidates terminate or are pruned according to heuristics, our streaming approach periodically "refills" the batch before proceeding with a selected subset of candidates. We apply our method to variable-width beam search on a state-of-the-art machine translation model. Our method decreases runtime by up to 71% compared to a fixed-width beam search baseline and 17% compared to a variable-width baseline, while matching baselines' BLEU. Finally, experiments show that our method can speed up decoding in other domains, such as semantic and syntactic parsing.

1 Introduction

While inference is often cheap compared to training in modern neural models, one may need to run inference frequently or continually. Such is the case for online machine translation (MT) services: as far back as 2016, Google Translate already translated 100 billion words daily (Turovsky, 2016). Large-scale inference is also required for methods such as iterative backtranslation and knowledge distillation to generate training data (Hoang et al., 2018; Kim and Rush, 2016). For such high-throughput applications, it is useful to decrease inference cost.

Meanwhile, we must preserve accuracy: beam search is slower than greedy decoding, but is nevertheless often preferred in MT. Not only is beam search usually more accurate than greedy search, but it also outputs a *diverse set* of decodings, enabling reranking approaches to further improve accuracy (Yee et al., 2019; Ng et al., 2019; Charniak and Johnson, 2005; Ge and Mooney, 2006).

However, it is challenging to optimize the performance of beam search for modern neural architectures. Unlike classical methods in sparse computation settings, modern neural methods typically operate in dense (batched) settings to leverage specialized hardware such as GPUs.

In this work, we propose a streaming method to optimize GPU-batched variable-output-length decoding. Our method does not use a fixed batch during inference; instead, it continually "refills" the batch after it finishes translating some fraction of the current batch. Our method then continues decoding on the remaining candidates in the batch, prioritizing those least expanded.

We apply our method to variable-width beam search. For variable-output-length decoding even in batched settings, variable-width beam search often modestly decreases accuracy in exchange for substantial speedups over fixed-width beam search (Freitag and Al-Onaizan, 2017; Wu et al., 2016). When decoding with Fairseq's state-of-the-art WMT'19 model (Ng et al., 2019), our method further improves over the speed of baseline variable-width beam search: up to 16.5% on a 32GB V100 GPU, without changing BLEU (Papineni et al., 2002). Our approach also improves decoding efficiency in lightweight models for semantic and syntactic parsing.[1] In principle, our method can be applied to any task which sequentially processes variable-length data.

2 Background: Beam Search

Given encoder $\mathcal{E}$ and decoder $\mathcal{D}$, our task is to convert inputs $\{x_1 \dots x_N\}$ into corresponding outputs $\{\bar{y}_1 \dots \bar{y}_N\}$, for data size N. For example, in machine translation, each x_i is a source sentence consisting of a sequence of tokens and each $\bar{y}_i$ is a translation. We assume $\mathcal{D}(e_i, y_i)$ receives $e_i = \mathcal{E}(x_i)$ and a partial y_i as input, constructing $\bar{y}_i$ one token at a time.

One method of constructing $\bar{y}_i$ for a given x_i is greedy search. Let $y_i^{l_t}$ be the in-construction *candidate* with length $l_t = t$ at timestep t. We initialize

[1] Code available at https://github.com/yangkevin2/emnlp2020-stream-beam-mt.

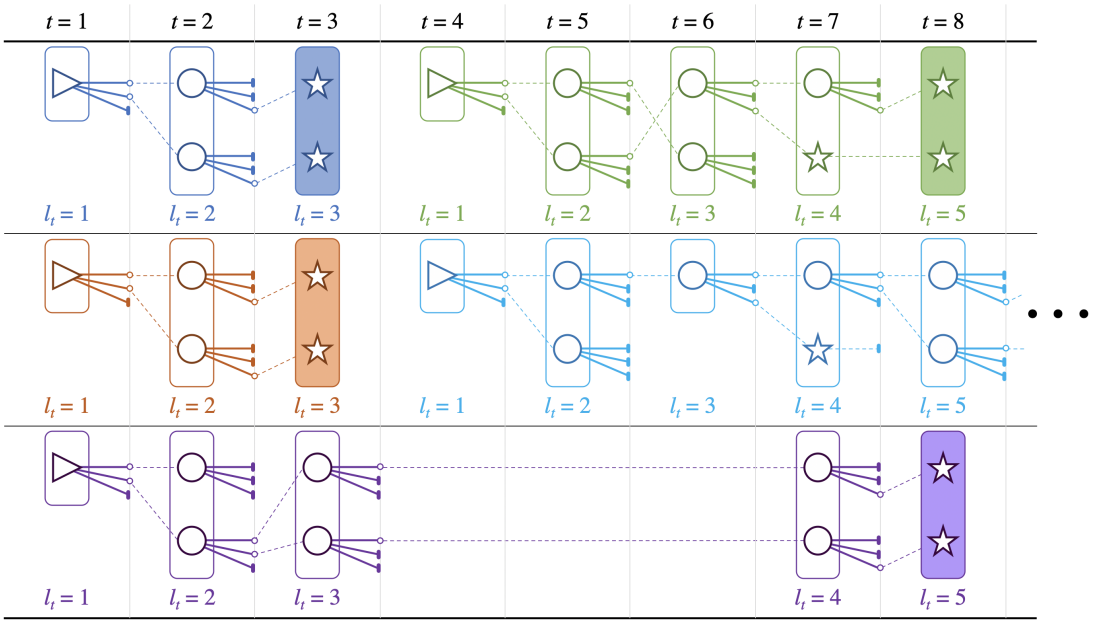

Figure 1: Illustration of our method VAR-STREAM for variable-width beam search with vocabulary size $|\mathcal{V}| = 3$, beam width $k = 2$, batch size $n = 3$, refill threshold $\epsilon = \frac{1}{3}$. Each color corresponds to the beam for a single input. The rounded rectangles at each timestep are beams $\mathcal{H}(B_i^{l_t})$, while the shapes inside are individual candidates. Shaded beams represent the end of the search. The right-facing triangles indicate the initial candidate containing just the start token w_{sos}, circles denote an active (non-terminated) candidate, and stars denote a finalized candidate. Candidates become finalized after following the third (bottom-most) branch in an expansion, corresponding to the end token w_{eos}; they then undergo only no-op expansions thereafter. The first two rows of beams depict normal operation of variable-width beam search, including heuristic pruning in the light blue beam at $t = 6$. The third row shows an important detail of our method: VAR-STREAM refills the batch after $t = 3$, when only ϵn beams remain, and the remaining purple beam halts computation until the two newly added beams reach the same l_t. (This detail matters in transformer architectures; see Appendix A.2.)

y_i^1 as the start token w_{sos}, and at each timestep t obtain $y_i^{l_t+1}$ by concatenating the maximum-probability token. We *finalize* $y_i^{l_t}$ as $\bar{y}_i$ once we append the end token, or at some maximum length.

Previous work has found that greedy search often underperforms beam search in accuracy, and gains from non-greedy decoding have also been observed in many classical models (Sutskever et al., 2014; Freitag and Al-Onaizan, 2017; Wilt et al., 2010). See the dark blue, green, and brown beams in Figure 1 for normal operation of fixed-width beam search using beam width $k = 2$. For each input x_i, fixed-width beam search tracks a length-l_t, width-k *beam* $B_i^{l_t}$ for each time t. $B_i^{l_t}$ contains k length-l_t candidates $y_{i1}^{l_t} \dots y_{ik}^{l_t}$ with maximum log-likelihood in order, denoted by the shapes inside the rounded rectangles (beams) in the figure. At each step, beam search considers all $k|\mathcal{V}|$ possible *candidate expansions* (one-token extensions of existing candidates), where $\mathcal{V}$ is the vocabulary. The top k expansions become the expanded beam $B_i^{l_t+1}$. Figure 1 shows these expansions at each timestep for $|\mathcal{V}| = 3$, with active non-terminated candidates (circles) becoming finalized (stars) after following the bottom-most branch, corresponding to the end token w_{eos}. In the end, beam search yields k finalized candidates $\bar{y}_{i1} \dots \bar{y}_{ik}$ compared to a single $\bar{y}_i$ in greedy search.

Variable-width beam search reduces the computational cost of the above *fixed-width* beam search by pruning the full beam $B_i^{l_t}$ using heuristics $\mathcal{H}$, for example at $t = 6$ for the light blue beam in the figure. The width of the resulting pruned beam $\mathcal{H}(B_i^{l_t})$ is no longer always exactly equal to k, and may vary over time.

3 Streaming Variable-Length Decoding

As an example, consider translating a batch of n German sentences into English via a traditionally-batched variable-width beam search (e.g., Freitag and Al-Onaizan (2017)) on a GPU. Henceforth we refer to this baseline as VAR-BATCH.

An inefficiency results from decoding being inherently variable in length: After t steps, we may

have completed $m < n$ translations, but the last $n - m$ beams may take several more timesteps. For example, in Figure 1, our initial batch consists of the dark blue, brown, and purple beams. After the dark blue and brown beams terminate, we would still be stuck decoding the purple beam by itself.

The resulting GPU underutilization motivates our streaming approach VAR-STREAM applied to variable-width beam search (Figure 1). For batch size n, VAR-STREAM initially proceeds identically to VAR-BATCH. But when the number of remaining beams drops below ϵn for some constant $\epsilon \in (0, 1)$, VAR-STREAM encodes a new batch of inputs x to "refill" its batch to size n.[2] This occurs at $t = 4$ in Figure 1, where we refill the batch using the green and light blue beams.

Note the active beams are no longer of equal length $l_t = t$ for every beam after refilling. At each subsequent t, VAR-STREAM only expands beams $\mathcal{H}(B_i^{l_t})$ with minimal l_t; in particular, the purple beam in Figure 1 pauses computation at $t = 4$.[3] When decoding with state-of-the-art transformer architectures for MT, it is advantageous to expand only beams with minimal l_t at each step, because self-attention causes steps at higher l_t to be more expensive; see Appendix A.2. (For RNN-based architectures, it may be faster to expand all active beams at each step.)

We emphasize that VAR-STREAM is an implementation optimization, exactly matching the output of VAR-BATCH. Full details in Algorithm 1.

When the memory bottleneck is partially the decoding process itself rather than caching the input encodings $\mathcal{E}(x_i)$ or beams $\mathcal{H}(B_i^{l_t})$, VAR-STREAM can cache additional encodings and beams on GPU. At each t, VAR-STREAM then selects beams up to some limit on total beam width, filling GPU capacity even in the case of variable-width beams. This batching constraint addresses a second inefficiency in GPU utilization: the widths of the pruned beams $\mathcal{H}(B_i^{l_t})$ may vary over time. We exploit this in semantic (Sec. 4.2) and syntactic parsing (Sec. 4.3).

Algorithm 1 VAR-STREAM

Input: inputs $X = \{x_1, \ldots x_N\}$, model $(\mathcal{E}, \mathcal{D})$, batch size n, refill threshold ϵ, beam width k, pruning heuristics $\mathcal{H}$

procedure DECODE($X, \mathcal{E}, \mathcal{D}, n, \epsilon, k, \mathcal{H}$)
  # initialize encodings, beams, final outputs
  $E, \mathcal{B}, Y = [\,], [\,], [\,]$
  # initialize index counter
  $c = 1$
  while $c \leq N$ or $|\mathcal{B}| > 0$ **do**
    if $c \leq N$ and $|\mathcal{B}| \leq \epsilon n$ **then**
      # refill batch by $m = n(1 - \epsilon)$
      $E = E + [\mathcal{E}(x_c) \ldots \mathcal{E}(x_{c+m-1})]$
      $\mathcal{B} = \mathcal{B} + [w_{sos}] \times m$
      $Y = Y + [\,] \times m$
      $c = c + m$
    Select $E_s \in E, \mathcal{B}_s \in \mathcal{B}$ with min l_t
    for $(e_i, \mathcal{H}(B_i^{l_t})) \in (E_s, \mathcal{B}_s)$ **do**
      for $y_{ij}^{l_t} \in \mathcal{H}(B_i^{l_t})$ **do**
        Compute expansion $\mathcal{D}(e_i, y_{ij}^{l_t})$
      Update $\mathcal{H}(B_i^{l_t})$ to $\mathcal{H}(B_i^{l_t+1})$
      Add finalized candidates to $Y[i]$
    Remove terminated beams from $E, \mathcal{B}$
  return Y

4 Experiments

We apply VAR-STREAM to variable-width beam search in machine translation, semantic parsing, and syntactic parsing. We use the absolute threshold and max candidates heuristics of Freitag and Al-Onaizan (2017) as $\mathcal{H}$, modifying only the heuristic hyperparameters for each domain based on a development set. The absolute threshold heuristic prunes candidates $y_{ij}^{l_t}$ whose log-probabilities fall short of the best candidate $y_{i1}^{l_t}$'s by some threshold δ, i.e. $\log P(y_{ij}^{l_t}) < \log P(y_{i1}^{l_t}) - \delta$. The max candidates heuristic prevents the search from selecting more than $M < k$ length-$l_t + 1$ candidates originating from the same length-l_t candidate at each step t.

In each domain we compare four methods:

1. GREEDY, a greedy search,
2. FIXED, a fixed-width beam search,
3. VAR-BATCH, a batched variable-width beam search, and
4. VAR-STREAM, our streaming method.

We sort and bucket inputs by length for batching.

4.1 Machine Translation

We evaluate on the transformer architecture implemented in Fairseq (Ott et al., 2019), which scored highest on several tracks of WMT'19 (Barrault

[2] Our method is relatively insensitive to ϵ (Appendix A.3).

[3] As very long translations could get "stuck" in the batch, one can periodically finish computation on all remaining beams in the batch if latency is a concern in addition to throughput.

	Method	BLEU	Wall Clock (s)
De-En	GREEDY	48.18	39.01 ± 0.14
k=50	FIXED	49.57	891.53 ± 0.77
32GB	VAR-BATCH	49.59*	308.15 ± 3.48
	VAR-STREAM	49.59*	257.20 ± 0.54
	Method	**BLEU**	**Wall Clock (s)**
Ru-En	GREEDY	37.84	42.09 ± 1.33
k=50	FIXED	38.98	893.23 ± 1.23
32GB	VAR-BATCH	39.04*	399.98 ± 1.25
	VAR-STREAM	39.04*	342.18 ± 1.28
	Method	**BLEU**	**Wall Clock (s)**
De-En	GREEDY	48.18	46.41 ± 0.23
k=5	FIXED	49.42	102.25 ± 0.58
32GB	VAR-BATCH	49.46*	114.18 ± 0.17
	VAR-STREAM	49.46*	92.39 ± 1.05
	Method	**BLEU**	**Wall Clock (s)**
Ru-En	GREEDY	37.84	42.09 ± 1.33
k=5	FIXED	38.83	103.59 ± 0.34
32GB	VAR-BATCH	39.03*	130.01 ± 2.33
	VAR-STREAM	39.03*	95.45 ± 0.21
	Method	**BLEU**	**Wall Clock (s)**
De-En	GREEDY	48.18	46.41 ± 0.23
k=50	FIXED	49.57	2072.86 ± 23.18
16GB	VAR-BATCH	49.59*	645.70 ± 17.49
	VAR-STREAM	49.59*	606.17 ± 4.96
	Method	**BLEU**	**Wall Clock (s)**
Ru-En	GREEDY	37.84	52.26 ± 0.50
k=50	FIXED	38.98	2155.95 ± 58.47
16GB	VAR-BATCH	39.04*	852.93 ± 9.11
	VAR-STREAM	39.04*	803.72 ± 15.94

Table 1: Top-1 BLEU and wall clock times for machine translation. Our method VAR-STREAM is substantially faster than VAR-BATCH (14-17% for $k = 50$ on 32GB, 19-27% for $k = 5$ on 32GB, 6% for $k = 50$ on 16GB) and FIXED (62-71% for $k = 50$, 8-10% for $k = 5$), while preserving high BLEU. *Our rules for finalizing candidates during decoding differ slightly from Fairseq, resulting in equal or higher BLEU for VAR-BATCH and VAR-STREAM compared to FIXED. Adapting our implementation to fixed-width beam search is slower but yields higher BLEU (Appendix A.1).[4]

et al., 2019). For our main experiments, we run German-English and Russian-English translation on newstest2018 using an ensemble of 5 models with $k = 50$, matching the setup of Ng et al. (2019) but without reranking. As smaller beam sizes are also common in practice, we evaluate with $k = 5$ as well. Our GREEDY and FIXED baselines are Fairseq's implementation, while VAR-BATCH and VAR-STREAM are our own. For all methods, we evaluate 5 runs on a 32GB V100 GPU. For $k = 50$, we also run on a 16GB V100 GPU, noting that 32GB is likely more realistic in a production setting. We choose batch size to saturate the GPU, using $\epsilon = \frac{1}{6}$ for VAR-STREAM, with pruning heuristics $\delta = 1.5, M = 5$. Appendix A.3 details hyperparameter choices.

As shown in Table 1, on both GPU settings and on both languages, GREEDY is fastest, but suffers heavily in BLEU. Our VAR-STREAM is the fastest beam-based search, and matches the BLEU of the beam search baselines. Compared to VAR-BATCH, VAR-STREAM is faster by 14-17% when using $k = 50$ on the 32GB GPU, and by 19-27% when $k = 5$. VAR-STREAM also remains 6% faster when using $k = 50$ on the 16GB GPU where overhead is higher. VAR-BATCH and VAR-STREAM match the BLEU of FIXED while being 2-3 times faster when using beam size 50, confirming the speedups from VAR-BATCH over FIXED in e.g., Freitag and Al-Onaizan (2017). FIXED is more competitive when $k = 5$ because the potential for heuristic beam pruning is much more limited; moreover, our implementations of VAR-STREAM and VAR-BATCH somewhat understate both speedups and BLEU cost compared to FIXED due to an implementation difference with Fairseq (Appendix A.1).[4] Thus VAR-BATCH becomes slower than FIXED when $k = 5$. Nevertheless, VAR-STREAM remains the fastest in this scenario by 8-10%.

4.2 Semantic Parsing

To explore our method's domain applicability, we experiment with semantic parsing using the seq2seq model of Dong and Lapata (2016). This lightweight model is no longer state of the art, but its decoding is representative of more recent architectures (Suhr et al., 2018; Yin and Neubig, 2018; Lin et al., 2019). We use the ATIS flight-booking dataset (Dahl et al., 1994), setting $n = k = \delta = 10, M = 3$. Due to the small dataset and model, our batching constraint is more theoretical: we constrain each method to expand at most

[4]Essentially, we allow finalized candidates to fall off the beam if we find enough other higher-likelihood candidates. See e.g., the star at $t = 7$ in the light blue beam in Figure 1. Fairseq does not allow this.

	Semantic Parsing (ATIS)				*Syntactic Parsing (Penn Treebank)*			
Method	**F1**	**Oracle**	**Time (s)**	**Exp. / Step**	**F1**	**Oracle**	**Time (s)**	**Exp. / Step**
GREEDY	86.4	86.4	1.4 ± 0.0	46.5	91.3	91.3	22.3 ± 0.2	86.7
FIXED	86.6	91.2	7.7 ± 0.1	48.0	91.2	94.0	224.2 ± 2.1	97.1
VAR-BATCH	86.6	90.2	8.2 ± 0.2	16.9	91.2	93.8	235.6 ± 1.8	48.2
VAR-STREAM	86.6	90.2	6.3 ± 0.1	72.1	91.2	93.8	220.5 ± 2.5	95.7

Table 2: Top-1 and oracle reranking F1, wall clock (avg. 5 runs), and average candidate expansions per timestep (i.e., total candidate expansions divided by total decoding timesteps) for semantic parsing on ATIS and syntactic parsing on the Penn Treebank (PTB). Theoretical maximum efficiency under our batching constraint is 100 expansions per step for both tasks. VAR-STREAM achieves substantially higher expansions per step than other methods on ATIS. On PTB, FIXED achieves near-perfect efficiency because all $\bar{y}_{ij}$ for a given x_i have the same length. But comparing variable-width beam searches, VAR-STREAM is much more efficient with batch capacity than VAR-BATCH.

$nk = 100$ candidates per timestep (i.e., total beam width), instead of simply saturating the GPU.[5]

As shown by the expansions per step in Table 2, VAR-STREAM uses the batch capacity of 100 most efficiently. Thus VAR-STREAM is faster than both VAR-BATCH and FIXED, despite overhead which is exacerbated in a small model. The speedup is larger on the JOBS and GEO datasets (Zettlemoyer and Collins, 2012) (Appendix A.4). While all methods achieve similar top-1 F1, oracle F1 (using an oracle to "rerank" all outputs $\bar{y}_{ij}$) highlights the benefit of producing a diverse set of translations.

4.3 Syntactic Parsing

We also experiment with the lightweight shift-reduce constituency parser of Cross and Huang (2016) on the Penn Treebank (Marcus et al., 1993). This task and model differ from our previous setups in that for a given input x_i, all valid parses $\bar{y}_{ij}$ have exactly the same length. When inputs are bucketed by length, this removes the variable-output-length inefficiency for traditional batching: we cannot get stuck finishing a small fraction of beams when the rest of the batch is done. Thus, this task isolates the effect of VAR-STREAM using batch capacity more efficiently in the case of variable-width beams. We use the same computational constraint as in semantic parsing, with $n = k = 10, \delta = 2.5, M = 3$.

As all $\bar{y}_{ij}$ have equal length for a given x_i, FIXED already achieves near-perfect efficiency in expansions per step (Table 2). Combined with the impact of overhead in this older (smaller) model, VAR-STREAM is not substantially faster than FIXED in this setting. However, when comparing variable-width beam searches where efficient batching is more difficult, we observe that VAR-STREAM doubles VAR-BATCH in expansions per step.

[5]Due to caching additional encodings and beams, VAR-STREAM uses more GPU memory in this idealized setting.

5 Discussion

In this work, we have proposed a streaming method for variable-length decoding to improve GPU utilization, resulting in cheaper inference. Applied to a state-of-the-art machine translation model, our method yields substantial speed improvements compared to traditionally-batched variable-width beam search. We also apply our method to both semantic and syntactic parsing, demonstrating our method's broader applicability to tasks that process variable-output-length data in a sequential manner.

Acknowledgments

We thank Steven Cao, Daniel Fried, Nikita Kitaev, Kevin Lin, Mitchell Stern, Kyle Swanson, Ruiqi Zhong, and the three anonymous reviewers for their helpful comments and feedback, which helped us to greatly improve the paper. This work was supported by Berkeley AI Research, DARPA through the Learning with Less Labeling (LwLL) grant, and the NSF through a fellowship to the first author.

References

Loïc Barrault, Ondřej Bojar, Marta R Costa-jussà, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Matthias Huck, Philipp Koehn, Shervin Malmasi, et al. 2019. Findings of the 2019 conference on machine translation (wmt19). In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 1–61.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 173–180. Association for Computational Linguistics.

James Cross and Liang Huang. 2016. Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles. *arXiv preprint arXiv:1612.06475*.

Deborah A Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. 1994. Expanding the scope of the atis task: The atis-3 corpus. In *Proceedings of the workshop on Human Language Technology*, pages 43–48. Association for Computational Linguistics.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. *arXiv preprint arXiv:1601.01280*.

Markus Freitag and Yaser Al-Onaizan. 2017. Beam search strategies for neural machine translation. *arXiv preprint arXiv:1702.01806*.

Ruifang Ge and Raymond Mooney. 2006. Discriminative reranking for semantic parsing. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 263–270. Association for Computational Linguistics.

Vu Cong Duy Hoang, Philipp Koehn, Gholamreza Haffari, and Trevor Cohn. 2018. Iterative back-translation for neural machine translation. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 18–24.

Yoon Kim and Alexander M Rush. 2016. Sequence-level knowledge distillation. *arXiv preprint arXiv:1606.07947*.

Kevin Lin, Ben Bogin, Mark Neumann, Jonathan Berant, and Matt Gardner. 2019. Grammar-based neural text-to-sql generation. *arXiv preprint arXiv:1905.13326*.

Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of english: The penn treebank.

Nathan Ng, Kyra Yee, Alexei Baevski, Myle Ott, Michael Auli, and Sergey Edunov. 2019. Facebook fair's wmt19 news translation task submission. *arXiv preprint arXiv:1907.06616*.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch.

Alane Suhr, Srinivasan Iyer, and Yoav Artzi. 2018. Learning to map context-dependent sentences to executable formal queries. *arXiv preprint arXiv:1804.06868*.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Barak Turovsky. 2016. Ten years of google translate.

Christopher Makoto Wilt, Jordan Tyler Thayer, and Wheeler Ruml. 2010. A comparison of greedy search algorithms. In *third annual symposium on combinatorial search*.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Kyra Yee, Nathan Ng, Yann N Dauphin, and Michael Auli. 2019. Simple and effective noisy channel modeling for neural machine translation. *arXiv preprint arXiv:1908.05731*.

Pengcheng Yin and Graham Neubig. 2018. Tranx: A transition-based neural abstract syntax parser for semantic parsing and code generation. *arXiv preprint arXiv:1810.02720*.

Luke S Zettlemoyer and Michael Collins. 2012. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *arXiv preprint arXiv:1207.1420*.

A Appendices

A.1 Implementation Differences Compared to Fairseq

In our main machine translation experiments, the FIXED baseline is Fairseq's implementation. Running our own beam search implementation—the basis of VAR-BATCH and VAR-STREAM—with a fixed beam width differs from Fairseq's implementation as follows. In our implementation, henceforth FIXED-OURS, terminated candidates $y_{ij}^{l_{t_0}}$ with $i > 1$ are kept on the beam, added to our list of final outputs only if they become the top candidate $y_{i1}^{l_{t_1}}$ in the beam at a subsequent step t_1. Fairseq instead immediately adds $y_{ij}^{l_{t_0}}$ to the list of final outputs at time t_0. The difference is that $y_{ij}^{l_{t_0}}$ may be removed from the beam at time $t > t_0$ if we later find multiple terminated candidates originating from a higher-probability beam $y_{ij'}^{l_{t_0}}$ for $j' < j$, e.g. between $t = 7$ and $t = 8$ in the light blue beam in Figure 1.

FIXED-OURS is slower than Fairseq's implementation. However, while the two implementations achieve more similar BLEU on the development set, FIXED-OURS achieves higher BLEU on the test set (49.75 vs 49.57 on De-En and 39.19 vs 38.98 on Ru-En). See Table 3 for De-En experiment details.

For completeness, we also present results in Table 3 for FIXED-STREAM, our streaming implementation adapted to fixed-size beam search on newstest2018 on the 32GB Nvidia V100, with $k = 50$ as in the FIXED baseline. We keep the $\epsilon = \frac{1}{6}$ hyperparameter. FIXED-STREAM is significantly faster than FIXED-OURS, demonstrating that our streaming method can also speed up fixed-size beam search. However, FIXED-STREAM is slower than Fairseq's implementation, although it outperforms Fairseq. It is possible that our implementation is less optimized, but we do not formally claim this.

Method	BLEU	Wall Clock (s)
FIXED-FAIRSEQ	49.57	891.53 ± 0.77
FIXED-OURS	49.75	1280.59 ± 5.34
FIXED-STREAM	49.75	1004.18 ± 6.82

Table 3: De-En translation experiments test set (newstest2018) on 32GB Nvidia V100 using different implementations of fixed-size beam search. FIXED-FAIRSEQ is the FIXED baseline in the main paper, while FIXED-OURS is our implementation of fixed-size beam search. FIXED-STREAM is a streaming implementation with $\epsilon = \frac{1}{6}$; FIXED-OURS corresponds to $\epsilon = 0$. FIXED-STREAM improves over FIXED-OURS in wall clock, but is still slower than FIXED-FAIRSEQ, although it achieves higher BLEU.

A.2 Alternative Method Analysis

We briefly analyze an alternative streaming method to our proposed VAR-STREAM, which we label VAR-STR-FIFO. At each decoding timestep t, instead of selecting only the beams with minimal l_t as in VAR-STREAM, VAR-STR-FIFO selects beams up to its batch capacity starting with the beam of maximal l_t. In Figure 1, this corresponds to not pausing computation for the purple beam. This is intuitively appealing and has potential advantages: as shown in Table 5, unlike VAR-STREAM which uses slightly more timesteps than VAR-BATCH due to using a slightly smaller effective batch size, VAR-STR-FIFO significantly reduces the number of timesteps required for decoding in the De-En translation task. Yet VAR-STR-FIFO is significantly slower than both VAR-STREAM and VAR-BATCH. This is due to Fairseq's architecture, a transformer reliant on decoder self-attention, causing decoding timesteps with longer l_t to be more expensive (Figure 2). VAR-STR-FIFO suffers because it must pad all selected beams' lengths up to the maximum l_t among those selected.

The difference between VAR-STREAM and VAR-STR-FIFO demonstrates that selecting the correct beams to expand during decoding timesteps can be highly impactful on speed, illustrating a new axis of optimization made possible by streaming. While VAR-STREAM is superior for the transformers used in state-of-the-art machine translation, we hypothesize that VAR-STR-FIFO may be preferred in other applications, especially in RNN architectures which do not use self-attention.

Method	Wall Clock (s)	Timesteps
VAR-BATCH	308.15 ± 3.48	2007
VAR-STREAM	257.20 ± 0.54	2180
VAR-STR-FIFO	343.10 ± 0.41	1538

Table 4: Comparison of variable-size beam search methods on De-En translation of newstest2018 on 32GB Nvidia V100 GPU. All methods achieve equal BLEU and expand the same number of candidates. VAR-STR-FIFO uses the fewest timesteps but takes the most time due to the transformer architecture requiring more time per step at high values of l_t.

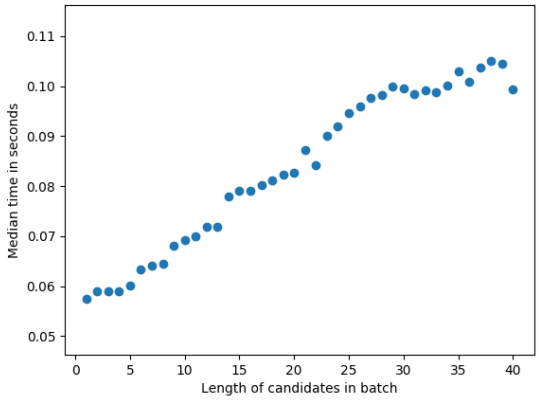

Figure 2: Median time for decoding steps at values of l_t up to 40, for a single run of De-En translation on the 32GB GPU with $k = 50$. The time taken increases roughly linearly with l_t due to self-attention in the transformer.

A.3 Experiment Details and Hyperparameters

We provide details on our setup. All code is written in Pytorch (Paszke et al., 2017). For hardware, for our 32GB and 16GB Nvidia V100 experiments, we use p3dn.24xlarge and p3.2xlarge instances respectively on AWS. Experiments are conducted serially with no other computation on the instance. Due to some variance between instances, all experiments within a single comparable group (e.g., all methods' runs for beam size 50 on a 32GB GPU) are conducted on the same instance.

Additionally, we specify an implementation detail: for the absolute threshold heuristic δ, we note that $y_{i1}^{l_t}$ may be a terminated candidate from a previous timestep.

For heuristic hyperparameters, in all domains we choose pruning heuristics to approximately match the performance of FIXED, based on the development set (newstest2017 for machine translation, and the ATIS and Penn Treebank development sets in semantic and syntactic parsing respectively). As usual, in heuristic selection, there is a tradeoff between time and performance, which we explore here in the machine translation domain.

We run FIXED and VAR-STREAM on the development set (newstest2017) for German-English translation using the 32GB Nvidia V100, using our main paper heuristics $\delta = 1.5, M = 5, \epsilon = \frac{1}{6}$. We additionally run versions where we individually tweak each heuristic, using $\delta = 0.5, \delta = 2.5, M = 3, M = 10, \epsilon = \frac{1}{12}, \epsilon = \frac{1}{4}$.

Both BLEU scores and computation time overall increase with δ and M (Table 5).

ϵ does not affect BLEU. Larger ϵ means we run fewer timesteps at high l_t, but our batch refills are smaller. At least in this machine translation setting, the effect of changing ϵ is typically a few seconds, indicating that our method is not overly sensitive to this hyperparameter choice as long as ϵ is small. (Note we re-adjusted batch sizes in multiples of 64 to saturate the GPU for each ablation.)

During initial hyperparameter selection, we ran $\epsilon = \frac{1}{6}, \frac{1}{3}$ and $\frac{1}{2}$. For M we tested $3, 5$, and 10 and for δ we tested $1.5, 2.5, 5$ and 10 based on manual tuning with single runs. Note that BLEU and F1 scores do not change with multiple trials, as we do not retrain models. Meanwhile, runtimes generally have fairly small standard deviation (see all tables), so we did not heavily optimize. Overall, the speedups enabled by VAR-STREAM over baselines are relatively insensitive to heuristic hyperparameters.

Method	Top 1 BLEU	Wall Clock (s)	Timesteps	Cand. Exp.	Exp. / Step
FIXED	42.71	921.57 ± 2.95	6620	3967200	599
VAR-STREAM	42.71	259.78 ± 2.52	2261	651786	288
$\delta = 0.5$	42.49	104.28 ± 1.35	860	131108	152
$\delta = 2.5$	42.72	558.71 ± 5.49	4325	1788047	413
$M = 3$	42.7	248.45 ± 2.66	2070	625099	302
$M = 10$	42.71	268.62 ± 3.91	2543	655300	258
$\epsilon = \frac{1}{12}$	42.71	255.03 ± 0.49	2479	651786	263
$\epsilon = \frac{1}{4}$	42.71	264.73 ± 1.05	2352	651786	277

Table 5: Exploration of effect of different hyperparameter choices on VAR-STREAM performance on De-En development set (newstest2017). Top 1 BLEU, wall clock (average of 5 runs), total decoding timesteps, total candidate expansions, and average candidate expansions per step for several methods. All experiments on 32GB Nvidia V100. Larger values of δ and M result in a method closer to fixed-width beam search, which tends to increase BLEU while taking more time due to needing more candidate expansions. Efficiency in expansions per step generally increases as the method approaches fixed-width beam search, as variable-width beams are more difficult to pack efficiently. Nevertheless, for the purpose of wall-clock time, this effect is outweighed by the vastly larger number of expansions required by fixed-width beam search.

Method	F1	Oracle	Time (s)	Timesteps	Cand. Exp.	Exp. / Step
GREEDY	87.1	87.1	0.67 ± 0.02	136	2715	20.0
FIXED	87.1	90.0	2.36 ± 0.04	705	27781	39.4
VAR-BATCH	87.1	89.3	2.29 ± 0.02	585	5071	8.7
VAR-STREAM	87.1	89.3	1.47 ± 0.01	126	5071	40.2

Table 6: Semantic parsing experiments on JOBS dataset of job listings. Top-1 F1, oracle reranking F1, wall clock average of 5 runs, total decoding timesteps, total candidate expansions, and average expansions per timestep. Although VAR-STREAM is not much more efficient than FIXED in expansions per timestep, it requires many fewer total expansions and is thus faster. Meanwhile, VAR-STREAM is several times more efficient than VAR-BATCH. However, the variable beam searches suffer slightly in oracle F1 compared to FIXED, while still remaining above GREEDY.

A.4 Additional Semantic Parsing Experiments

In Tables 6 and 7, we present results from applying our method to the JOBS and GEO datasets. We use the same hyperparameters and heuristics as for ATIS, and operate under the same candidate-expansion constraint. VAR-STREAM is substantially faster than Fixed and VAR-BATCH under this setting.

Method	F1	Oracle	Time (s)	Timesteps	Cand. Exp.	Exp. / Step
GREEDY	82.5	82.5	0.96 ± 0.02	138	5492	39.8
FIXED	82.5	89.3	4.54 ± 0.09	1469	57550	39.2
VAR-BATCH	82.5	89.6	4.77 ± 0.05	1344	14154	10.5
VAR-STREAM	82.5	89.6	2.95 ± 0.04	248	14154	57.1

Table 7: Semantic parsing experiments on GEO dataset of geographical queries. VAR-STREAM is substantially more efficient than both FIXED and VAR-BATCH in expansions per timestep, and this is reflected in the wall clock time.

A.5 Dataset Details

A.5.1 Machine Translation

Evaluation datasets (newstest2018 and newstest2017) are available at http://www.statmt.org/wmt19/translation-task.html. newstest2018 contains 2998 and 3000 examples for De-En and Ru-En respectively, while newstest2017 contains 3004 and 3001.

A.5.2 Semantic Parsing

Datasets can be obtained by running the data scripts at https://github.com/Alex-Fabbri/lang2logic-PyTorch, which re-implements Dong and Lapata (2016) in PyTorch. We use Dong and Lapata (2016)'s training, development (for ATIS), and test sets. ATIS, JOBS, and GEO contain 5410, 640, and 880 examples respectively.

A.5.3 Syntactic Parsing

The Penn Treebank dataset and splits are available at https://github.com/jhcross/span-parser. The training, data, and test splits are the standard Penn Treebank splits (sections 2-21 for training, 22 for development, and 23 for test, containing 39832, 1700, and 2416 examples respectively).