Accelerating Transformer Inference for Translation via Parallel Decoding

Andrea Santilli[1], Silvio Severino[1], Emilian Postolache[1], Valentino Maiorca[1], Michele Mancusi[1], Riccardo Marin[2], Emanuele Rodolà[1]
[1]Sapienza University of Rome [2]University of Tübingen
santilli@di.uniroma1.it

Abstract

Autoregressive decoding limits the efficiency of transformers for Machine Translation (MT). The community proposed specific network architectures and learning-based methods to solve this issue, which are expensive and require changes to the MT model, trading inference speed at the cost of the translation quality. In this paper, we propose to address the problem from the point of view of decoding algorithms, as a less explored but rather compelling direction. We propose to reframe the standard greedy autoregressive decoding of MT with a parallel formulation leveraging Jacobi and Gauss-Seidel fixed-point iteration methods for fast inference. This formulation allows to speed up *existing models* without training or modifications while *retaining translation quality*. We present three parallel decoding algorithms and test them on different languages and models showing how the parallelization introduces a speedup up to 38% w.r.t. the standard autoregressive decoding and nearly 2x when scaling the method on parallel resources. Finally, we introduce a decoding dependency graph visualizer (DDG*viz*) that let us see how the model has learned the conditional dependence between tokens and inspect the decoding procedure.

1 Introduction

In recent years there have been dramatic improvements in Machine Translation (MT) (Edunov et al., 2018; Liu et al., 2020) thanks to the transition to neural models and the advent of the Transformer architecture (Vaswani et al., 2017). These models can produce high-quality translations while being extremely parallelizable during training. However, Transformers are used sequentially at inference time, generating one token per time (i.e., sending each token as input for the next autoregressive iteration). This process of autoregressive inference hampers the efficiency of neural machine translation systems in terms of latency, limiting applications and portability. Considering that these systems are extensively used in production multiple times to produce new translations (e.g., Google Translate[1], DeepL Translator[2]), even a minor speedup would be beneficial in the long run, especially if the translation is done on embedded devices.

To address this issue, the community proposed *ad-hoc* trained models specific for parallel machine translation under the umbrella term of Non-Autoregressive Machine Translation models (NAT) (Gu et al., 2018). These models produce the translation in parallel but require (i) a complete reengineering of the MT system, (ii) extensive training resources and (iii) complex design choices like distillation from larger autoregressive models. These requirements are quite demanding and not easily satisfiable. For example, production systems are heavily optimized for hardware and software and even introducing a minimal modification requires non-trivial human effort (Wu et al., 2016; Kim et al., 2019). Furthermore, training a new model from scratch is not always possible due to non-released training data or low-resource languages having few or lacking parallel corpora.

In this paper, we propose to address the problem of parallel machine translation with an orthogonal approach consisting in novel decoding algorithms that work in parallel and can be used on top of *existing autoregressive models* for MT. We overcome previous limitations with a flexible and generic method that does not require any modification to the model or costly retraining. Specifically, inspired by previous successes in speeding up feed-forward computation for image generation (Song et al., 2021b), we reframe the greedy autoregressive decoding for MT as a system of nonlinear equations solvable in parallel. This simple formulation speeds up the decoding procedure by using fixed-point iteration methods like Jacobi and Gauss-Seidel while having mathematical guarantees on

[1]https://translate.google.com/
[2]https://www.deepl.com/

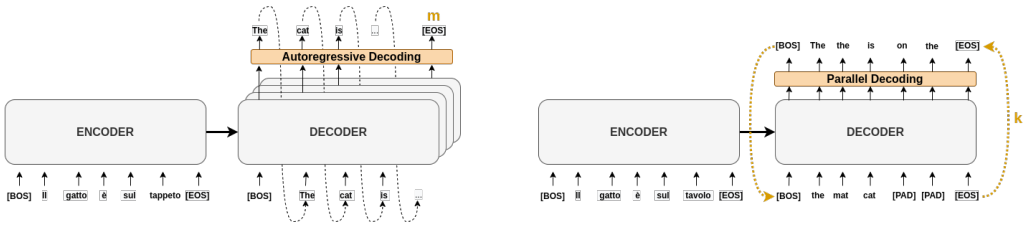


Figure 1: **On the left**, the classical Autoregressive Decoding for MT. The target sentence is produced token-by-token sequentially, sending the partial result as input for the next autoregressive iteration up to the length m of the target. **On the right** Parallel Decoding proposed in this paper. This method changes only the *decoding algorithm* (orange block) and is usable on top of any autoregressive model without modifications. Parallel Decoding algorithms resolve the whole sentence or a block of b tokens in parallel: an initial translation (init tokens) is gradually refined with k steps until a stopping condition is reached. Crucially, $k \leqslant m$ with quality guarantees and overall decoding speedups.

the quality of the translation. A high-level description of the method is available in (Fig. 1). Our contributions can be summarized as the following:

- We reframe the standard greedy autoregressive decoding procedure in MT with a parallel formulation, introducing three parallel decoding algorithms (PJ, PGJ, HGJ) and a stopping condition that preserves translation quality.
- We perform extensive experiments with different transformer sizes (base and large) and datasets, showing speedups up to 38% in time, obtaining a nearly $2\times$ speedup when scaling the model on parallel resources while preserving quality. To the best of our knowledge, this is one of the first studies to introduce a speedup in multilingual machine translation.
- We introduce a decoding dependency graph visualizer (DDG*viz*) to inspect the learned tokens' conditional dependence and when parallel decoding is effective.

All the code is publicly released[3].

2 Related Work

Gu et al. (2018) first introduced Non-Autoregressive Translation models (NAT) as ad-hoc trained models capable of producing the translation all at once in parallel. With NATs, it is possible to consistently reduce the latency and speed up the translation at the expense of a slightly worse translation quality due to the multimodality problem (i.e., we lose the dependency between tokens in the target output). Finding a tradeoff between translation quality and speed is an active research direction, with current methods trying to fill the gap in terms of translation quality (Geng et al., 2021; Savinov et al., 2022). Nevertheless, all proposed NAT models are learning-based and require different tricks to reach the quality of autoregressive models (Gu and Kong, 2021). The most common is the sequence-level knowledge distillation of large autoregressive models into parallel models (Kim and Rush, 2016). Other approaches include defining alternative training objectives (Ghazvininejad et al., 2020a; Saharia et al., 2020; Du et al., 2021; Huang et al., 2021), architectures that model dependencies between output sentence tokens (Ghazvininejad et al., 2019; Qian et al., 2021; Song et al., 2021a; Gu and Kong, 2021; Song et al., 2022) or multi-iteration methods (Ghazvininejad et al., 2020b; Kasai et al., 2020; Hao et al., 2021; Geng et al., 2021; Savinov et al., 2022; Huang et al., 2022; Xia et al., 2022) that apply iterative refinements to a translation, trading some speed for greater quality. In our approach, we also employ iterative refinements of solutions to non-linear equations, but *we do not perform any training or modification to the model*. Other works that require retraining or modifications to the model add additional decoding heads (Stern et al., 2018) or use shallow decoders (Kasai et al., 2021). We refer the reader to Xiao et al. (2022) for a thorough survey on NAT methods. Further orthogonal approaches use specialized hardware (TPU) with low-precision calculations (Wu et al., 2016) or software optimizations (Kim et al., 2019). In the context of Grammatical Error Correction, Sun et al. (2021) recently proposed aggressive parallel decoding, assuming that the model output is similar to the input. More recently, inspiring our work, Song et al. (2021b) showed that it is possible to parallelize feedforward computations by thinking of them as a system of non-linear

[3] https://github.com/teelinsan/parallel-decoding

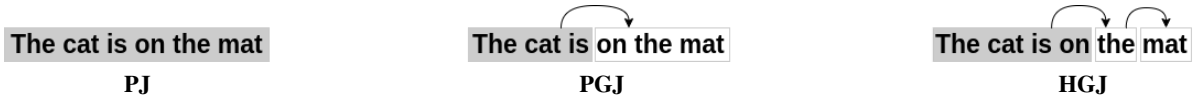


Figure 2: **Parallel Decoding algorithms**: **PJ** resolves the whole sequence in parallel iteratively. **PGJ** resolves blocks in parallel; once a block is finished, it moves on to the next one and decodes it again in parallel (in figure $b = 3$). **HGJ** decodes the sentence in parallel as PGJ up to a certain length h; afterwards, it goes autoregressively until [EOS] token is generated. Decoding actually happens in sub-word tokens (not depicted here).

equations. They parallelized the backpropagation of RNNs, feedforward layers and autoregressive generative models on images. We extend the approach defined on dense pixel prediction to the discrete conditional token generation in MT. While this work was under submission and anonymity period, Leviathan et al. (2022), Chen et al. (2023) and Kim et al. (2023) concurrently proposed decoding approaches that speed up inference of a large transformer model by using another smaller model to draft tokens. Compared to these approaches our method requires just an existing autoregressive model (no matter the size) and mathematically guarantees the output quality. In the next Section we describe the method.

3 Method

In this Section, we introduce notations, develop the theory behind Parallel Decoding, present three algorithms (Fig. 2), and discuss the initialization and stopping conditions for the proposed approaches.

3.1 Notation

The goal of MT is to translate a sentence $\mathbf{x}$ in a source language (e.g., Italian) with its translation $\mathbf{y}$ in the target language (e.g., English). Source and target sentences are generally tokenized in words or subwords (Kudo and Richardson, 2018; Schuster and Nakajima, 2012; Sennrich et al., 2016; Kudo, 2018); here, we use the subfix notation $\mathbf{x} = (x_1, \dots, x_n)$ and $\mathbf{y} = (y_1, \dots, y_m)$ to indicate specific tokens in the sequence. We also use the notation $\mathbf{x}_{1:n}$ to indicate a slice of a sequence as a shorthand of $\mathbf{x} = (x_1, \dots, x_n)$. From a probabilistic perspective, an MT model estimates $p_\theta(\mathbf{y} \mid \mathbf{x})$. Once an MT model has been trained, the inference phase is traditionally performed by sampling tokens from the model probability conditioned on the input sequence $\mathbf{x}$ and previously generated tokens $(y_1, \dots, y_{i-1})$:

$$p_\theta\left(y_i \mid y_1, \dots, y_{i-1}, \mathbf{x}\right). \tag{1}$$

Different sampling strategies are employed (e.g., Greedy, Top-K, Top-p (Kool et al., 2020; Holtzman et al., 2020)) alongside search strategies that estimate the total conditional probability (e.g., Greedy search, Beam search (Reddy, 1977)). The most straightforward strategy, Greedy Search, selects the element y_i of a sequence with:

$$y_i = \arg\max p_\theta(y_i \mid \mathbf{y}_{1:i-1}, \mathbf{x}). \tag{2}$$

Given the formalization above, a standard autoregressive setting runs m inference steps *sequentially* to generate an output sequence of m elements.

Parallel Decoding. Given Equation (2), it is possible to write the greedy decoding procedure on all tokens as:

$$\begin{cases} y_1 = \arg\max p_\theta(y_1 \mid \mathbf{x}) \\ y_2 = \arg\max p_\theta(y_2 \mid y_1, \mathbf{x}) \\ \vdots \\ y_m = \arg\max p_\theta(y_m \mid \mathbf{y}_{1:m-1}, \mathbf{x}) \end{cases} \tag{3}$$

Defining $f(y_i, \mathbf{y}_{1:i-1}, \mathbf{x}) = y_i - \arg\max p_\theta(y_i \mid \mathbf{y}_{1:i-1}, \mathbf{x})$, we can rewrite the system of Equations (3) as:

$$\begin{cases} f(y_1, \mathbf{x}) = 0 \\ f(y_2, y_1, \mathbf{x}) = 0 \\ \vdots \\ f(y_m, \mathbf{y}_{1:m-1}, \mathbf{x}) = 0 \end{cases} \tag{4}$$

This system has m non-linear equations (each equation employ a neural network) with m variables.

3.2 Parallel Decoding Algorithms

The autoregressive decoding implicitly solves the system of Equations (4) by substitution, i.e., given the [BOS] token and the input sentence x, it solves equations from first to last, progressively replacing the resolved variables. In this paper, we rely on Jacobi and Gauss-Seidel (GS) fixed-point iteration methods (Ortega and Rheinboldt, 1970) to solve in parallel system (4) until a stopping condition is reached. This formulation is particularly flexible and has several advantages: Firstly, it is completely agnostic to the underlying MT model used; Secondly, it can be analyzed with analytical tools and

has guarantees of convergence to the exact solution for system (4); Thirdly, it can be potentially extended by drawing from the numerical methods literature for non-linear equations solving methods (Saad, 2003). We see that, with the proper stopping condition, it is possible to have quality guarantees over the output. We present here three algorithms (PJ, PGJ, HGJ) that leverage these fixed-point iteration methods to speedup decoding in MT.

Parallel Jacobi (PJ) Decoding. First, we propose Algorithm 1. This algorithm works by initializing a draft translation for the whole target sentence and then iteratively translating the whole sentence in parallel until the stopping condition is triggered. This is equivalent to solving system (4) with Jacobi, hence the name of the method.

Parallel GS-Jacobi (PGJ) Decoding. Decoding the whole target sentence in parallel may introduce difficulties in inferring long dependencies between tokens since the underlying model is trained to model the conditional distribution of a token given the previous tokens. In general, we observed that shorter dependencies are easily predicted since decoding happens at the sub-word level, and the model can decode sub-word unities in parallel rather than the whole sentence. To this end, we propose Algorithm 2, called GS-Jacobi, that splits the sentence into contiguous b-dimensional blocks. Starting from the first one, it decodes in parallel all its elements. Once a block is finished or the stopping condition within the block is triggered, the algorithm performs a sequential (Gauss-Seidel) step and proceeds with (Jacobi) decoding on the next one.

Hybrid GS-Jacobi (HGJ) Decoding. Algorithms 1 and 2 assume to know beforehand the number of equations m (i.e., the target length). This is not usually the case for MT, where the model dynamically controls the length through the emission of a special end-of-sentence token [EOS]. To overcome this issue, we propose a flexible Hybrid Algorithm 3 that mixes PGJ computations with standard autoregressive decoding. This algorithm performs parallel GS-Jacobi decoding up to a certain prefixed length h. If the [EOS] token is generated within a block, then the algorithm stops, returning the translation up to [EOS]. Otherwise, the algorithm concludes the translation by reaching the [EOS] token with standard autoregressive decoding. In this case, the length h regulates the trade-off between parallel and sequential computation, limiting the waste of resources beyond [EOS].

Algorithm 1 Parallel Jacobi Decoding

Input: $\mathbf{x} = (x_1, \ldots, x_n)$, p_θ
Output: $\mathbf{y} = (y_1, \ldots, y_m)$

$\mathbf{y} \leftarrow \textsc{InitT}(\mathbf{x})$
$m \leftarrow len(\mathbf{y})$
for $i = 1$ to m **do**
  $\mathbf{o} \leftarrow copy(\mathbf{y}_{1:m})$
  $\mathbf{y}_{1:m} \leftarrow \arg\max(p_\theta(\mathbf{y}_{1:m}|\mathbf{y}_{1:m}, \mathbf{x}))$
  $stop \leftarrow \textsc{StopC}(\mathbf{o}, \mathbf{y}_{1:m})$
  if $stop$ **then**
    break
  end if
end for
return $\mathbf{y}$

3.3 Initialization and Stopping

Our algorithms share two components: the *initialization procedure* and the *stopping condition*.

Initialization $\textsc{InitT}(\mathbf{x})$. The initialization procedure is a function that inputs the source sentence and produces an initial draft translation as output. In this paper we experimented with a simple initialization procedure that initialize the translation with all [PAD] tokens. This choice is fast and doesn't depend on the underlying MT model. We leave as future work the research of different initialization procedures to further speedup the decoding.

Stopping Condition $\textsc{StopC}(\mathbf{y}^{k-1}, \mathbf{y}^k)$. The stopping condition is a function that takes as input the previous-iteration sentence $\mathbf{y}^{k-1}$ and the current-iteration sentence $\mathbf{y}^k$ and decides whether to stop the algorithm or not. This function is crucial since it regulates the trade-off between speedup and translation quality. In this paper we introduce as stopping condition for MT:

$$\mathbf{y}^{k-1} - \mathbf{y}^k = \mathbf{0} \tag{5}$$

i.e., the sentence from the previous step has not changed. This stop condition allows for preserving quality and quickening translations simultaneously.

3.4 Quality Guarantees

Compared to NAT methods which do not have any quality guarantee since a novel parallel model is trained from scratch, our formulation guarantees to have the same quality of using autoregressive decoding with the same MT model. System (4) is known in literature as a *triangular system* of m equations with m variables, this characterization allows to state an important property.

Decoding Algorithm	en→de		de→en		en→ro		ro→en	
	Speed	BLEU	Speed	BLEU	Speed	BLEU	Speed	BLEU
Opus								
Greedy Autoregressive	1.00×	28.24	1.00×	33.10	1.00×	27.41	1.00×	37.01
Beam Search (beam = 5)	0.71×	28.68	0.72×	33.92	0.70×	27.61	0.72×	37.84
PJ Decoding	0.73×	28.24	0.75×	33.10	0.66×	27.41	0.66×	37.01
PGJ Decoding (b = 5)	1.28×	28.24	1.32×	33.10	1.33×	27.41	1.29×	37.01
PGJ Decoding (b = 3)	**1.34**×	28.24	**1.37**×	33.10	**1.38**×	27.41	**1.35**×	37.01
HGJ Decoding (b = 3)	**1.34**×	28.24	**1.37**×	33.10	**1.38**×	27.41	**1.35**×	37.01
MBart50								
Greedy Autoregressive	1.00×	23.97	1.00×	31.58	1.00×	24.99	1.00×	34.77
Beam Search (beam = 5)	0.76×	24.93	0.77×	32.61	0.77×	25.31	0.76×	35.16
PJ Decoding	0.88×	23.97	0.88×	31.58	0.86×	24.99	0.85×	34.77
PGJ Decoding (b = 5)	0.98×	23.97	0.98×	31.58	0.97×	24.99	0.99×	34.77
PGJ Decoding (b = 3)	**1.06**×	23.97	**1.08**×	31.58	**1.03**×	24.99	**1.04**×	34.77
HGJ Decoding (b = 3)	1.05×	23.97	1.07×	31.58	1.01×	24.99	1.02×	34.77

Table 1: Comparison of parallel decoding algorithms (highlighted in grey) with sequential decoding using Opus (CPU) and MBart50 (GPU) on WMT14 and WMT16. Speed is measured in time w.r.t. the autoregressive baseline.

		WMT17 En-Fi		ITTB En-Hi		IWSLT15 En-Vi		FLORES En-It		FLORES En-Fr	
Dec. Algorithm	**Speed**	←	→	←	→	←	→	←	→	←	→
PJ	Iters	1.04×	1.04×	1.04×	1.04 ×	1.06×	1.03×	1.02×	1.04×	1.03×	1.03×
	Time	0.86×	0.88×	0.89×	0.89×	0.87×	0.86×	0.85×	0.86×	0.85×	0.85×
PGJ (b=3)	Iters	1.07×	1.09×	1.09×	1.09×	1.10×	1.07 ×	1.07×	1.08×	1.08×	1.11×
	Time	1.01×	1.05×	1.05×	1.07×	1.04×	1.02×	1.02×	1.03×	1.03×	1.05×
HGJ (b=3)	Iters	1.05×	1.07×	1.07×	1.07×	1.07×	1.06×	1.07×	1.06×	1.05×	1.07×
	Time	1.01×	1.03×	1.04×	1.05×	1.03×	1.01×	1.01×	1.02×	1.01×	1.03×

Table 2: Comparison over different languages in terms of speedup and iterations on MBart50. Arrows indicate the direction of translation. Qualitative results and BLEU scores are available in the appendix D.

Proposition 1. *Algorithms 1, 2, 3 converge and yield the same results of greedy autoregressive decoding in at most m parallel iterations, for any initialization and providing stopping condition* (5).

We refer the reader to Song et al. (2021b) for a formal proof. Intuitively, with m steps the algorithm used the same number of iterations of autoregressive, hence the final solution is the same regardless the initialization. In this worst case, the wall-clock time is the same but in general the algorithm reach the stopping condition earlier with a lower wall-clock time and overall speedup.

3.5 DDG*viz*

Equation 1 models the dependency between tokens in the decoding phase. In the classical autoregressive mode, each token depends on all the previous ones for the generation. However, it is possible to show that this dependency is actually relaxed (i.e., not all tokens depends on all the previous ones), thus it would be interesting to visualize the actual distribution $p_\theta\left(y_i \mid \cdot, \mathbf{x}\right)$ learned by an existing MT model. To this end, we build the Decoding Dependency Graph visualizer (DDG*viz*) to visualize the dependency graph of tokens in the decoding phase. In the standard autoregressive decoding this graph is a fully-connected chain where the i-th token is connected to all the previous tokens, starting from the encoding $\mathbf{x}$: to decode y_i you need to decode first $y_1, \ldots, y_{i-1}$. Instead we show that there are skipping connections between independent tokens that can be visualized with DDG*viz*. We detail DDG*viz* with an example in section 4.3.

4 Experiments

4.1 Experimental Settings

Datasets. We evaluate our approach using standard evaluation datasets proposed for parallel MT (Gu et al., 2018): WMT14 English-German [En-De], WMT16 English-Romanian [En-Ro] (Bojar et al., 2016, 2014). Additionally, we tested our method on different language pairs with varying (low-medium) resources: IWSLT15 (English-Vietnamese [En-Vi]) (Tran et al., 2015), ITTB (English-Hindi [En-Hi]) (Kunchukuttan et al., 2018), WMT17 (English-Finnish [En-Fi]) (Bojar et al., 2017), FLORES-101 (English-Italian [En-It]; English-French [En-Fr]) (Goyal et al., 2022). All the datasets are evaluated in both directions.

Evaluation. All the evaluations are performed using the official test split for each dataset, downloaded using Huggingface dataset library (Lhoest et al., 2021). No training or hyperparameters tun-

Method	Requirements			WMT14		Efficiency		
	Arch	Loss	seq-KD	Speed ↑	BLEU ↑	Train FLOPs ↓	Total FLOPs ↓	FLOPs / Speed ↓
Parallel Decoding - HGJ (Ours)	No	No	No	1.34×	28.24	**0**	**2.53e+13**	**1.89e+13**
SUNDAE † (Savinov et al., 2022)	Yes	No	No	1.4×	28.46	5.27e+21	5.27e+21	3.77e+21
ShallowDec (12-1) (Kasai et al., 2021)	Yes	No	No	1.4×	26.90	1.02e+19	1.02e+19	7.30e+18
Semi-NAT (Wang et al., 2018)	Yes	No	Yes	1.5×	26.90	1.55e+17	1.55e+17	1.03e+17
DisCo (Kasai et al., 2020)	Yes	Yes	Yes, Big	3.5×	27.34	4.06e+19	4.06e+19	1.16e+19
DSLP (Huang et al., 2021)	Yes	Yes	Yes	14.8×	27.02	1.93e+19	1.93e+19	1.31e+18
F-VAE (Gu and Kong, 2021)	Yes	Yes	Yes, Big	16.5×	27.49	4.06e+19	4.06e+19	2.46e+18

Table 3: Comparison of different methods for parallel MT on WMT14 En-De. Results are ordered by speed, highlighted in green the two highest BLEU scores, † indicates diffusion models. Existing methods require training, architecture modifications, additional losses to force parallel translation, and distillation from an additional MT transformer model ("Big" indicates the size). Details on FLOPs computation are available in the Appendix C.

ing is performed. We use SacreBLEU to evaluate the translation quality (Papineni et al., 2002; Post, 2018). We measure speedup in wall-clock time and iterations w.r.t. the same autoregressive model. GPU times are calculated after calling `torch.cuda.synchronize()`. All the experiments were performed by caching the past Keys and Values of the transformer to further speed up the computation (Ramachandran et al., 2017) and in the online inference setting with batch size equal to 1. For the Jacobi and GS-Jacobi algorithms, we assume to know beforehand the length m of the target and measure the speedup in the ideal condition. For the Hybrid GS-Jacobi algorithm, we set h equal to the maximum (i.e., the stopping condition is triggered within a parallel block) to decouple the effective speedup regardless of the length produced by the initialization function (see Section 3.2). We remark that HGJ does not assume to know beforehand the target length and is applicable to real MT translation scenarios.

Model Configuration. We tested transformer models in the two standard configurations: base (512 model dimension, 6 attention layers for both encoder and decoder) and big (1024 model dimension, 12 attention layers for both encoder and decoder). We used pretrained models of Opus (Tiedemann and Thottingal, 2020) for the former and MBart50 (Tang et al., 2020) for the latter. Opus is a transformer base model (74M parameters) trained on language pairs from the homonymous dataset (Zhang et al., 2020). MBart50 is a large multilingual transformer model fine-tuned for translation on 50 languages (610M parameters). We tested the models on CPU since this is the default environment for MT models in production, except for the model MBart50 which runs on GPU. We run the experiments on a standard 16-core machine, except for the scaling experiments. Additional specifications are available in Appendix B

4.2 Algorithms Comparison

In Table 1 we compare the proposed parallel decoding algorithms with the standard sequential autoregressive decoding baselines. As we can observe, the fastest algorithms are PGJ Decoding (b=3) and HGJ Decoding (b=3) which are up to 34% and 38% times faster on Opus and up to 5% and 8% faster on MBart50, depending on the language pair. We note also that results empirically show that all the parallel decoding algorithms guarantee the same quality of greedy autoregressive decoding, as evidenced by the unchanged BLEU scores. This is an experimental verification of the formal Proposition 1. The table also shows that the Beam Search algorithm with a beam size of 5 generally performs better in terms of BLEU score, although at a cost of speed. This difference in terms of BLEU is expected, as beam search is a heuristic search strategy, while our method is a decoding algorithm. We discussed better this aspect in the "Beam Search" paragraph. Nevertheless, beam search is ∼30% slower than greedy autoregressive and 63% to 68% slower than PGJ, depending on the model and language pair. This means that the proposed parallel algorithms allow trading a little translation quality (e.g., on en→ro the difference between beam search and parallel decoding algorithms in BLEU is just 0.20 points) for greater decoding speed.

Another aspect to note is that the algorithms PJ and PGJ (b=5) are sometimes slower than greedy autoregressive. There are several factors that can influence the actual wall-clock time like how the underlying hardware schedule and execute the various operations, which might vary according to the architecture and the workload. In particular, longer sequences (e.g., the whole sentence in PJ or blocks of 5 tokens in PGJ) may require more memory to store, and the CPU/GPU may have to perform more memory accesses, which can slow down the computation (although theoretically it should happen in parallel). In the end, these computational

overheads slow down the actual execution. This is also the case for the difference in speedups between MBart50 and Opus. We better investigated this aspect in the section "Computational Scaling" and report in the appendix results on a different architecture, with also results in terms of iterations speedups which are architecture agnostic.

4.3 Analysis and Validation

Cross Languages. In order to demonstrate the robustness of our decoding algorithms with respect to the translation languages, we leveraged the multilingual capabilities of the MBart50 model and selected a diverse range of language pairs for evaluation. The results, presented in Table 2, show that both PGJ and HGJ achieve a consistent speedup in comparison to the autoregressive decoding method, with an improvement ranging from 2-7% for PGJ and 1-5% for HGJ, regardless of the language pair used. Additionally, we observed a speedup in terms of iterations of 7-11% for PGJ and 5-7% for HGJ. These findings indicate that our algorithms have the potential to match or surpass the speedup in terms of wall-clock time by fully exploiting this saving in terms of iterations. We note that, similar to the previous experiment, PJ suffers from an overhead problem. To the best of our knowledge, this is one of the first studies that have achieved a speedup in multilingual machine translation, concurrent with the work of Song et al. (2022), while this latter is significantly different in spirit and requirements (NAT model). We leave BLEU scores in the Appendix D for space constraints together with qualitative results in different languages.

Computational Scaling. In Figure 3, we present an analysis of the scalability of our proposed methods in relation to increasing computational resources. Starting with 8 cores, our methods demonstrate a slight improvement in terms of wall-clock time for PGJ and HGJ, with speedups of 1.11 and 1.09 respectively. On the other hand, this amount of resources is too restricting for PJ which needs to fit the whole sentence and thus achieve a score of 0.46 due to the aforementioned overhead problem. As the resources are increased, our method demonstrates the ability to effectively leverage hardware and significantly reduce decoding time, while the autoregressive baseline is constrained by sequential processing. With 122 cores, a substantial speedup of $1.98\times$ and $1.99\times$ is achieved for PGJ and HGJ respectively, while the autoregressive baseline is

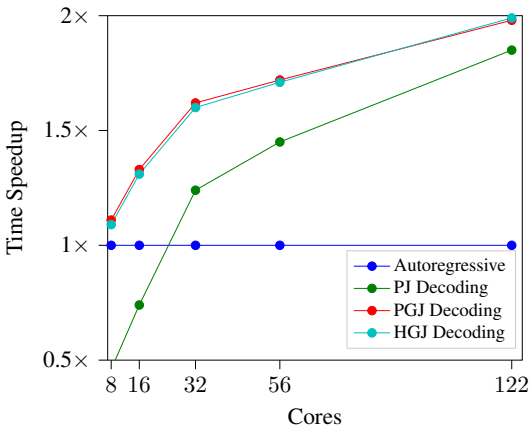


Figure 3: Scaling experiments on WMT16 En-De with PGJ and HGJ blocks = 3. Increasing the number of available resources (number of CPU cores) allows the methods to decrease the parallel overheads. As a result, the speedup increases and the methods scale.

bounded by sequential processing at $1.00\times$. It is important to note that this experiment does not simulate a real production system, but rather it is meant to show what results can be achieved when the underlying computation is properly optimized to run in parallel. In our case, we simulated this setting with increasing cores, nevertheless similar results can be achieved with additional software optimizations to further reduce latency and overheads (Ahmed et al., 2022; Kim et al., 2019) and increase the speed gain with parallel-optimized computations. Overall this experiment serves as a proof of concept for the capabilities of parallel decoding in contexts with limited overhead and shows a promising direction for further improvements.

Comparison with NATs. Table 3 reports the comparison of our parallel decoding algorithm with a selection of NAT methods for parallel MT. Following prior works, we report for each method the speedup relative to the autoregressive transformer base baseline from their original paper (Xiao et al., 2022). It is worth noting that, although these methods can achieve higher speedups, they are very demanding in terms of computational resources which must be accounted for in a fair comparison. To estimate quantitatively this cost, we evaluated the number of floating point operations (FLOPs) required for training and inference on WMT14.

Results show that our method HGJ uses the least number of computational resources, even considering the additional cost at inference time. Relating the speedup obtained with the used resources (FLOPs/speed), our method still achieves the best

cost-benefit ratio. Furthermore, NATs generally degrade the translation quality if compared to their autoregressive baseline. On the contrary, our method mathematically guarantees the same quality of autoregressive decoding, which is higher than standard NAT models.

SUNDAE achieves BLEU of 28.46, but requires more resources than training RoBERTa (Liu et al., 2019) on 16 TPUs (see Appendix C). Other methods require further elaborate techniques like profound architectural changes, additional losses to force parallel translation and sequence-level distillation from large autoregressive transformers (Gu and Kong, 2021). Our approach is a decoding method that does not involve any training or modification to the model and can be used to speed up existing models on standard desktop hardware.

Speedup Analysis. We provide here a preliminary analysis of the factors responsible for the observed speedup in our method. We first distinguish between two types of speedup: wall-clock speedup and iterations speedup. The former is primarily driven by the parallelization capability of our method, as demonstrated in the "Computational Scaling" section. With parallel decoding, underlying operations can be optimized and fused to be executed fastly. Compared to Sheng et al. (2023), our method allows parallelizing sequence operations ("row-by-row" setting). The latter instead may vary consequently to several factors (e.g., model/vocabulary size, training data, language, etc). For this reason, we experimented with several variations of these factors (models Transformer Base vs. Big, vocabularies 58K Marian vs. 250K MBart50, languages, and hardware). While it is challenging to decouple different elements, our analysis point out several interesting insights. For example, we observed that iteration results on MBart50 are generally higher compared to Marian (Tables 2-5), possibly due to the finer-grained tokenization of MBart50. We also hypothesize that language and linguistic features, such as inflectionally rich or agglutinative/gendered languages, may influence iteration speedups. To facilitate this type of analysis, we developed DDG*viz*, which we believe will be useful for research in this area.

Visualizing Parallel Decoding. In previous experiments, we demonstrated that parallel decoding is feasible. This suggests that the dependency learned by the model between certain tokens is relaxed, as some tokens can be decoded in parallel.

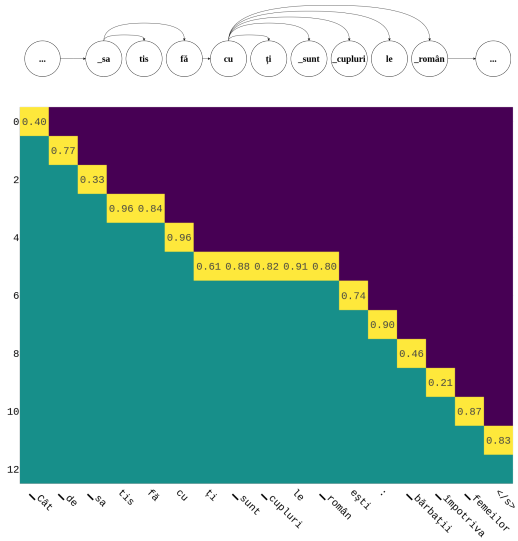


Figure 4: **DDG*viz*.** Visualization of the translation En-Ro: "How satisfied are the Romanian couples: men versus women"→"Cât de satisfacuti sunt cuplurile românești: bărbații împotriva femeilor". (Highlighted tokens decoded in parallel). **On top**: the Decoding Dependency Graph, omitting redundant edges on non-parallel tokens to ease visualization. **On bottom**: DDG*viz* shows at each Parallel Jacobi iteration (vertical axis) which tokens have been generated in parallel (horizontal axis) with the corresponding probability (cell number).

Analyzing and understanding when this happens allows shedding light on the behavior of existing models and a separate study focused on this issue would be needed. In this work, we lay the ground for a such study introducing the necessary inspection tools. While we have already introduced DDG*viz* in Section 3.5, in this experiment we show how it works and how it can be used with a practical example. In summary, the DDG*viz* visualizer allows to show the *real* decoding distribution $p_\theta\left(y_i \mid \cdot, \mathbf{x}\right)$ learned by a MT model. This decoding distribution is plotted as a graph, where a connection indicates the dependency $p_\theta(y_i \mid \cdot)$, by using Parallel Jacobi decoding. At each PJ decoding iteration (vertical axis of Figure 4), DDG*viz* keeps track of which tokens have been correctly decoded w.r.t. the gold autoregressive reference of the model, showing the tokens correctly decoded and the probability of each one (horizontal axis). Figure 4 shows DDG*viz* applied on an example. The example shows that for $y_4 = \_sa$ it is possible to decode more than one token in parallel $y_5 = tis$, $y_6 = fa$, hence here the decoding of y_6 does not depend on the decoding of y_5 - $p_\theta\left(y_6 \mid \mathbf{y}_{1:4}, \mathbf{x}\right)$. We observed this phenomenon frequently, explaining the speedups in the previous experiments. The

example also shows that the model is able to decode five tokens in parallel after $y_7 = \_cu$. This is a peculiar case since the model, given *"How satisfi_"*, is generating all at once *"_ed are the Romanian couples"* (proposed here in English for better readability, original version in Romanian is available in Figure). This example indeed shows how DDG*viz* can be used to highlight possible biases encoded in the model as it is not clear how the model can be so confident (see cell probability) that after "satisfied" the most straightforward tokens to decode are "Romanian couples" (Chang et al., 2019; Savoldi et al., 2021). We leave other use cases for future works and show in Appendix D several visualizations with equally interesting phenomena.

5 Conclusions

In this paper, we showed that is possible to speed up *existing* machine translation models by simply changing the decoding procedure with a parallel formulation. We introduced three parallel decoding methods which achieve consistent speedups without requiring any training, modifications, or quality loss. Our solution is orthogonal to previous approaches proposed in literature which are demanding in terms of data, computational resources, and engineering effort. This makes it particularly useful in limited-resource scenarios when one or all of these requirements are not satisfiable and alternatives like NATs cannot be deployed. While our method is not without shortcomings, it represents a valuable first step in the development of parallel decoding algorithms for machine translation that can be seamlessly integrated with any model. We believe that further advancements in this area, including the exploration of optimal initialization procedures and stopping conditions, as well as the use of alternative parallel solvers for non-linear equations, will close the gap with learning-based techniques and continue to improve the efficiency and effectiveness of parallel decoding algorithms.

Acknowledgements

We would like to thank Sébastien Bratières for his throughout feedback provided on this project. This work is supported by Translated with an Imminent Research Grant, ERC Starting Grant No. 802554 (SPECGEO), and PRIN 2020 project n.2020TA3K9N "LEGO.AI". Riccardo Marin is also supported by an Alexander von Humboldt Foundation Research Fellowship.

Limitations

The proposed algorithms allow to speed up an existing model out-of-the-box, without any modification or retraining. However, there are some considerations to bear in mind when using parallel decoding in order to have a speedup in terms of wall-clock time. Firstly, as the name implies, the method executes the decoding phase in parallel. Therefore, to appreciate the speedup one should be able to run computations in parallel. Using parallel decoding without parallel resources or parallel-optimized software may increase wall-clock time due to overheads, leading to a waste of computation. This is further discussed in Section 4.3 "Computational Scaling". The reported wall-clock time results are thus to be considered within the scope of the experimental setup proposed in this paper and they may vary depending on the underlying hardware and software. Secondly, the method allows speedup of the decoding by scaling on parallel resources. This implies an additional computational cost during the inference phase to achieve a speedup. While using parallel decoding, one should consider a trade-off between the desired acceleration and the utilization of computational resources. Thirdly, since our method performs the decoding in parallel, as for NAT systems, it is difficult to combine it with Beam Search. Beam Search is inherently a dynamic programming algorithm and it is not possible to efficiently maximize the joint probability of the large search space without using sequential intermediate computations. We better explain this aspect in the next paragraph.

Beam Search. Beam search is widely employed to enhance the translation quality in MT (Sutskever et al., 2014; Bahdanau et al., 2015) as well as in other domains such as audio (Reddy, 1977; Postolache et al., 2023). However, it is an inherently sequential procedure that stores partial joint probabilities of the entire sequence (beams) while progressing with autoregressive decoding. Determining the maximal joint probability of all sequences in parallel is a challenging task, equivalent to a full maximum a posteriori (MAP) estimation. This is an open research problem and it is also an issue for NAT methods. NAT methods patch up this limitation with sequence-level KD which has the advantage of "not requiring any beam search at test-time" (Kim and Rush, 2016) thanks to learning and distillation from large models. Since our

method is a decoding algorithm, we cannot use the same approach without learning. Nevertheless, the quality guarantee allows our methods to have performance on par with greedy autoregressive and generally better than a NAT model. We think of our method, not as a replacement for beam search, but rather as a way to obtain a speedup at inference time that is a middle ground between autoregressive greedy decoding (high quality, no requirements, no speed) and NATs (quality compromises, increasing requirements with increasing speed). Future works might address the quality gap with beam search by combining parallel decoding with alternative techniques like Minimum Bayes Risk (Eikema and Aziz, 2020).

Ethics Statement

Increasing the inference speed of MT can positively impact society by giving people a fast and good translation. This will enable people from different language backgrounds to communicate with each other and help remove cultural and trade barriers. As demonstrated by comparing the number of FLOPs in Table 3, our method uses fewer resources compared to alternatives and thus has a smaller carbon footprint, making it a more sustainable choice (Strubell et al., 2019). Furthermore, since our method does not involve training procedures or change the quality of results, we do not introduce any societal bias (e.g. racism, sexism, homophobia) into the translations. The latter, however, can be introduced through data in the training of the backbone autoregressive models and NATs. It is the task of those who train these models to mitigate this problem. DDG*viz* can also help investigate and visualize some potential harmful biases encoded in the model like in Figure 4.

References

Ibrahim Ahmed, Sahil Parmar, Matthew Boyd, Michael Beidler, Kris Kang, Bill Liu, Kyle Roach, John Kim, and Dennis Abts. 2022. Answer fast: Accelerating bert on the tensor streaming processor. In *2022 IEEE 33rd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 80–87. IEEE.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Aleš Tamchyna. 2014. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 12–58, Baltimore, Maryland, USA. Association for Computational Linguistics.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Shujian Huang, Matthias Huck, Philipp Koehn, Qun Liu, Varvara Logacheva, Christof Monz, Matteo Negri, Matt Post, Raphael Rubino, Lucia Specia, and Marco Turchi. 2017. Findings of the 2017 conference on machine translation (wmt17). In *Proceedings of the Second Conference on Machine Translation, Volume 2: Shared Task Papers*, pages 169–214, Copenhagen, Denmark. Association for Computational Linguistics.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurelie Neveol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, and Marcos Zampieri. 2016. Findings of the 2016 conference on machine translation. In *Proceedings of the First Conference on Machine Translation*, pages 131–198, Berlin, Germany. Association for Computational Linguistics.

Kai-Wei Chang, Vinodkumar Prabhakaran, and Vicente Ordonez. 2019. Bias and fairness in natural language processing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP): Tutorial Abstracts*, Hong Kong, China. Association for Computational Linguistics.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating large language model decoding with speculative sampling.

Cunxiao Du, Zhaopeng Tu, and Jing Jiang. 2021. Order-agnostic cross entropy for non-autoregressive machine translation. In *International Conference on Machine Learning*, pages 2849–2859. PMLR.

Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. 2018. Understanding back-translation at scale. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 489–500, Brussels, Belgium. Association for Computational Linguistics.

Bryan Eikema and Wilker Aziz. 2020. Is MAP decoding all you need? the inadequacy of the mode in neural machine translation. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 4506–4520, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Xinwei Geng, Xiaocheng Feng, and Bing Qin. 2021. Learning to rewrite for non-autoregressive neural machine translation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3297–3308, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Marjan Ghazvininejad, Vladimir Karpukhin, Luke Zettlemoyer, and Omer Levy. 2020a. Aligned cross entropy for non-autoregressive machine translation. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3515–3523. PMLR.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. 2019. Mask-predict: Parallel decoding of conditional masked language models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6112–6121, Hong Kong, China. Association for Computational Linguistics.

Marjan Ghazvininejad, Omer Levy, and Luke Zettlemoyer. 2020b. Semi-autoregressive training improves mask-predict decoding. *arXiv preprint arXiv:2001.08785*.

Naman Goyal, Cynthia Gao, Vishrav Chaudhary, Peng-Jen Chen, Guillaume Wenzek, Da Ju, Sanjana Krishnan, Marc'Aurelio Ranzato, Francisco Guzmán, and Angela Fan. 2022. The Flores-101 evaluation benchmark for low-resource and multilingual machine translation. *Transactions of the Association for Computational Linguistics*, 10:522–538.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O.K. Li, and Richard Socher. 2018. Non-autoregressive neural machine translation. In *International Conference on Learning Representations*.

Jiatao Gu and Xiang Kong. 2021. Fully non-autoregressive neural machine translation: Tricks of the trade. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 120–133, Online. Association for Computational Linguistics.

Yongchang Hao, Shilin He, Wenxiang Jiao, Zhaopeng Tu, Michael Lyu, and Xing Wang. 2021. Multi-task learning with shared encoder for non-autoregressive machine translation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3989–3996, Online. Association for Computational Linguistics.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *International Conference on Learning Representations*.

Chenyang Huang, Hao Zhou, Osmar R. Zaïane, Lili Mou, and Lei Li. 2021. Non-autoregressive translation with layer-wise prediction and deep supervision. *CoRR*, abs/2110.07515.

Xiao Shi Huang, Felipe Perez, and Maksims Volkovs. 2022. Improving non-autoregressive translation models without distillation. In *International Conference on Learning Representations*.

Jungo Kasai, James Cross, Marjan Ghazvininejad, and Jiatao Gu. 2020. Non-autoregressive machine translation with disentangled context transformer. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5144–5155. PMLR.

Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah Smith. 2021. Deep encoder, shallow decoder: Reevaluating non-autoregressive machine translation. In *International Conference on Learning Representations*.

Sehoon Kim, Karttikeya Mangalam, Jitendra Malik, Michael W. Mahoney, Amir Gholami, and Kurt Keutzer. 2023. Big little transformer decoder.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327, Austin, Texas. Association for Computational Linguistics.

Young Jin Kim, Marcin Junczys-Dowmunt, Hany Hassan, Alham Fikri Aji, Kenneth Heafield, Roman Grundkiewicz, and Nikolay Bogoychev. 2019. From research to production and back: Ludicrously fast neural machine translation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 280–288, Hong Kong. Association for Computational Linguistics.

Wouter Kool, Herke van Hoof, and Max Welling. 2020. Ancestral gumbel-top-k sampling for sampling without replacement. *Journal of Machine Learning Research*, 21(47):1–36.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, Melbourne, Australia. Association for Computational Linguistics.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *CoRR*, abs/1808.06226.

Anoop Kunchukuttan, Pratik Mehta, and Pushpak Bhattacharyya. 2018. The IIT Bombay English-Hindi parallel corpus. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2022. Fast inference from transformers via speculative decoding.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. 2021. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 175–184, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Xiaodong Liu, Kevin Duh, Liyuan Liu, and Jianfeng Gao. 2020. Very deep transformers for neural machine translation. *arXiv preprint arXiv:2008.07772*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

J.M. Ortega and W.C. Rheinboldt. 1970. *Iterative Solution of Nonlinear Equations in Several Variables*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104).

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 of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels. Association for Computational Linguistics.

Emilian Postolache, Giorgio Mariani, Michele Mancusi, Andrea Santilli, Cosmo Luca, Emanuele Rodola, et al. 2023. Latent autoregressive source separation. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Lihua Qian, Hao Zhou, Yu Bao, Mingxuan Wang, Lin Qiu, Weinan Zhang, Yong Yu, and Lei Li. 2021. Glancing transformer for non-autoregressive neural machine translation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1993–2003, Online. Association for Computational Linguistics.

Prajit Ramachandran, Tom Le Paine, Pooya Khorrami, Mohammad Babaeizadeh, Shiyu Chang, Yang Zhang, Mark A. Hasegawa-Johnson, Roy H. Campbell, and Thomas S. Huang. 2017. Fast generation for convolutional autoregressive models. *CoRR*, abs/1704.06001.

Raj Reddy. 1977. *Speech understanding systems: A summary of results of the five-year research effort*. Carnegie Mellon University.

Yousef Saad. 2003. *Iterative methods for sparse linear systems*. SIAM.

Chitwan Saharia, William Chan, Saurabh Saxena, and Mohammad Norouzi. 2020. Non-autoregressive machine translation with latent alignments. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1098–1108, Online. Association for Computational Linguistics.

Nikolay Savinov, Junyoung Chung, Mikolaj Binkowski, Erich Elsen, and Aaron van den Oord. 2022. Step-unrolled denoising autoencoders for text generation. In *International Conference on Learning Representations*.

Beatrice Savoldi, Marco Gaido, Luisa Bentivogli, Matteo Negri, and Marco Turchi. 2021. Gender Bias in Machine Translation. *Transactions of the Association for Computational Linguistics*, 9:845–874.

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2012, Kyoto, Japan, March 25-30, 2012*, pages 5149–5152. IEEE.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Daniel Y Fu, Zhiqiang Xie, Beidi Chen, Clark Barrett, Joseph E Gonzalez, et al. 2023. High-throughput generative inference of large language models with a single gpu. *arXiv preprint arXiv:2303.06865*.

Jongyoon Song, Sungwon Kim, and Sungroh Yoon. 2021a. AligNART: Non-autoregressive neural machine translation by jointly learning to estimate alignment and translate. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1–14, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Yang Song, Chenlin Meng, Renjie Liao, and Stefano Ermon. 2021b. Accelerating feedforward computation via parallel nonlinear equation solving. In *International Conference on Machine Learning*, pages 9791–9800. PMLR.

Zhenqiao Song, Hao Zhou, Lihua Qian, Jingjing Xu, Shanbo Cheng, Mingxuan Wang, and Lei Li. 2022. switch-GLAT: Multilingual parallel machine translation via code-switch decoder. In *International Conference on Learning Representations*.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy. Association for Computational Linguistics.

Xin Sun, Tao Ge, Furu Wei, and Houfeng Wang. 2021. Instantaneous grammatical error correction with shallow aggressive decoding. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5937–5947, Online. Association for Computational Linguistics.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27.

Yuqing Tang, Chau Tran, Xian Li, Peng-Jen Chen, Naman Goyal, Vishrav Chaudhary, Jiatao Gu, and Angela Fan. 2020. Multilingual translation with extensible multilingual pretraining and finetuning. *CoRR*, abs/2008.00401.

Jörg Tiedemann and Santhosh Thottingal. 2020. OPUS-MT — Building open translation services for the World. In *Proceedings of the 22nd Annual Conferenec of the European Association for Machine Translation (EAMT)*, Lisbon, Portugal.

Viet Hong Tran, Huyen Vu Thong, Nguyen Van-Vinh, and Trung Le Tien. 2015. The English-Vietnamese machine translation system for IWSLT 2015. In *Proceedings of the 12th International Workshop on Spoken Language Translation: Evaluation Campaign*, pages 80–83, Da Nang, Vietnam.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Chunqi Wang, Ji Zhang, and Haiqing Chen. 2018. Semi-autoregressive neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 479–488, Brussels, Belgium. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

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*.

Heming Xia, Tao Ge, Furu Wei, and Zhifang Sui. 2022. Lossless speedup of autoregressive translation with generalized aggressive decoding.

Yisheng Xiao, Lijun Wu, Junliang Guo, Juntao Li, Min Zhang, Tao Qin, and Tie-yan Liu. 2022. A survey on non-autoregressive generation for neural machine translation and beyond. *arXiv preprint arXiv:2204.09269*.

Biao Zhang, Philip Williams, Ivan Titov, and Rico Sennrich. 2020. Improving massively multilingual neural machine translation and zero-shot translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1628–1639, Online. Association for Computational Linguistics.

A Algorithms details

We propose here the pseudocode of Algorithms 2 and 3 due to space limitations in the main body of the paper.

The function $copy(y_{i:i+b})$ creates a copy of the tensor in input detached from the source. This is done in practice to avoid the overwriting of pointers to the same memory location. Function CHECKEOS$(y_{i:i+b})$ returns the index of the token EOS in the block if present, else -1. Function

Algorithm 2 Parallel GS-Jacobi Decoding

Input: $\mathbf{x} = (x_1, \ldots, x_n), p_\theta, b$
Output: $\mathbf{y} = (y_1, \ldots, y_m)$

$\mathbf{y} \leftarrow \text{INITT}(\mathbf{x})$
$m \leftarrow len(\mathbf{y})$
$i \leftarrow 1$
while $i \leqslant m$ **do**
  $\mathbf{o} \leftarrow copy(y_{i:i+b})$
  $\mathbf{y}_{i:i+b} \leftarrow \arg\max(p_\theta(\mathbf{y}_{i:i+b}|\mathbf{y}_{1:i+b}, \mathbf{x}))$
  $stop \leftarrow \text{STOPC}(o, y_{i:i+b})$
  if $stop$ **then**
    $i \leftarrow i + b$
    break
  end if
end while
return $\mathbf{y}$

Algorithm 3 Hybrid GS-Jacobi Decoding

Input: $\mathbf{x} = (x_1, \ldots, x_n), p_\theta, b$
Output: $\mathbf{y} = (y_1, \ldots, y_m)$

$\mathbf{y} \leftarrow \text{INITT}(\mathbf{x})$
$h \leftarrow len(\mathbf{y})$
$i \leftarrow 1$
$eos\_cond \leftarrow False$
while $i \leqslant h$ **do**
  $\mathbf{o} \leftarrow copy(\mathbf{y}_{i:i+b})$
  $\mathbf{y}_{i:i+b} \leftarrow \arg\max(p_\theta(\mathbf{y}_{i:i+b}|\mathbf{y}_{1:i+b}, \mathbf{x}))$
  $stop \leftarrow \text{STOPC}(\mathbf{o}, \mathbf{y}_{i:i+b})$
  $eos\_ind \leftarrow \text{CHECKEOS}(\mathbf{y}_{i:i+b})$
  if $stop$ **and** $eos\_ind > -1$ **then**
    $\mathbf{y} \leftarrow \mathbf{y}_{1:eos\_ind}$
    $eos\_cond \leftarrow True$
    break
  end if
  if $stop$ **then**
    $i \leftarrow i + b$
    break
  end if
end while
while $eos\_cond\ != True$ **do**
  $y_i \leftarrow \arg\max(p_\theta(y_i|y_{i-1}, \mathbf{x}))$
  $i \leftarrow i + 1$
  $eos\_cond \leftarrow \text{ISEOS}(y_i)$
end while
return $\mathbf{y}$

CHECKEOS(y_i) returns $True$ if the tokes in exactly the token EOS, else $False$. The function $\arg\max$ selects from the model distribution over the vocabulary the index (token) with maximum probability. This procedure is done for all the tokens in parallel, in the case of parallel decoding, or for just a single token in the case of autoregressive decoding. Generally, the output is the prediction for the next token; hence it should be shifted left before the reassignment to a variable. We omitted this implementation detail for clarity.

B Additional implementation details

We run Opus experiments in table 1 on an AMD EPYC Milan with 16 cores at 2.45 GHz and 64GB of RAM (accessible on Google Cloud - `c2d-standard-16`). For the scalability experiment in figure 3, we also used Google Cloud instances with an increasing number of cores (referred to as `c2d-standard-XX`, where `XX` is the number of used cores). Experiments with MBart50 on table 1, 2 and 5 are performed on a Desktop machine with Ubuntu 20.04.4 LTS, AMD Ryzen 9 3900X 12-Core Processor, 32GB of RAM, and a Palit Nvidia 3090 GPU. Additional experiments with Opus in table 5 are also performed on this machine. Models are implemented in Pytorch 1.11.0 (Paszke et al., 2019) and the Huggingface Transformer library (Wolf et al., 2020). We used python 3.8 and NVIDIA-SMI Drivers 510.73.05 with CUDA version 11.6. For OPUS we used Huggingface models available on the hub under the tag `Helsinki-NLP/opus-mt-{src}-{tgt}` except for the language pair Ro-En where we used the model `Helsinki-NLP/opus-mt-roa-en` and the pair En-De where we used the checkpoint `opus-2021-02-22` [4]. For the model MBart50, we used the `facebook` pre-trained model available on the hub with the tag `mbart-large-50-many-to-many-mmt`. Since this is a multilingual model, we prepend the source and target language tag corresponding properly to the language pair to be translated. We report results for a single run over the test dataset since we found low variance in estimates with multiple runs which can be calculated by simply varying the corresponding parameter in the `config.yaml` file.

Model	Train FLOPs	Infer. FLOPs	Total FLOPs
Semi-NAT	1.55e17	2.08e13	1.55e17
Shallow Dec.	1.02e19	1.15e13	1.02e19
DSLP	1.93e19	1.58e13	1.93e19
F-VAE	4.06e19	1.58e13	4.06e19
DisCo	4.06e19	1.58e13	4.06e19
SUNDAE	5.27e21	1.58e14	5.27e21
BERT base	6.43e19	-	-
BERT large	1.92e20	-	-
RoBERTa	3.19e21	-	-

Table 4: FLOPs comparison with other models.

C FLOPs calculation details

We measured computational complexity using floating point operations (FLOPs), which, as the name imply, counts the number of floating point operation performed by a model. This is a standard metric used in literature to measure hardware-agnostic complexity. This means that hardware and software optimizations are not counted in the score (Wu et al., 2016; Kim et al., 2019). We used the ELECTRA flops calculator[5] inserting the number of parameters and the number of training step performed for each model analyzed in table 3 according to the training specification in each paper. For inference FLOPs, we computed the decoding cost of each sentence in the testset of WMT14 En-De for each model. For a scale reference, we report in here Table 4 training flops of other well-known architecture. The code package contains the scripts to replicate all the experiments.

D Additional results

We propose here additional results to the experiments in the paper that were omitted due to limitations constraints. Table 5 shows the same experiments of Table 1 in the main paper, proposed here on a standard desktop CPU with also the speedup in terms of iterations. It is possible to observe that in the case of MBart50 and PGJ there is a speedup of $8 - 11\%$ in terms of iterations compare to a time speedup of $3 - 8\%$. This means that there is room for improvement for our algorithm. Furthermore, results show that the time speedups are consistent also with standard desktop hardware. Table 6 shows the BLEU scores for the cross-lingual experiment. It is possible to observe that parallel decoding algorithms guarantee quality compared to greedy autoregressive and are not so distant from beam search. We show also here in table 5 some qualitative results for the experiments in table 2. Finally, we propose additional visualizations using DGG*viz* in Figure 6.

[4]https://object.pouta.csc.fi/Tatoeba-MT-models/eng-deu/opus-2021-02-22.zip

[5]https://github.com/google-research/electra/blob/master/flops_computation.py

Decoding Algorithm	en→de		de→en		en→ro		ro→en	
	Time	Iters	Time	Iters	Time	Iters	Time	Iters
Opus								
Greedy Autoregressive	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
Beam Search (beam = 5)	0.71×	1.00×	0.71×	1.00×	0.70×	1.00×	0.72×	1.00×
PJ Decoding	0.72×	1.03×	0.74×	1.04×	0.69×	1.04×	0.67×	1.03×
PGJ Decoding (b = 3)	**1.16**×	1.04×	**1.19**×	1.07×	1.17×	1.05×	**1.17**×	1.03×
HGJ Decoding (b = 3)	**1.16**×	1.04×	**1.19**×	1.06×	1.17×	1.05×	**1.17**×	1.03×
MBart50								
Greedy Autoregressive	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
Beam Search (beam = 5)	0.76×	1.00×	0.77×	1.00×	0.77×	1.00×	0.76×	1.00×
PJ Decoding	0.88×	1.03×	0.88×	1.03×	0.86×	1.04×	0.85×	1.03×
PGJ Decoding (b = 3)	**1.06**×	1.10×	**1.08**×	1.11×	**1.03**×	1.08×	**1.04**×	1.11×
HGJ Decoding (b = 3)	1.05×	1.07×	1.07×	1.01×	1.01×	1.02×	1.02×	1.08×

Table 5: Comparison of parallel decoding algorithms (highlighted in grey) with sequential decoding using Opus (CPU) and MBart50 (GPU) on WMT14 and WMT16. Speed is showed here both in Time and Iterations w.r.t. the greedy autoregressive baseline.

	WMT17 En-Fi		ITTB En-Hi		IWSLT15 En-Vi		FLORES En-It		FLORES En-Fr	
Dec. Algorithm	←	→	←	→	←	→	←	→	←	→
Autoregressive	17.55	25.34	16.50	24.70	31.92	33.94	22.78	26.38	39.51	38.90
Beam Search	18.39	26.04	16.87	25.24	32.14	34.59	23.52	26.80	39.59	39.21
PJ	17.54	25.35	16.50	24.69	31.92	33.94	22.78	26.38	39.50	38.90
PGJ (b=3)	17.55	25.35	16.50	24.70	31.93	33.94	22.78	26.38	39.51	38.90
HGJ (b=3)	17.55	25.35	16.50	24.70	31.93	33.94	22.78	26.38	39.51	38.90

Table 6: BLEU scores on MBart50.

Example 1 - Wmt16 En-Ro			
TARGET	Dl Corbyn va adresa primele dintre cele șase întrebări la care are dreptul la scurt timp după prânz; prestația sa va fi probabil analizată îndeaproape de mass-media și parlamentarii laburiști.	Times (s)	BLEU
A	Dl Corbyn va ridica pentru a adresa prima dintre cele şase întrebări alocate la scurt timp după miezul zilei, iar performanţa sa va fi probabil examinată îndeaproape de presă şi de parlamentarii laburişti.	0.51	19.71
PJ	Dl Corbyn va ridica pentru a adresa prima dintre cele şase întrebări alocate la scurt timp după miezul zilei, iar performanţa sa va fi probabil examinată îndeaproape de presă şi de parlamentarii laburişti.	0.56	19.71
PGJ	Dl Corbyn va ridica pentru a adresa prima dintre cele şase întrebări alocate la scurt timp după miezul zilei, iar performanţa sa va fi probabil examinată îndeaproape de presă şi de parlamentarii laburişti.	0.45	19.71
HGJ	Dl Corbyn va ridica pentru a adresa prima dintre cele şase întrebări alocate la scurt timp după miezul zilei, iar performanţa sa va fi probabil examinată îndeaproape de presă şi de parlamentarii laburişti.	**0.44**	19.71

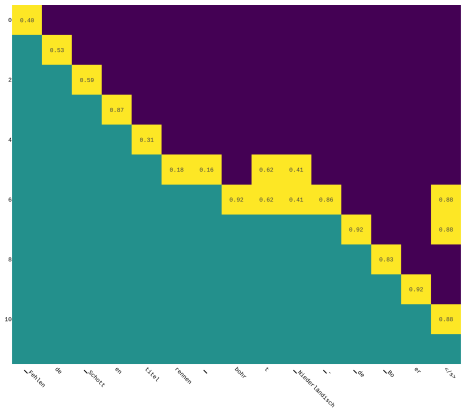
Example 2 - Flores En-It			
TARGET	Quando un piccolo gruppo di esseri viventi (una piccola popolazione) si separa dalla popolazione principale alla quale appartiene (per esempio se si sposta oltre una catena montuosa o un fiume, o si sposta su una nuova isola, rendendo quindi difficile un eventuale ritorno), esso si ritroverà probabilmente in un ambiente diverso da quello in cui si trovava prima.	Times (s)	BLEU
A	Quando un piccolo gruppo di esseri viventi si separa dalla popolazione principale da cui provengono, come se si muovano su una catena di montagne o su un fiume o se si trasferiscono su una nuova isola per non poter tornare facilmente, si troveranno spesso in un ambiente diverso da quello in cui erano prima.	0.61	31.69
PJ	Quando un piccolo gruppo di esseri viventi si separa dalla popolazione principale da cui provengono, come se si muovano su una catena di montagne o su un fiume o se si trasferiscono su una nuova isola per non poter tornare facilmente, si troveranno spesso in un ambiente diverso da quello in cui erano prima.	0.73	31.69
PGJ	Quando un piccolo gruppo di esseri viventi si separa dalla popolazione principale da cui provengono, come se si muovano su una catena di montagne o su un fiume o se si trasferiscono su una nuova isola per non poter tornare facilmente, si troveranno spesso in un ambiente diverso da quello in cui erano prima.	**0.58**	31.69
HGJ	Quando un piccolo gruppo di esseri viventi si separa dalla popolazione principale da cui provengono, come se si muovano su una catena di montagne o su un fiume o se si trasferiscono su una nuova isola per non poter tornare facilmente, si troveranno spesso in un ambiente diverso da quello in cui erano prima.	0.59	31.69

Example 3 - Wmt14 En-De			
TARGET	Bei der diesjährigen Veranstaltung gibt es Auftritte von Wanda Sykes, Kathy Griffin und Bill Maher sowie auch von „Stand Up for Heroes“, einer jährlichen Musik- und Comedy-Benefizveranstaltung für Armeeveteranen im Madison Square Garden, bei der unter anderem Bruce Springsteen, Jon Stewart, Roger Waters und Bill Cosby auftreten.	Times (s)	BLEU
A	Zu den diesjährigen Veranstaltungen gehören Auftritte von Wanda Sykes, Kathy Griffin und Bill Maher sowie "Stand Up for Heroes", ein jährlicher Musik- und Komödie-Vorteil für Militärveteranen, im Madison Square Garden, mit u.a. Bruce Springsteen, Jon Stewart, Roger Waters und Bill Cosby.	1.30	47.04
PJ	Zu den diesjährigen Veranstaltungen gehören Auftritte von Wanda Sykes, Kathy Griffin und Bill Maher sowie "Stand Up for Heroes", ein jährlicher Musik- und Komödie-Vorteil für Militärveteranen, im Madison Square Garden, mit u.a. Bruce Springsteen, Jon Stewart, Roger Waters und Bill Cosby.	2.43	47.04
PGJ	Zu den diesjährigen Veranstaltungen gehören Auftritte von Wanda Sykes, Kathy Griffin und Bill Maher sowie "Stand Up for Heroes", ein jährlicher Musik- und Komödie-Vorteil für Militärveteranen, im Madison Square Garden, mit u.a. Bruce Springsteen, Jon Stewart, Roger Waters und Bill Cosby.	1.09	47.04
HGJ	Zu den diesjährigen Veranstaltungen gehören Auftritte von Wanda Sykes, Kathy Griffin und Bill Maher sowie "Stand Up for Heroes", ein jährlicher Musik- und Komödie-Vorteil für Militärveteranen, im Madison Square Garden, mit u.a. Bruce Springsteen, Jon Stewart, Roger Waters und Bill Cosby.	**1.08**	47.04

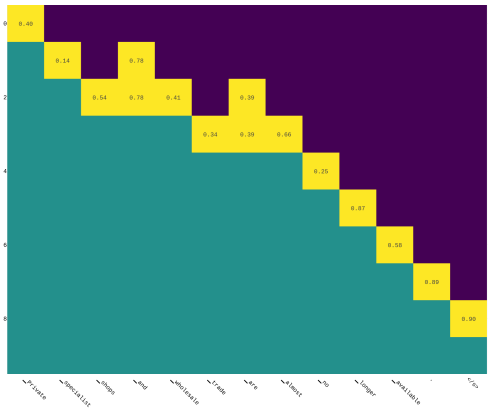
Example 4 - Flores En-Fr			
TARGET	Cinq minutes après le début de l’exposition, un vent se met à souffler pour atteindre, environ une minute plus tard, la vitesse de 70km/h... puis la pluie arrive, mais si forte et si grosse qu’elle frappe votre peau comme une aiguille, puis la grêle tombe du ciel, les gens paniquent, crient et se roulent dessus.	Times (s)	BLEU
A	Cinq minutes après l’exposition, le vent commence à tourner, environ un minute plus tard, le vent atteint 70 km/h, puis la pluie arrive, mais si forte et si grande qu’elle vous frappe la peau comme une aiguille, puis le hail tombe du ciel, les gens paniquent, s’expriment et se courent l’un sur l’autre.	0.82	39.90
PJ	Cinq minutes après l’exposition, le vent commence à tourner, environ un minute plus tard, le vent atteint 70 km/h, puis la pluie arrive, mais si forte et si grande qu’elle vous frappe la peau comme une aiguille, puis le hail tombe du ciel, les gens paniquent, s’expriment et se courent l’un sur l’autre.	0.94	39.90
PGJ	Cinq minutes après l’exposition, le vent commence à tourner, environ un minute plus tard, le vent atteint 70 km/h, puis la pluie arrive, mais si forte et si grande qu’elle vous frappe la peau comme une aiguille, puis le hail tombe du ciel, les gens paniquent, s’expriment et se courent l’un sur l’autre.	0.73	39.90
HGJ	Cinq minutes après l’exposition, le vent commence à tourner, environ un minute plus tard, le vent atteint 70 km/h, puis la pluie arrive, mais si forte et si grande qu’elle vous frappe la peau comme une aiguille, puis le hail tombe du ciel, les gens paniquent, s’expriment et se courent l’un sur l’autre.	**0.72**	39.90

Example 5 - IWSLT15 En-Vi			
TARGET	Tôi yêu sức mạnh của tiến hoá, và tôi nhận ra một điều rất cơ bản đối với mọi sự sống trong những sinh vật đơn bào, mỗi tế bào chỉ đơn giản là phân chia, và mọi thông tin di truyền trong tế bào đó được truyền sang hai tế bào con.	Times (s)	BLEU
A	Tôi đã yêu thích sức mạnh của sự tiến hoá và tôi nhận ra một điều rất căn bản trong hầu hết sự tồn tại của sự sống trong các sinh vật đơn bào mỗi tế bào đơn giản là chia ra và tất cả năng lượng di truyền của tế bào đó được vận hành trong cả hai tế bào con.	0.61	31.45
PJ	Tôi đã yêu thích sức mạnh của sự tiến hoá và tôi nhận ra một điều rất căn bản trong hầu hết sự tồn tại của sự sống trong các sinh vật đơn bào mỗi tế bào đơn giản là chia ra và tất cả năng lượng di truyền của tế bào đó được vận hành trong cả hai tế bào con.	0.71	31.45
PGJ	Tôi đã yêu thích sức mạnh của sự tiến hoá và tôi nhận ra một điều rất căn bản trong hầu hết sự tồn tại của sự sống trong các sinh vật đơn bào mỗi tế bào đơn giản là chia ra và tất cả năng lượng di truyền của tế bào đó được vận hành trong cả hai tế bào con.	0.54	31.45
HGJ	Tôi đã yêu thích sức mạnh của sự tiến hoá và tôi nhận ra một điều rất căn bản trong hầu hết sự tồn tại của sự sống trong các sinh vật đơn bào mỗi tế bào đơn giản là chia ra và tất cả năng lượng di truyền của tế bào đó được vận hành trong cả hai tế bào con.	**0.53**	31.45

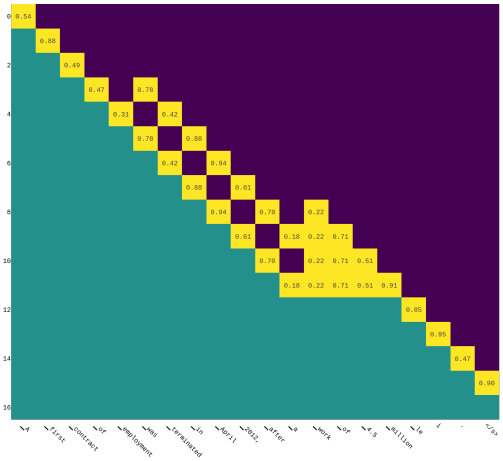
Table 7: Translation examples generated with the autoregressive (A) and the different decoding algorithms proposed (PJ, PGJ, HGJ) on Opus (WMT datasets) and MBart50. The decoding time is shown in seconds.



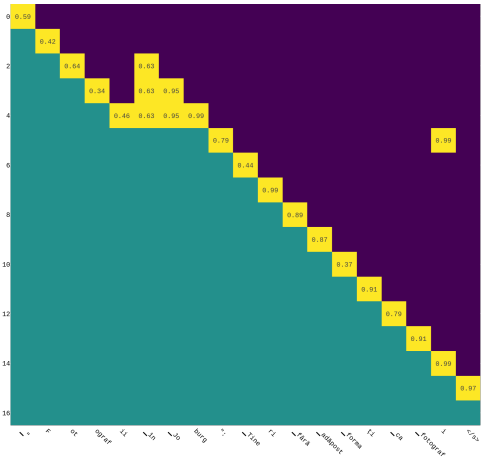
(a) En-De: "Lack of Scots title race bores Dutch - de Boer"→"Fehlende Schottentitelrennen bohrt Niederländisch - de Boer"



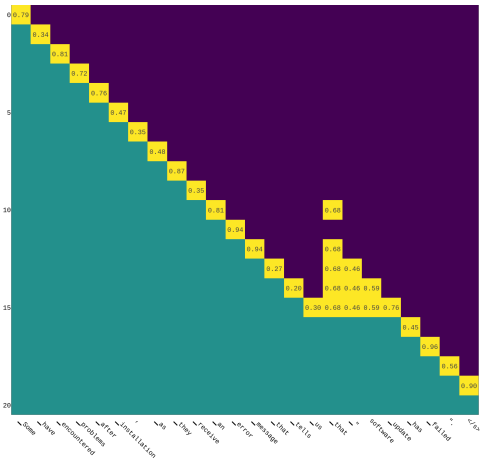
(b) De-En: "Private Fachgeschafte und auch den Großhandel gibt es fast nicht mehr."→"Private specialist shops and wholesale trade are almost no longer available."



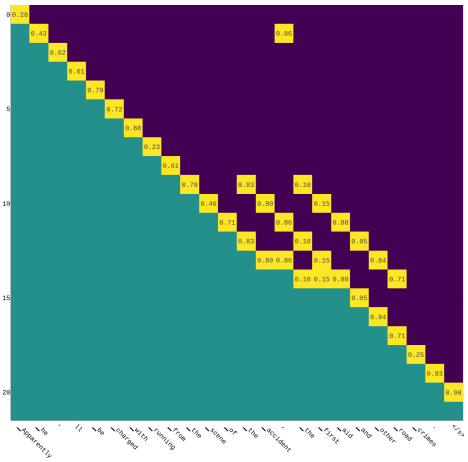
(c) Ro-En: "Un prim contract de lucrări a fost reziliat în aprilie 2012, după ce se efectuaseră lucrări de 4,5 milioane lei."→ "A first contract of employment was terminated in April 2012, after a work of 4.5 million lei."



(d) En-Ro: "'Shot in Joburg': Homeless youth trained as photographers"→ "'Fotografii in Joburg': Tineri fără adăpost formaţi ca fotografi"



(e) De-En: "Einige sind nach der Installation auf Probleme gestoßen, da sie eine Fehlermeldung erhalten, die mitteilt, dass die "Software-Aktualisierung fehlgeschlagen" ist."→"Some have encountered problems after installation, as they receive an error message that tells us that "software update has failed"."



(f) Ro-En: "Se pare că va fi acuzat de fugă de la locul accidentului, neoferirea primului ajutor și alte infracțiuni rutiere."→ "Apparently he'll be charged with running from the scene of the accident, the first aid and other road crimes."

Figure 6: DGG*viz* additional visualizations