Decoding at the Speed of Thought: Harnessing Parallel Decoding of Lexical Units for LLMs

Chenxi Sun[1], Hongzhi Zhang[2], Zijia Lin[2], Jingyuan Zhang[2], Fuzheng Zhang[2], Zhongyuan Wang[2], Bin Chen[2], Chengru Song[2], Di Zhang[2], Kun Gai[2], Deyi Xiong[1]*

[1]College of Intelligence and Computing, Tianjin University, Tianjin, China
[2]Kuaishou Technology Inc., Beijing, China
{cxsun, dyxiong}@tju.edu.cn
{ zhanghongzhi, linzijia, zhangjingyuan06, zhangfuzheng, wangzhongyuan, chenbin08, songchengru, zhangdi08, yuyue06 }@kuaishou.com

Abstract

Large language models have demonstrated exceptional capability in natural language understanding and generation. However, their generation speed is limited by the inherently sequential nature of their decoding process, posing challenges for real-time applications. This paper introduces Lexical Unit Decoding (LUD), a novel decoding methodology implemented in a data-driven manner, accelerating the decoding process without sacrificing output quality. The core of our approach is the observation that a pre-trained language model can confidently predict multiple contiguous tokens, forming the basis for a *lexical unit*, in which these contiguous tokens could be decoded in parallel. Extensive experiments validate that our method substantially reduces decoding time while maintaining generation quality, i.e., 33% speed up on natural language generation with no quality loss, and 30% speed up on code generation with a negligible quality loss of 3%. Distinctively, LUD requires no auxiliary models and does not require changes to existing architectures. It can also be integrated with other decoding acceleration methods, thus achieving an even more pronounced inference efficiency boost. We posit that the foundational principles of LUD could define a new decoding paradigm for future language models, enhancing their applicability for a broader spectrum of applications. All codes are be publicly available at https://github.com/tjunlp-lab/Lexical-Unit-Decoding-LUD-.

Keywords: Parallel Decoding, Lexical Unit Decoding, Large Language Model

1. Introduction

The Transformer architecture (Vaswani et al., 2017) has been crucial in recent advancements in Natural Language Processing (NLP) (Brown et al., 2020; Touvron et al., 2023b). Empirical evidences (OpenAI, 2023; Anil et al., 2023; Hoffmann et al., 2022; Clark et al., 2022; Kaplan et al., 2020; Hernandez et al., 2021) suggest a positive correlation between model size and performance, encouraging the continuous scaling of Large Language Models (LLMs). In this context, the decoder-only architecture has emerged as the de-facto standard. However, while this architecture facilitates rapid training, it still inherently predicts tokens sequentially. This is a constraint rooted in language modeling principles (Shannon, 1948; Bengio et al., 2000). This auto-regressive nature limits generation speed, posing challenges for real-time applications.

Addressing auto-regressive decoding challenges in LLMs led to numerous advancements. The initial breakthroughs occurred in machine translation with non-autoregressive transformers of encoder-decoder architectures. Those methods focus on utilizing latent variables for parallel predictions, but often sacrificed quality. Their architectural disparities generally prevent their direct applicability to accelerating LLMs (Gu et al., 2017; Kaiser et al., 2018; Qian et al., 2021; Cheng and Zhang, 2022; Xiao et al., 2023). Subsequent strategies have predominantly focused on computational optimization, employing techniques that reduce the complexity of models or the number of operations, though often at the expense of a certain degree of quality (Hinton et al., 2015; Jaszczur et al., 2021; Hubara et al., 2017; So et al., 2021). Recent studies have revealed that some tokens are more predictable than others (Zhu et al., 2023). Capitalizing on this insight,

[Question]

What is the primary purpose of mental exercise in psychological well-being?

[Answer]

Mental exercise is **designed to** help people **manage their anxiety**.

Figure 1: Illustration of "**lexical units**" as consecutive token spans. These units, as conceptualized in our study, can potentially be identified and decoded in parallel, enhancing the decoding speed of LLMs.

* Corresponding author

LREC-COLING 2024, pages 4476–4487
20-25 May, 2024.

contemporary *adaptive computation* approaches (Leviathan et al., 2023) aim to efficiently predict these easier tokens and only employ complex models for challenging tokens. While those methods align with established language modeling principles and achieve desired quality levels, they often necessitate modifications to the training paradigm and the model structure (Schwartz et al., 2020; Schuster et al., 2021; Cai et al., 2023) or the integration of auxiliary models (Stern et al., 2018; Leviathan et al., 2023). Such alterations can introduce additional complexities, potentially complicating the model deployment process.

In this study, we identify a notable and naturally emerging pattern within LLMs: certain span of tokens are consistently predicted with high confidence, forming what we term as "lexical units". The observation here intriguingly aligns with findings from linguistics and cognitive science, where humans are believed to process and produce continuous speech by segmenting it into smaller units or chunks (Vetchinnikova et al., 2023). For a visual representation of our conceptualization of lexical units, please refer to Figure 1.

Drawing inspiration from this observation, we introduce Lexical Unit Decoding (LUD), a novel strategy enhancing the decoding speed of LLMs. The essence of LUD lies in the identification of 'lexical units'. A lexical unit is defined as spans of consecutive tokens predicted with high confidence by the model. This critical identification is instrumental for later fine-tuning, steering the model's capability of concurrently predicting multiple tokens during inference. LUD enables model to swiftly predict multiple tokens at once. If certainty wavers, it reverts to single-token predictions. This adaptability sets LUD apart, striking a balance between swift inference and high-quality predictions packed in one model. LUD simplifies deployment by eliminating the need for two separate models. Additionally, its compatibility with arbitrary model architectures, including the prevalent decoder-only architecture, requires no architectural modifications, further facilitating its practical application.

In our evaluations with LLaMA-13B (Touvron et al., 2023a), LUD achieves a 33% acceleration in decoding, maintaining superb output quality. When tested on programming languages, which inherently exhibit more consistent patterns and reduced variability (Fu et al., 2024; Kirchenbauer et al., 2023), the acceleration ratio experiences a significant upswing. This acceleration difference between natural language and code validates our method's linguistic rationality and adaptability based on content predictability. Further analysis of LUD's outputs indicates that tokens decoded concurrently by LUD invariably present coherent and linguistically meaningful units, validating our intuition that LLMs can identify these units effectively.

The elegance of our method is its deployment simplicity. Instead of resorting to complex architectural modifications, we take advantage of the model's inherent ability to generate new data based on the original dataset. The generated new data is used for continual training of parallel decoding to optimize the model's generation speed and ensure straightforward implementation.

Our contributions can be summarized as follows.

- We uncover a naturally emerging pattern within LLMs, highlighting the consistent high-confidence prediction of certain spans of tokens, which we term as "lexical units".
- We present Lexical Unit Decoding (LUD), an linguistically-adaptive, data-centric methodology that ensures lossless acceleration in decoding and seamless integration without intricate modifications of the model's architecture.
- We conducted an in-depth analysis on common issues in parallel decoding from a new perspective and discussed potential avenues for future research.

2. Related Work

The scaling laws of LLMs have intensified the pursuit of faster inference, sparking a wave of innovation.

The initial advancements in accelerated decoding were most prominent in machine translation, highlighted by the emergence of non-autoregressive transformers (Gu et al., 2017). These models, often reliant on encoder-decoder frameworks, introduced latent variables to enable parallel predictions (Kaiser et al., 2018; Qian et al., 2021; Cheng and Zhang, 2022; Xiao et al., 2023). However this approach comes at the cost of quality, thereby limiting their universality in decoder-only LLMs.

Concurrently, more general acceleration methods like distillation (Hinton et al., 2015), sparcification (Jaszczur et al., 2021), quantization (Hubara et al., 2017), and architectural modifications (So et al., 2021) have been explored. These strategies, centered around computational optimization, seek to expedite inference with minimal performance compromise. However, they often entail an unavoidable quality reduction.

Adaptive computation has emerged as a potent strategy, with established methods like "early exits" leading the charge (Schuster et al., 2021; Bapna et al., 2020; Schwartz et al., 2020; Elbayad et al., 2020). Those models introduce dynamic computational depth adjustment, allowing predictions to be made earlier in the process for simpler cases,

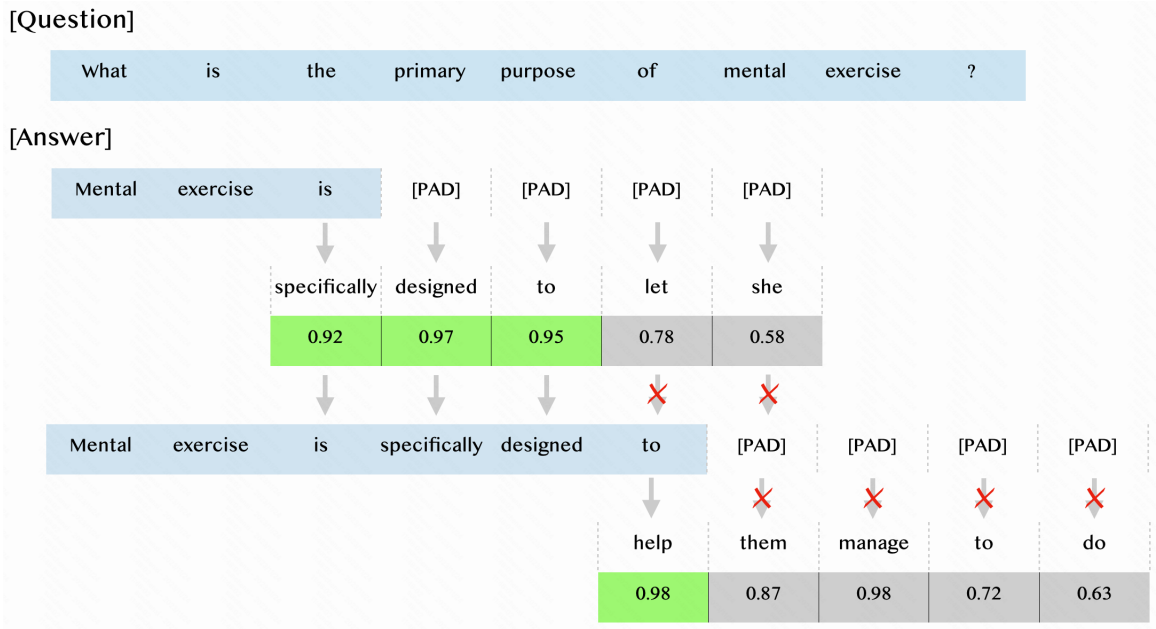

Figure 2: Illustration of the lexical unit decoding procedure. In the decoding process, we look ahead $k = 5$ tokens by appending $k - 1 = 4$ [] tokens and retrieving the last k predicted tokens with their probabilities. However, we only accept consecutive tokens with probabilities larger than $\alpha = 0.9$.

thereby enhancing inference speed (Scardapane et al., 2020).

The landscape of local-non-autoregressive adaptive computation has expanded recently. Methods aiming to decode multiple tokens simultaneously have gained traction. Innovations like separate decoding heads with tree attention mechanisms (Cai et al., 2023), and direct input-to-output segment copying (Sun et al., 2021) demonstrate the diversity of approaches. Speculative Decoding (Leviathan et al., 2023) and Blockwise Parallel Decoding (Stern et al., 2018) stand out by offering lossless acceleration, cleverly navigating the quality-speed trade-off through the use of auxiliary models. Yet, those advancements also introduce new challenges, including increased computational overhead and the complexity of integrating additional system components.

3. Lexical Unit Decoding

Our proposed methodology aims for efficient decoding by leveraging coherent linguistic chunks, termed as "lexical units". This efficiency is achieved by allowing models to predict up to k continuous tokens at once as a lexical unit, rather than being confined to a single next-token prediction. The detailed approach unfolds as follows.

3.1. Inference

Unlike traditional models that predict one token at a time, our model, equipped with the knowledge of lexical units, attempts to predict multiple tokens in a single step, thereby accelerating the decoding process.

Look-Ahead Prediction Given a context, our model doesn't restrict itself to predicting just the immediate next token. Instead, it ambitiously casts a look-ahead window of a fixed length k, aiming to predict the next k tokens in one sweep. This block of tokens, denoted as $x_{t:t+k}$, is predicted as:

$$x_{t:t+k} \sim P(x_{t:t+k}|\text{context}, \text{pos}(x_{t-1:t+k-1})) \quad (1)$$

The probability distribution of the block $x_{t:t+k}$ is conditioned on the preceding context and the positional information of the tokens before the target token within this block. In practice, we append $k-1$ [PAD] tokens to the context input and extract the last k logits to compute the probability distribution.

Adaptive Span Acceptance While the model attempts to predict k tokens, not all predictions might be of high confidence. Thus, we introduce a mechanism to selectively accept tokens from this prediction. Specifically, only the first l tokens that consistently maintain probabilities above a specified threshold β are accepted as a span S. l is estimated as follows:

$$l = \max(1, m) \quad (2)$$

where m is the largest integer such that:

$$\forall i \in [t, t+m] : P(x_i) \geq \beta,$$
$$P(x_{t+m+1}) < \beta.$$

Algorithm 1 Data Generation Process

Require: Dataset D, Model M, Threshold α
Ensure: Reconfigured training data D'

function IDENTIFY($logits$, α)

function RECONFIGURE($item$, $lexicalUnits$)

$M_{\mathrm{FT}} \leftarrow$ Fine-tune M on D

$\bar{D} \leftarrow []$
for each $item$ in D **do**
    $logits \leftarrow M_{\mathrm{FT}}(item)$
    $lexicalUnits \leftarrow$ IDENTIFY($logits$, α)
    $data \leftarrow$ RECONFIGURE($item$, $lexicalUnits$)
    Extend $\bar{D}$ with $data$
end for
$D' \leftarrow D + \bar{D}$
return D'

Token Repetition Reduction Parallel decoding method suffers more from the token repetition problem (Gu et al., 2017) since the prefixing token is not ready for reference like the auto-regressive one. To address this issue, we implemented a straightforward method: for the decoded k tokens, we check for consecutive token ids or instances where x_{i-1} ends with x_i. If detected, the model immediately halts acceptance of new tokens.

A notable feature of our inference strategy is its adaptability. If the first token from the look-ahead window does not meet the confidence threshold, the model reverts to accepting only the first token. This ensures that in scenarios where the model isn't confident about predicting a lexical unit, it equals to the conventional auto-regressive decoding strategy, ensuring robustness across diverse linguistic contexts. The inference process is illustrated in Figure 2.

3.2. Data Generation

Lexical Unit Identification Central to the methodology is the identification of "lexical units". These are continuous sequences of tokens that captures semantically and linguistically coherent constructs. It has been observed that pre-trained language models, especially when fine-tuned on the target dataset, tend to predict tokens within these units with remarkable confidence.

To harness this observation, we establish an identification criterion. Specifically, a continuous span of tokens is deemed a lexical unit if the prediction probability of each token within this span surpasses a predefined threshold α. This threshold serves as a confidence measure, ensuring that the identified spans actually represent coherent linguistic constructs.

Formally, given a span of tokens $x_{t:t+l}$, we define it as a lexical unit if the following constraints are satisfied:

1. $P(x_i) \geq \alpha$ for all $i \in [t, t+l]$, where $P(x_i)$ is the prediction probability of the i-th token and α is the predefined threshold.

2. $P(x_{t-1}) < \alpha$ and $P(x_{t+l+1}) < \alpha$, ensuring that the tokens immediately before and after the sequence have prediction probabilities lower than α.

To facilitate the model to recognize these lexical units, we first fine-tune the original model M on the target dataset D. This auto-regressive training refines the model into M_{FT}. During the forward pass with M_{FT} on D, by comparing probability of the ground truth label against the threshold α, we can effectively identify spans of tokens that the model predicts with high confidence. These high-confidence spans are then designated as lexical units. This process is elaborated in Algorithm 1.

Data Construction Upon identifying the lexical units, we can proceed with data reconfiguration for continual training. First, as shown in Figure 3, given a single piece of data, multiple lexical units can be identified. Note that lexical units can contain multiple tokens or only a single token with coherent semantic meaning.

For every detected lexical unit, a new training instance is instantiated by replacing the tokens inside with trainable $[\mathrm{PAD}]$ tokens. This utilization of the trainable $[\mathrm{PAD}]$ token over other potential masking strategies ensures the integrity of positional information, which is critical for the model's functioning. Further more, for the new training instance, the context tokens before it are left unchanged to provide complete context information, and the loss is only calculated for the tokens within the lexical unit. In this way, we can guarantee that the loss of each token is still calculated only once even though a single sequence is split into multiple ones.

The streamlined process of data generation, from lexical unit identification to data reconfiguration, is demonstrated in Figure 3. A more detailed illustration can be found in Algorithm 1. Note that the generated dataset $\bar{D}$ is mixed with the original dataset D as the final reconfigured dataset D'.

3.3. Training

The core of our methodology lies in its data-centric approach. By leveraging the reconfigured dataset D', we ensure that the model is proficient at both recognizing and generating lexical units during the inference phase. This is achieved without deviating

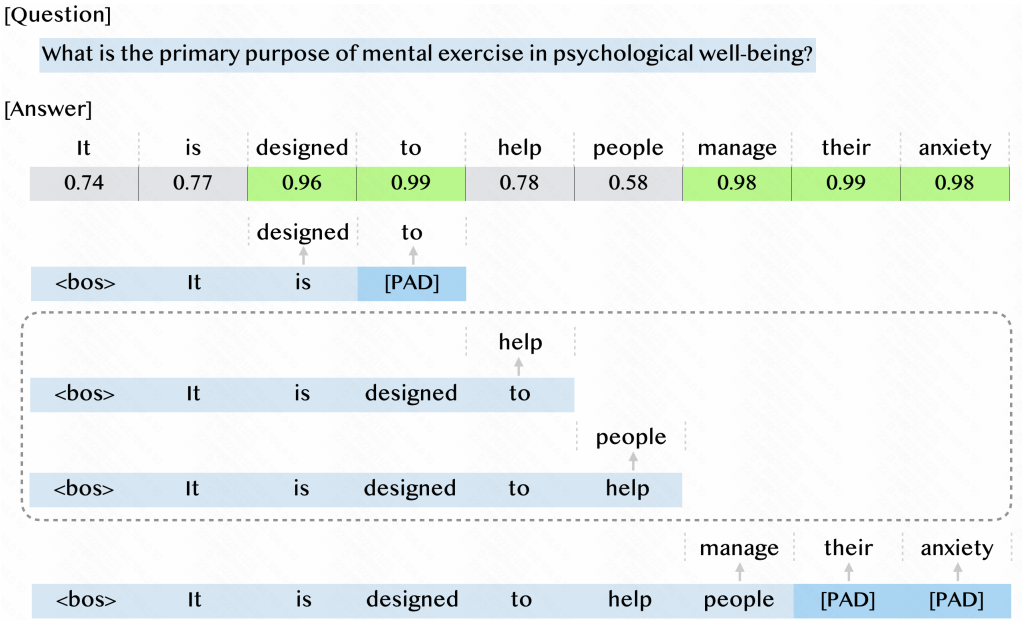

Figure 3: Visualization of the streamlined Data Generation process. Given a sequence of tokens and their corresponding probabilities, lexical units are segmented based on a threshold $\alpha = 0.9$. Probabilities above the threshold are highlighted in green. Lexical units can consist of either a single token or multiple tokens. Multi-token lexical units are appended with $[\mathrm{PAD}]$ tokens to enable the training of parallel decoding. For individual tokens with lower prediction confidence, the model falls back to the auto-regressive training manner to maintain output quality. In practice, the second and third instances with a lexical unit of length 1 are combined as one, which is basically the same to a standard auto-regressive training example.

from conventional training procedures, highlighting the pivotal role of our data generation process.

Standard Token Prediction Tokens that are not encapsulated within any lexical units are trained in the traditional language modeling manner. In this scenario, the model predicts the next token in the sequence based on the preceding tokens, adhering to the standard auto-regressive nature of LLMs.

Lexical Unit Token Prediction When it comes to Lexical Unit Tokens, the prediction mechanism deviates slightly. The model is trained to predict these tokens by considering not just the complete information from the tokens preceding the lexical unit, but also the positional information, which is actually embedded in the trainable $[\mathrm{PAD}]$ tokens, within the lexical unit. Note that unlike setting a look-ahead window with a fixed length k during inference, the length of the identified lexical units can be varied during training. This is formulated in Equation (3).

$$x_t \sim P(x_t | x_{1:i}, \mathrm{pos}(x_{i:t-1})) \tag{3}$$

where x_t denotes the Lexical Unit Tokens to be predicted and i denotes the initial position of the lexical unit to which x_t pertains.. Similar to inference, its prediction is conditioned on Context Tokens $x_1, x_2, ..., x_i$ preceding the lexical unit, and the positional information $\mathrm{pos}(x_{i:t-1})$ of the Prefixing Tokens within the lexical unit. This formulation underscores that, although the training procedure remains unchanged, the underlying training dynamics differs based on a token's association with a lexical unit.

As mentioned before, in practice, token ids of the Lexical Unit Tokens are replaced with $\mathrm{pad\_token\_id}$, and labels for Context Tokens are set to -100 to exclude them from loss calculation. Note that for Lexical Unit Tokens, the model is trained to predict the original tokens rather than $[\mathrm{PAD}]$ token. $[\mathrm{PAD}]$ tokens are to provide the positional context.

4. Experiments

In this section, we detail the experiments conducted to evaluate the efficacy of our method. Despite the existence of numerous tasks across various domains (Moradshahi et al., 2023; Liu et al., 2023; Ge et al., 2021), our evaluation specifically concentrates on text and code generation tasks to thoroughly validate the adaptive acceleration capabilities of our method.

4.1. Experimental Setup

We used LLaMA-13B as the LLM for both experiments. Supervised Fine-tuning (SFT) is conducted

with identical hyper-parameters and prompt templates as specified in Alpaca (Taori et al., 2023) and Code Alpaca (Chaudhary, 2023). During data generation for continual training, we set α to 0.85. The fine-tuned models then undergo continual training on the generated data with a batch size of 768 and a learning rate of $3e-5$ for 3 and 5 epochs for text and code respectively. During inference, we vary β from 0.75 to 1.0 while fix the size of the look-ahead window, k, as 10.

4.1.1. Text Generation

Dataset We used the dataset released along with Alpaca (Taori et al., 2023) as the training dataset and the dataset released by Wang et al. (2023) as the test set. This high-quality instruction-following training dataset comprises 52,000 unique instructions generated using the self-instruction technique proposed by Wang et al. (2023). Specifically, 175 manually-written tasks are utilized as seed tasks, which serve as in-context learning examples to prompt text-davinci-003 for generating more diverse and high-quality data items. The test set comprises 252 instructions across various domains. Following Chen et al. (2023), a pairwise comparison between the generations from the fine-tuned model and its LUD version was executed. The answers from both models were fed as a pair with a prompt template to obtain two scores indicating the quality of answers respectively. To mitigate the effect of order when prompting GPT-4 for scoring, each pair was scored twice in exchanged order, and the mean score was computed as the final result.

Quality Metric The objective is to compare the quality loss before and after the LUD training process. Each scored pair can yield three possible outcomes - higher, same, or lower. We calculate the number of examples for each result and compute the quality metric as follows:

$$R_{\text{quality}} = \frac{g+s}{b+s}, \tag{4}$$

where g, s and b denote the number of examples with scores higher than, same as, and lower than those generations from Alpaca, respectively.

4.1.2. Code Generation

Dataset For code generation, we utilized the code instruction-following dataset released with Code Alpaca (Chaudhary, 2023) as the training set and HumanEval (Chen et al., 2021) for evaluation. This training dataset contains 20,000 coding instructions along with Python solutions, generated using the self-instruction technique, analogous to the Alpaca text dataset. HumanEval (Chen et al., 2021) encompasses 164 hand-written coding problems, each with an average of 7.7 unit tests.

Quality Metric The Pass@1 (Chen et al., 2021) is adopted as the basic metric to evaluate the quality of code generation. We compare LUD with the auto-regressive baseline via calculating

$$R_{\text{quality}} = \frac{\text{Pass@1}_{\text{LUD}}}{\text{Pass@1}_{\text{AT}}} \tag{5}$$

4.2. Acceleration Metrics

Forward Compression Ratio (FCR) The Forward Compression Ratio (FCR) quantifies the efficiency gains achieved by our Lexical Unit Decoding (LUD) method. In a conventional auto-regressive setting, each token generated necessitates a forward calculation, making the number of tokens generated equal to the number of forward calculations. However, LUD's capability to decode multiple tokens in a single forward pass introduces a disparity between these two numbers. FCR is defined as:

$$R_{\text{FCR}} = \frac{N_{\text{tokens}} - N_{\text{lexical}}}{N_{\text{tokens}}} \tag{6}$$

A higher FCR value signifies greater computational efficiency improvement, as it indicates that fewer forward calculations are required to produce an equivalent number of tokens.

Wall-time Acceleration Ratio (WAR) While the FCR offers a theoretical perspective on efficiency, the Wall-time Acceleration Ratio (WAR) provides a more pragmatic view. It measures the acceleration in terms of actual computation time, factoring in real-world considerations including hardware efficiency, potential parallelization, and other computational overheads. WAR is given by:

$$R_{\text{WAR}} = \frac{t_{\text{AT}} - t_{\text{LUD}}}{t_{\text{AT}}} \tag{7}$$

where, t_{AT} represents the average time taken to generate a single token using the auto-regressive approach, while t_{LUD} denotes the corresponding time for the LUD method. In practice, we measure the time consumption of solely the generation loop excluding data loading while setting batch size to 1. The total time consumed is then divided by the number of generated tokens to get a precise estimate of the average time per token.

4.3. Results

The main results are shown in Table 1. LUD substantially reduces decoding time while maintaining generation quality - 33% speed up on natural language generation with no quality loss. And a 30%

	R_{quality}	R_{FCR}	R_{WAR}	β
Text	100%	33.24%	33.68%	0.9
Code	97%	29.83%	29.77%	0.99987

Table 1: Main experimental results.

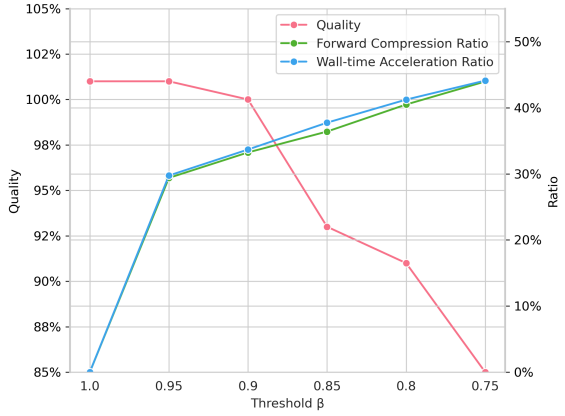

Figure 4: Quality and Acceleration curves of *text* generation

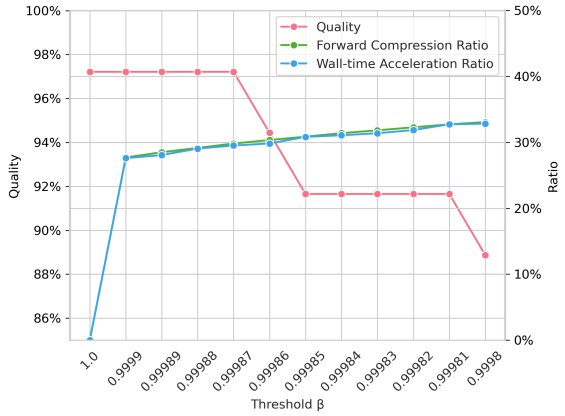

Figure 5: Quality and Acceleration curves of *code* generation

speed up on code generation is observed with a negligible quality loss of 3%.

To provide a clear and direct comparison, we also present the quality metrics alongside the acceleration curves across different β values. These are illustrated separately for text and code in Figure 4 and Figure 5, respectively.

FCR and WAR The efficiency of the Lexical Unit Decoding (LUD) approach is evident when examining both the FCR and WAR metrics. As we decrease the parameter β, we observe a consistent increase in both metrics . This indicates that the model is more aggressive in accepting multiple tokens at once when the confidence threshold is lower, leading to higher acceleration. However, this efficiency achieved by a smaller β value can cause quality degeneration.

Quality Loss The quality of the generated content, both for text and code, shows a declining trend as we push for more acceleration by decreasing β. This degradation in quality underscores the trade-off between acceleration and quality. As we push the model to be more aggressive in its predictions, the chances of making errors increase, leading to a drop in the quality of the generated content. We do a deeper dive into the generation process in Section 5.

Trade-Off and Optimal β value It's apparent there's some trade-off between the acceleration ratio and quality loss when varying β. However, we did notice that the generation quality can be maintained when β is above a specific value, within which decreasing β can lead to faster decoding. This means there is a "sweet pot" of β that achieves fastest lossless decoding. While both text and code data reconfiguration is performed with α set to 0.85, the best β values are 0.9 and 0.99987 respectively, which doesn't align with α strictly.

Text vs. Code Generation Another clear observation is with the same β value, codes generation can be accelerated faster than text generation. The disparity between code and text generation is intriguing. One possible explanation is code sequences are viewed as a kind of low-entropy sequences while text sequences have much higher entropy (Kirchenbauer et al., 2023). Codes often follow specific patterns and structures, making them more predictable. This predictability might allow the model to confidently generate larger chunks of code tokens at once, leading to faster decoding acceleration. This observation validates that our method can accelerate the decoding process adaptively. Note that models for natural language generation and code generation are fine-tuned only with in-domain dataset. However, we argue that using a more balanced dataset, a sweet pot can still be achieved for a general large language model.

However, the quality of code generation degrades much faster than text as β decreases. This could be due to the fact that even minor errors in code can render it non-functional, whereas text might still be understandable even with minor inaccuracies. Noticeably, code decoding is also much more sensitive to β. We varied the beta value carefully and found that 0.99987 can accelerate the decoding speed by 30% with 3% quality loss. We suspect that the code's attribute of being more predictable makes the model confident on its decoding. But further experiments should be conducted to understand what's happening inside the model. We leave it to our future work.

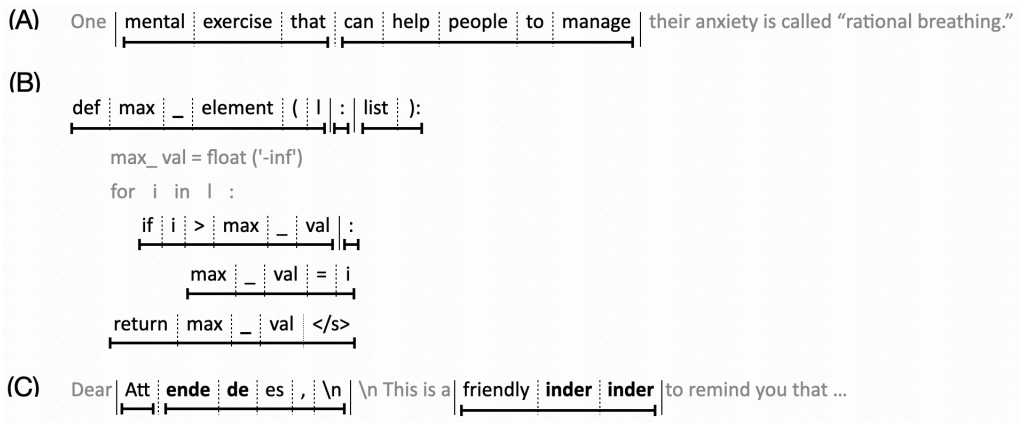

Figure 6: Examples generated with LUD. Example (A) and (B) are generated with $\beta = 0.9$. Sequence (C) is generated using a smaller $\beta = 0.85$ to expose the token repetition issue.

5. Analysis

Our results provide a basic understanding of the Lexical Unit Decoding (LUD) approach. To further explain its behavior and implications, we delve deeper into the generation process. Given the intricate nature of those results, we perform a hands-on, empirical analysis. The codes for generating the visualization of the generation process will be publicly available.

5.1. Coherence of Parallel Decoded Tokens

A qualitative analysis of the tokens decoded in parallel offers a window into the model's perception of coherent linguistic constructs and grounds our definition of “lexical units”. Our detailed inspection reveals patterns that are expected in some aspects and surprising in others. Several iconic examples are illustrated in Figure 6.

Example (A) showcases that LUD can indeed generate coherent constructs. We also provide example (B) as an instance of code generation. It's apparent that tokens are generated in much larger chunks leading to faster acceleration. Moreover, we have found that while tokens within a line can be highly paralleled, it's almost impossible to parallelly decode multiple tokens spanning two lines of code, except \n and \t. This finding echoes Zhu et al. (2023), where they state that the first token in a line of code is more difficult to predict than others.

5.2. Distribution of the Number of Parallel Decoded Tokens

Figure 7 offers an overview of the distribution of the number of accepted tokens with the window size $k = 10$. The statistics is collected with $\beta = 0.9$ for text and $\beta = 0.99987$ for code as before.

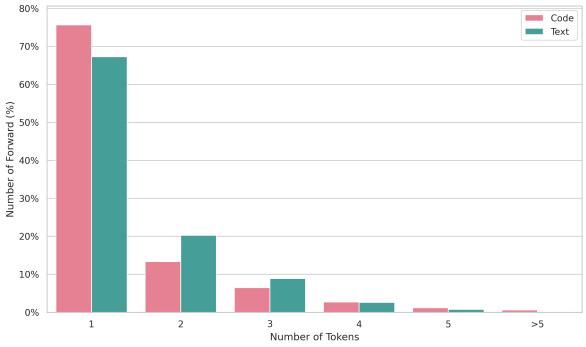

Figure 7: Distribution of the number of accepted tokens

5.3. Token Repetition

Parallel decoding method suffers more from the token repetition problem (Gu et al., 2017) since the prefixing token is not ready for reference like the auto-regressive one. For consecutive token pairs $[x_{i-1}, x_i]$, we have occasionally observed token repetition during LUD's decoding process. As depicted in Figure 6 (C), repetition can manifest in various ways, such as the end of x_i matching the ends of x_{i-1} (partial repetition), or x_i being identical to x_{i-1} (identical repetition). While both repetition phenomena are rare and can be largely avoided with a larger β, empirical observation indicates that partial repetition happens much more frequent than identical repetition.

Even though, our method halts token acceptance upon detecting a repeated token, effectively reducing unnecessary repetition. However, it still permits the generation of essential repeated tokens in subsequent forward passes, recognizing when such repetitions are indeed necessary. Despite its simplicity, this approach proves effective, mitigating repetition while only marginally reducing decoding speed by 2-3% on average.

6. Discussion

While our method effectively accelerate the decoding process, there are still several points worth studying:

More Advanced Lexical Unit Identification Method In this paper, lexical units are identified based on the prediction probability of each token. While effective, more advanced identification methods can also be further explored. For example, one can take the attention state (Ren and Xiong, 2023) or find patterns in the neuron activation state (Zou et al., 2023) into account, then try to establish the relationship between them and meaningful lexical units, which will be used to reconfigure the dataset for parallel decoding training. Moreover, as the model undergoes continual training, its perception of lexical units might also shift so the static pre-generated data might not always be optimal. This brings forth the potential "on-the-fly" data generation. By dynamically generating training examples aligned with the model's current understanding, we can ensure a more harmonized training process.

Lexical Unit Decoding During Pre-training In this work we focus on the adaption of finetuned model into parallel decoding mode via a lightweight training. It would be interesting to pre-build the lexical unit decoding capability into LLM during the pre-training procedure and we leave it as a future work.

7. Conclusion

In this study, we have introduced and evaluated the Lexical Unit Decoding (LUD), a novel method designed to bolster the decoding efficiency of sequence generation models. Our findings underscore the efficacy of LUD to accelerate the decoding process without sacrificing generation quality - 33% acceleration on text generation and 30% acceleration on code generation. We also analyze the intriguing patterns in the tokens decoded in parallel, providing insights into the model's perception of coherent linguistic constructs.

8. Acknowledgements

The present research was partially supported by the Key Research and Development Program of Yunnan Province (Grant No. 202203AA080004). We would like to thank the anonymous reviewers for their insightful comments.

9. Bibliographical References

Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, Eric Chu, Jonathan H. Clark, Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, Sebastian Ruder, Yi Tay, Kefan Xiao, Yuanzhong Xu, Yujing Zhang, Gustavo Hernández Ábrego, Junwhan Ahn, Jacob Austin, Paul Barham, Jan A. Botha, James Bradbury, Siddhartha Brahma, Kevin Brooks, Michele Catasta, Yong Cheng, Colin Cherry, Christopher A. Choquette-Choo, Aakanksha Chowdhery, Clément Crepy, Shachi Dave, Mostafa Dehghani, Sunipa Dev, Jacob Devlin, Mark Díaz, Nan Du, Ethan Dyer, Vladimir Feinberg, Fangxiaoyu Feng, Vlad Fienber, Markus Freitag, Xavier Garcia, Sebastian Gehrmann, Lucas Gonzalez, and et al. 2023. Palm 2 technical report. *CoRR*, abs/2305.10403.

Ankur Bapna, Naveen Arivazhagan, and Orhan Firat. 2020. Controlling computation versus quality for neural sequence models. *CoRR*, abs/2002.07106.

Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. 2000. A neural probabilistic language model. In *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*, pages 932–938. MIT Press.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *CoRR*, abs/2005.14165.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, and Tri Dao. 2023. Medusa: Simple framework for accelerating llm generation with multiple decoding heads. https://github.com/FasterDecoding/Medusa.

Sahil Chaudhary. 2023. Code alpaca: An instruction-following llama model for code generation. https://github.com/sahil280114/codealpaca.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Zhihong Chen, Feng Jiang, Junying Chen, Tiannan Wang, Fei Yu, Guiming Chen, Hongbo Zhang, Juhao Liang, Chen Zhang, Zhiyi Zhang, Jianquan Li, Xiang Wan, Benyou Wang, and Haizhou Li. 2023. Phoenix: Democratizing chatgpt across languages. *CoRR*, abs/2304.10453.

Hao Cheng and Zhihua Zhang. 2022. MR-P: A Parallel Decoding Algorithm for Iterative Refinement Non-Autoregressive Translation. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 285–296, Dublin, Ireland. Association for Computational Linguistics.

Aidan Clark, Diego de Las Casas, Aurelia Guy, Arthur Mensch, Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake A. Hechtman, Trevor Cai, Sebastian Borgeaud, George van den Driessche, Eliza Rutherford, Tom Hennigan, Matthew J. Johnson, Albin Cassirer, Chris Jones, Elena Buchatskaya, David Budden, Laurent Sifre, Simon Osindero, Oriol Vinyals, Marc'Aurelio Ranzato, Jack W. Rae, Erich Elsen, Koray Kavukcuoglu, and Karen Simonyan. 2022. Unified scaling laws for routed language models. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 4057–4086. PMLR.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. 2020. Depth-adaptive transformer. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Yu Fu, Deyi Xiong, and Yue Dong. 2024. Watermarking conditional text generation for ai detection: Unveiling challenges and a semantic-aware watermark remedy.

Huibin Ge, Chenxi Sun, Deyi Xiong, and Qun Liu. 2021. Chinese WPLC: A Chinese dataset for evaluating pretrained language models on word prediction given long-range context. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3770–3778, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. 2017. Non-autoregressive neural machine translation. *CoRR*, abs/1711.02281.

Danny Hernandez, Jared Kaplan, Tom Henighan, and Sam McCandlish. 2021. Scaling laws for transfer. *CoRR*, abs/2102.01293.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. 2022. Training compute-optimal large language models. *CoRR*, abs/2203.15556.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *J. Mach. Learn. Res.*, 18:187:1–187:30.

Sebastian Jaszczur, Aakanksha Chowdhery, Afroz Mohiuddin, Lukasz Kaiser, Wojciech Gajewski, Henryk Michalewski, and Jonni Kanerva. 2021. Sparse is enough in scaling transformers. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 9895–9907.

Lukasz Kaiser, Samy Bengio, Aurko Roy, Ashish Vaswani, Niki Parmar, Jakob Uszkoreit, and Noam Shazeer. 2018. Fast decoding in sequence models using discrete latent variables. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15,*

2018, volume 80 of *Proceedings of Machine Learning Research*, pages 2395–2404. PMLR.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *CoRR*, abs/2001.08361.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. 2023. A watermark for large language models.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR.

Chuang Liu, Junzhuo Li, and Deyi Xiong. 2023. TabCQA: A tabular conversational question answering dataset on financial reports. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 5: Industry Track)*, pages 196–207, Toronto, Canada. Association for Computational Linguistics.

Mehrad Moradshahi, Tianhao Shen, Kalika Bali, Monojit Choudhury, Gael de Chalendar, Anmol Goel, Sungkyun Kim, Prashant Kodali, Ponnurangam Kumaraguru, Nasredine Semmar, Sina Semnani, Jiwon Seo, Vivek Seshadri, Manish Shrivastava, Michael Sun, Aditya Yadavalli, Chaobin You, Deyi Xiong, and Monica Lam. 2023. X-RiSAWOZ: High-quality end-to-end multilingual dialogue datasets and few-shot agents. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 2773–2794, Toronto, Canada. Association for Computational Linguistics.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

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. ArXiv:2008.07905 [cs].

Yuqi Ren and Deyi Xiong. 2023. HuaSLIM: Human attention motivated shortcut learning identification and mitigation for large language models. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 12350–12365, Toronto, Canada. Association for Computational Linguistics.

Simone Scardapane, Michele Scarpiniti, Enzo Baccarelli, and Aurelio Uncini. 2020. Why should we add early exits to neural networks? *Cogn. Comput.*, 12(5):954–966.

Tal Schuster, Adam Fisch, Tommi S. Jaakkola, and Regina Barzilay. 2021. Consistent accelerated inference via confident adaptive transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 4962–4979. Association for Computational Linguistics.

Roy Schwartz, Gabriel Stanovsky, Swabha Swayamdipta, Jesse Dodge, and Noah A. Smith. 2020. The right tool for the job: Matching model and instance complexities. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 6640–6651. Association for Computational Linguistics.

Claude E. Shannon. 1948. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27(3):379–423.

David R. So, Wojciech Manke, Hanxiao Liu, Zihang Dai, Noam Shazeer, and Quoc V. Le. 2021. Primer: Searching for efficient transformers for language modeling. *CoRR*, abs/2109.08668.

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.

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, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 5937–5947. Association for Computational Linguistics.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez,

Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Svetlana Vetchinnikova, Alena Konina, Nitin Williams, Nina Mikušová, and Anna Mauranen. 2023. Chunking up speech in real time: linguistic predictors and cognitive constraints. *Language and Cognition*, 15(3):453–479.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 13484–13508. Association for Computational Linguistics.

Yisheng Xiao, Lijun Wu, Junliang Guo, Juntao Li, Min Zhang, Tao Qin, and Tie-yan Liu. 2023. A Survey on Non-Autoregressive Generation for Neural Machine Translation and Beyond. ArXiv:2204.09269 [cs].

Yuqi Zhu, Jia Allen Li, Ge Li, Yunfei Zhao, Jia Li, Zhi Jin, and Hong Mei. 2023. Improving code generation by dynamic temperature sampling. *CoRR*, abs/2309.02772.

Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, Shashwat Goel, Nathaniel Li, Michael J. Byun, Zifan Wang, Alex Mallen, Steven Basart, Sanmi Koyejo, Dawn Song, Matt Fredrikson, J. Zico Kolter, and Dan Hendrycks. 2023. Representation engineering: A top-down approach to ai transparency.