LLM as Effective Streaming Processor: Bridging Streaming-Batch Mismatches with Group Position Encoding

Junlong Tong[1,2,3], Jinlan Fu[4], Zixuan Lin[5], Yingqi Fan[3], Anhao Zhao[3], Hui Su[6], Xiaoyu Shen[2,3]*

[1]Shanghai Jiao Tong University
[2]Ningbo Key Laboratory of Spatial Intelligence and Digital Derivative
[3]Institute of Digital Twin, EIT [4]National University of Singapore
[5]University of Science and Technology of China [6]Meituan Inc.
jl-tong@sjtu.edu.cn xyshen@eitech.edu.cn

Abstract

Large Language Models (LLMs) are primarily designed for batch processing. Existing methods for adapting LLMs to streaming rely either on expensive re-encoding or specialized architectures with limited scalability. This work identifies three key mismatches in adapting batch-oriented LLMs to streaming: (1) input-attention, (2) output-attention, and (3) position-ID mismatches. While it is commonly assumed that the latter two mismatches require frequent re-encoding, our analysis reveals that only the input-attention mismatch significantly impacts performance, indicating re-encoding outputs is largely unnecessary. To better understand this discrepancy with the common assumption, we provide the first comprehensive analysis of the impact of position encoding on LLMs in streaming, showing that preserving relative positions within source and target contexts is more critical than maintaining absolute order. Motivated by the above analysis, we introduce a group position encoding paradigm built on batch architectures to enhance consistency between streaming and batch modes. Extensive experiments on cross-lingual and cross-modal tasks demonstrate that our method outperforms existing approaches. Our method requires no architectural modifications, exhibits strong generalization in both streaming and batch modes. The code is available at repository https://github.com/EIT-NLP/StreamingLLM.

1 Introduction

Large language models (LLMs) have revolutionized a multitude of tasks (Zhang et al., 2023b; Liu et al., 2024; Chu et al., 2023; Kojima et al., 2022; Kocmi and Federmann, 2023). However, research on LLMs has largely focused on *batch-processing*, where the entire input is processed at once (Zhao et al., 2023). In contrast, human cognition operates incrementally, interpreting information as it arrives—a capability essential for real-time decision-making, interactive dialogue, and other latency-sensitive applications (Gonzalez et al., 2003; Altmann and Mirković, 2009). *Bridging this gap between batch-oriented LLMs and streaming-aware processing* is vital for unlocking their potential in dynamic, real-world scenarios.

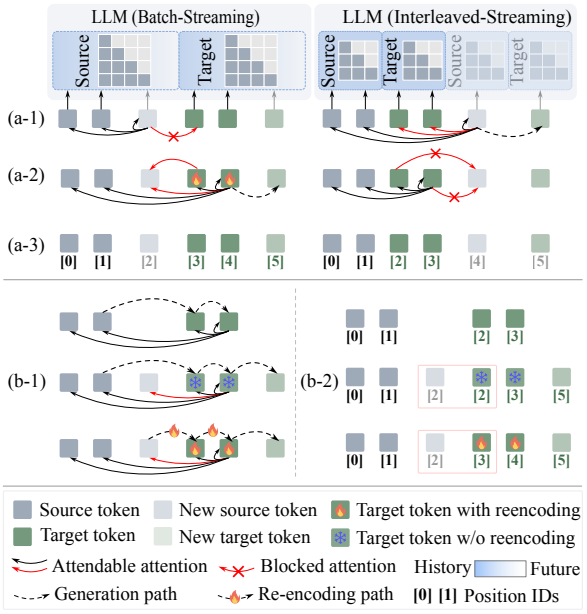


Figure 1: Two streaming paradigms of LLMs: (a) Batch-streaming simulates batch-processing, while interleaved-streaming encodes streaming data in arrival order. (a-1) *Input-Attention Mismatch*: Whether the source tokens can attend to the target tokens. (a-2) *Output-Attention Mismatch*: Whether the target tokens can attend to the new source token. (a-3) *Position-ID Mismatch*: Whether the position IDs reflect the actual token order. (b) Batch-streaming relies on (b-1) KV cache re-encoding and (b-2) position re-encoding to simulate batch-processing.

A naive strategy to adapt LLMs for streaming involves iteratively re-encoding both new inputs and prior outputs with each incoming data segment (Agostinelli et al., 2024; Wang et al., 2024; Guo et al., 2024b; Koshkin et al., 2024), as illustrated in Figure 1(b). While this *batch-streaming* paradigm

*Corresponding author

Findings of the Association for Computational Linguistics: ACL 2025, pages 23497–23517

preserves compatibility with batch-processing architectures, it introduces prohibitive computational costs. Existing efforts to optimize LLMs for streaming data typically fall into two categories: (1) *Directly encoding streaming data in arrival order* (Du et al., 2024; Yang et al., 2024), an *interleaved-streaming* paradigm, which introduces structural mismatches with batch-processing setups used in pre-training and degrades performance; (2) *Designing entirely new architectures tailored to the streaming mode* (Guo et al., 2024a; Tsunoo et al., 2024; Chen et al., 2024), which is costly, lacks scalability, and fails to fully leverage pre-trained LLM capabilities. Furthermore, existing methods lack rigorous analysis of the fundamental discrepancies between batch and streaming processing modes.

This work tackles these limitations by identifying three key mismatches in adapting batch-oriented LLMs to streaming, as shown in Figure 1:

- **Input-Attention Mismatch**:Batch-streaming confines input tokens to attending only prior inputs, whereas interleaved-streaming permits attention to previously decoded outputs.
- **Output-Attention Mismatch**:Batch-streaming allows decoded output tokens to attending to all received input tokens by *KV cache re-encoding*, while interleaved-streaming mode limits each output token's attention to the subset of inputs available at decoding time.
- **Position-ID Mismatch**: Batch-streaming relies on *position re-encoding*, assigning contiguous position IDs to inputs followed by outputs, whereas interleaved-streaming processes alternate between inputs and outputs incrementally, resulting in discontinuous positional ids that disrupt sequential coherence.

Building on the identification of these mismatches, we systematically studied their effects on LLM performance. Our analysis revealed that input-attention mismatch does affect streaming model performance. In contrast, output-attention and position-ID mismatches have negligible effects. A common assumption is that streaming models require re-encoding of previously generated content to mitigate *token position inconsistencies* arising from the incremental nature of streaming setting (Raffel et al., 2024; Guo et al., 2024a; He et al., 2024), as shown in Figure 1(b). However, our empirical findings do not support this hypothesis. Instead, we observe that re-encoding the output is not necessary [1]. This discrepancy with the common assumption raises a fundamental question: **How does position encoding impact LLMs in streaming scenarios? And how should we design appropriate position encoding for streaming LLMs?**

Existing research on positional encoding in LLMs has largely focused on static scenarios (Likhomanenko et al., 2021; Haviv et al., 2022; Kazemnejad et al., 2024), while its role in streaming scenarios remains underexplored. We conducted a more in-depth analysis to further explore the impact of position encoding on streaming models. Experimental results reveal that the absolute positional order of tokens has a negligible effect on model performance in streaming tasks. However, maintaining the internal relative order within the source and target sequences is significantly more important. Based on the findings, we propose a grouped position encoding streaming paradigm built on batch architectures(*group-streaming*), which groups input and output position ids to enable more consistent processing with the batch model. This strategy is not only computationally efficient but also generalizable across different tasks and model architectures. We validated its effectiveness on cross-lingual (machine translation) and cross-modal (automatic speech recognition) tasks, demonstrating that it significantly outperforms existing solutions, including LLMs with more complex streaming-optimized architectures.

The main contributions of this study can be summarized as: (1) We systematically analyze the mismatches between batch and streaming processing in LLMs, providing deep insights into key factors affecting their adaptation to streaming. Contrary to mainstream assumptions, our experiments reveal that position disorder is not the primary factor affecting LLM streaming performance. (2) We conduct the first comprehensive study on the impact of position encoding in streaming scenarios, demonstrating that absolute positional order is unnecessary, while maintaining relative order within source and target contexts is more critical. (3) We introduce a *group streaming paradigm* for streaming LLMs. This method imposes no architectural constraints on batch-processing LLMs, allowing seamless application to any pre-trained LLM while ensuring high scalability and adaptability to various real-world streaming tasks.

[1]We clarify that re-encoding the target tokens is solely for refining the generation of the latest token without altering previously generated content.

2 Streaming-Batch Mismatches

LLMs are pre-trained in a batch-processing paradigm, where the entire input sequence $\mathbf{X} = [x_1, \ldots, x_n]$ is processed simultaneously to generate the output sequence $\mathbf{Y} = [y_1, \ldots, y_m]$. This paradigm assumes full input availability, allowing both self-attention and cross-attention mechanisms to operate over complete sequences. In contrast, streaming tasks require incremental processing, where inputs and outputs arrive and are processed in an interleaved manner over time. At any time step t, the model only has access to a partial input sequence $\mathbf{X}_t = [x_1, \ldots, x_t]$ and generates a corresponding partial output sequence $\mathbf{Y}_{t'} = [y_1, \ldots, y_{t'}]$. This shift from batch to streaming introduces three key mismatches:

Input-Attention Mismatch In batch-streaming mode, self-attention enforces a strict ordering, where each input token x_i can only attend to prior inputs $\mathbf{X}_{<i}$. This is typically expressed as:

$$h_i = \text{SelfAttention}(x_i, \mathbf{X}_{<i}), \tag{1}$$

where h_i is the hidden representation of x_i. However, in interleaved-streaming mode, as outputs are generated incrementally, previously decoded outputs $\mathbf{Y}_{<t'}$ become available and are included in the attention context:

$$h_i^{\text{interleaved}} = \text{SelfAttention}(x_i, \mathbf{X}_{<i} \cup \mathbf{Y}_{<t'}). \tag{2}$$

This disrupts the model's pre-trained assumptions, as input tokens in batch mode never attend to output, potentially leading to degraded performance.

Output-Attention Mismatch In the batch-streaming mode, each generated output token y_k can attend to all input tokens $\mathbf{X}$ by KV cache re-encoding:

$$h_k = \text{CrossAttention}(y_k, \mathbf{X}) \qquad k \leq j, \tag{3}$$

where y_j is the latest generated output token. However, in interleaved-streaming mode, output tokens can only attend to the subset of inputs $\mathbf{X}_t$ received up to the current step:

$$h_j^{\text{interleaved}} = \text{CrossAttention}(y_j, \mathbf{X}_{\leq t}). \tag{4}$$

This temporal constraint means that the hidden representation of each decoded token is computed based only on the partial input sequence available at the time, which may lead to inconsistencies compared to batch-mode processing.

Position-ID Mismatch In batch-streaming, tokens receive contiguous position IDs by position re-encoding, so that for an input sequence $\mathbf{X}_t$ and output sequence $\mathbf{Y}_{t'}$, we have:

$$p(x_i) = i, \quad p(y_j) = t + j, \tag{5}$$

ensuring that the relative positional differences, $p(t_j) - p(t_i)$, accurately reflect the true token order and guide the positional embedding function $g(p(t))$ to generate coherent embeddings. For interleaved-streaming, however, inputs and outputs are interleaved (e.g., $x_1, y_1, x_2, y_2, \ldots$) while still being assigned continuous IDs from 1 to $n + m$. This misrepresents the true temporal gaps between tokens; the relative differences $p(t_j) - p(t_i)$ no longer mirror the actual sequence structure.

3 Impact Analysis of Mismatches

Applying batch-trained LLMs to streaming mode introduces structural mismatches. Existing research has not systematically analyzed the nature of these mismatches between streaming and batch-processing. We employ a stepwise ablation approach to systematically isolate each mismatch and assess its impact on streaming task performance.

Setup This section analyzes the impact of the three mismatches using the streaming text translation task with wait-k reading & writing policy (Ma et al., 2019). All experiments are conducted on the IWSLT-17 dataset (Cettolo et al., 2017), covering two cross-lingual translation tasks: En-Fr and En-De. We use Gemma2-2B-Instruct model (Team et al., 2024) and Phi3-Mini-Instruct model (Abdin et al., 2024) with 3.8B parameters for all experiments, evaluating model performance using BLEU scores. (Post, 2018).

Effects of Input-Attention Mismatch The interleaved-streaming mode, which exhibits all three mismatches, serves as our baseline for comparison. The batch-streaming mode eliminates input-attention mismatch by preventing source tokens from attending to generated target tokens. Building on this, we *apply the same positional encoding* as interleaved-streaming within the batch-streaming framework. Notably, without KV cache and position embedding re-encoding, the batch-streaming approach still retains both output-attention mismatch and position-ID mismatch.

Table 1 shows that eliminating the input-attention mismatch improves BLEU scores across

Dataset	Mode	Gemma2-2B-Instruct (wait-k)				
		1	3	5	7	Max. Imp.
En-Fr	Interleaved-streaming	$30.93_{\pm 0.08}$	$37.67_{\pm 0.11}$	$39.12_{\pm 0.09}$	$39.65_{\pm 0.07}$	
	Batch-streaming (No re.)	$33.13_{\pm 0.09}^{\uparrow 2.20}$	$39.29_{\pm 0.06}^{\uparrow 1.62}$	$40.66_{\pm 0.10}^{\uparrow 1.54}$	$40.82_{\pm 0.09}^{\uparrow 1.17}$	$\uparrow 2.20$
	Batch-streaming (Pos re.)	$33.19_{\pm 0.07}^{\uparrow 0.06}$	$39.43_{\pm 0.13}^{\uparrow 0.14}$	$40.78_{\pm 0.08}^{\uparrow 0.12}$	$40.89_{\pm 0.07}^{\uparrow 0.07}$	$\uparrow 0.14$
	Batch-streaming (All re.)	$33.47_{\pm 0.10}^{\uparrow 0.28}$	$39.62_{\pm 0.08}^{\uparrow 0.19}$	$40.91_{\pm 0.11}^{\uparrow 0.13}$	$41.01_{\pm 0.09}^{\uparrow 0.12}$	$\uparrow 0.28$
En-De	Interleaved-streaming	$20.44_{\pm 0.06}$	$26.86_{\pm 0.10}$	$29.13_{\pm 0.08}$	$29.90_{\pm 0.07}$	
	Batch-streaming (No re.)	$21.97_{\pm 0.04}^{\uparrow 1.53}$	$28.30_{\pm 0.07}^{\uparrow 1.44}$	$30.52_{\pm 0.06}^{\uparrow 1.39}$	$31.36_{\pm 0.05}^{\uparrow 1.46}$	$\uparrow 1.53$
	Batch-streaming (Pos re.)	$22.06_{\pm 0.03}^{\uparrow 0.09}$	$28.38_{\pm 0.05}^{\uparrow 0.08}$	$30.63_{\pm 0.04}^{\uparrow 0.11}$	$31.45_{\pm 0.05}^{\uparrow 0.09}$	$\uparrow 0.11$
	Batch-streaming (All re.)	$22.25_{\pm 0.05}^{\uparrow 0.19}$	$28.61_{\pm 0.06}^{\uparrow 0.23}$	$30.77_{\pm 0.07}^{\uparrow 0.14}$	$31.56_{\pm 0.06}^{\uparrow 0.11}$	$\uparrow 0.23$
Dataset	**Mode**	**Phi3-Mini-Instruct (wait-k)**				
		1	3	5	7	Max. Imp.
En-Fr	Interleaved-streaming	$29.03_{\pm 0.10}$	$36.54_{\pm 0.14}$	$38.42_{\pm 0.13}$	$39.27_{\pm 0.09}$	
	Batch-streaming (No re.)	$30.96_{\pm 0.10}^{\uparrow 1.93}$	$38.42_{\pm 0.08}^{\uparrow 1.88}$	$39.80_{\pm 0.07}^{\uparrow 1.42}$	$40.93_{\pm 0.11}^{\uparrow 1.66}$	$\uparrow 1.93$
	Batch-streaming (Pos re.)	$31.08_{\pm 0.06}^{\uparrow 0.12}$	$38.51_{\pm 0.08}^{\uparrow 0.09}$	$39.87_{\pm 0.12}^{\uparrow 0.07}$	$40.96_{\pm 0.05}^{\uparrow 0.03}$	$\uparrow 0.12$
	Batch-streaming (All re.)	$31.21_{\pm 0.09}^{\uparrow 0.20}$	$38.67_{\pm 0.13}^{\uparrow 0.16}$	$39.98_{\pm 0.11}^{\uparrow 0.11}$	$41.05_{\pm 0.07}^{\uparrow 0.09}$	$\uparrow 0.20$
En-De	Interleaved-streaming	$20.74_{\pm 0.05}$	$27.46_{\pm 0.14}$	$29.56_{\pm 0.10}$	$30.67_{\pm 0.06}$	
	Batch-streaming (No re.)	$22.21_{\pm 0.08}^{\uparrow 1.47}$	$28.85_{\pm 0.11}^{\uparrow 1.39}$	$30.88_{\pm 0.05}^{\uparrow 1.32}$	$31.92_{\pm 0.07}^{\uparrow 1.25}$	$\uparrow 1.47$
	Batch-streaming (Pos re.)	$22.28_{\pm 0.06}^{\uparrow 0.07}$	$28.87_{\pm 0.09}^{\uparrow 0.02}$	$30.91_{\pm 0.11}^{\uparrow 0.03}$	$31.95_{\pm 0.13}^{\uparrow 0.03}$	$\uparrow 0.07$
	Batch-streaming (All re.)	$22.45_{\pm 0.07}^{\uparrow 0.17}$	$28.98_{\pm 0.07}^{\uparrow 0.11}$	$31.01_{\pm 0.07}^{\uparrow 0.10}$	$32.03_{\pm 0.07}^{\uparrow 0.08}$	$\uparrow 0.17$

Table 1: The BLEU performance variations reflect the stepwise elimination of mismatches between batch processing and streaming. Interleaved-streaming represents the presence of all three mismatches. Batch-streaming (No re.) corresponds to batch-streaming with interleaved position encoding, where the input-attention mismatch is eliminated. Batch-streaming (Pos re.) further removes the position-ID mismatch through position re-encoding. Finally, Batch-streaming (All re.) eliminates the output-attention mismatch by re-encoding the KV cache.

different wait-k strategies, with a maximum increase of 2.20 on the En-Fr translation task and 1.53 on the En-De translation task. This indicates that processing streaming data in an interleaved streaming manner with a batch-pretrained model leads to performance degradation.

Effects of Position-ID Mismatch Re-encoding can address the remaining two mismatches. We further decompose re-encoding into two components: KV cache re-encoding and position embedding re-encoding. The former enables target tokens to attend to the most recently available tokens, thereby resolving the output-attention mismatch. The latter corrects the position-ID mismatch by adjusting position embeddings to align with the streaming paradigm. Expanding on this, the batch-streaming paradigm with position embedding re-encoding further resolves the position-ID mismatch while still retaining output-attention mismatch.

Table 1 shows that position embeddings re-encoding does not lead to significant performance improvements, with a maximum gain of only 0.14 on the En-Fr and En-De translation tasks. This suggests that position-ID mismatch is not a primary factor affecting streaming task performance, challenging previous claims regarding the role of positional encoding in streaming models.

Effects of Output-Attention Mismatch The KV cache re-encoding can address the remaining output-attention mismatch. On the setting of the former, incorporating both KV cache and position embedding re-encoding into batch-streaming paradigm eliminates all mismatches, making it closely resemble the batch-processing setting.

Table 1 shows that re-encoding previously generated tokens also does not significantly improve model performance. Although re-encoding allows target tokens to attend to the most recent input context, the inherent constraints of streaming tasks prevent already generated outputs from being modified. As a result, re-encoding does not alter the fundamental partial information nature of streaming tasks; instead, its primary effect is to correct the generation path of subsequent tokens. However, experimental results indicate that this correction is not a decisive factor in performance improvement.

Our experiments demonstrate that input-attention mismatch significantly impacts streaming

Model	Position setting	En-Fr (Wait-k)				En-De (Wait-k)			
		1	3	5	7	1	3	5	7
Gemma2-2B-Instruct	Remove all pos.	27.11	34.98	37.54	38.02	19.01	25.93	27.71	28.87
	Remove source pos.	28.35	36.12	38.42	39.03	19.63	26.82	28.08	29.36
	Remove target pos.	29.14	36.83	39.01	39.62	19.91	27.01	28.59	29.51
	Retain all pos.	**33.23**	**39.39**	**40.76**	**40.92**	**22.35**	**28.88**	**30.84**	**31.47**
Phi3-Mini-Instruct (3.8B)	Remove all pos.	26.73	34.85	37.31	37.92	18.86	25.87	27.79	29.01
	Remove source pos.	27.98	35.96	38.17	38.95	19.47	26.78	28.19	29.54
	Remove target pos.	28.84	36.58	39.04	39.46	19.83	26.95	28.64	29.78
	Retain all pos.	**30.96**	**38.45**	**39.89**	**40.57**	**22.21**	**28.86**	**30.92**	**31.94**

Table 2: Effect of source and target position removal on streaming LLMs performance. We simulate position removal by assigning a constant position ID of 0 to all tokens instead of removing the positional embeddings.

translation performance, highlighting the performance gains of using a batch-processing architecture for streaming tasks.[2] On the other hand, contrary to existing studies (Raffel et al., 2024; Guo et al., 2024a; He et al., 2024), position-ID mismatch is not the primary reason for re-encoding, and interleaved positional encoding achieves performance comparable to continuous position encoding in batch processing. To investigate the discrepancy between our findings with the common assumption, we conduct a comprehensive analysis of **how position encoding impacts LLMs in streaming scenarios.**

4 Impact Analysis of Position Encoding

The above analysis suggests that positional mismatches do not significantly impact the performance of streaming tasks. To further elucidate this phenomenon, this section provides a detailed investigation into the impact of positional encoding on the performance of LLMs in streaming scenarios.

4.1 Is Position Encoding Necessary for Streaming Tasks?

Building upon the experimental setup from the previous section, we further investigate the necessity of positional encoding in streaming tasks by separately removing global positional encoding and target-side positional encoding. Table 2 presents the BLEU scores on the En-Fr and En-De streaming translation tasks after removing position encodings at different locations. We simulate position removal by assigning a constant position ID of 0 to all tokens instead of removing the positional encoding module. For the setting of position-retaining, we apply interleaved positional encoding as illustrated in previous section. The table reveals that removing positional information from either the source or target side results in a clear performance degradation, with the maximum drop exceeding 10%. In contrast, when both source and target positional information are removed, the model maintains roughly 80% of its BLEU score compared to the fully position-retaining setting.

This finding aligns with previous studies suggesting that LLMs can still learn certain positional information even without explicit positional encoding (Haviv et al., 2022). However, it is important to emphasize that positional encoding remains relevant for streaming tasks, particularly on the target side. Notably, the absence of target-side positional encoding leads to a measurable performance decline, highlighting its role in maintaining effective token generation in streaming scenarios.

4.2 Group Position Encoding Is An Option for Streaming Tasks

Given that positional encoding is necessary and interleaved positional encoding has minimal impact on streaming task performance, one might question whether streaming problems can be modeled using interleaved positional encoding and batch-streaming mode. However, this is not an optimal choice, as **interleaved positional encoding lacks direct generalizability to batch processing.**

In real-world scenarios, the target sequence is not available in advance, making it impossible to predefine source positions. This limitation hinders the generalization of streaming models to offline settings. To address this issue, we propose **a group position encoding** based on batch-streaming framework for streaming LLMs as shown in Figure 2,

[2]We provide the detailed training process for different settings in the Appendix B.3.

Model	Wait-k	En-Fr (Target start id ϕ)						En-De (Target start id ϕ)					
		0	0.5	128	256	512	Δ	0	0.5	128	256	512	Δ
Gemma2-2B-Instruct	5	40.76	40.76	40.70	40.57	40.68	*0.19*	30.84	30.84	30.90	30.80	30.95	*0.15*
	7	40.92	40.92	40.85	40.91	40.92	*0.07*	31.47	31.47	31.44	31.57	31.67	*0.23*
	9	40.91	40.91	40.90	40.88	40.97	*0.09*	31.73	31.73	31.87	31.91	31.88	*0.18*
	11	41.10	41.10	41.14	40.96	41.05	*0.18*	31.95	31.95	31.98	31.95	31.89	*0.09*
Phi3-Mini-Instruct (3.8B)	5	39.89	39.89	39.91	40.06	39.87	*0.19*	30.92	30.92	30.76	30.81	30.86	*0.16*
	7	40.57	40.57	40.53	40.72	40.71	*0.19*	31.94	31.94	31.78	31.84	31.78	*0.16*
	9	41.31	41.31	41.24	41.35	41.44	*0.20*	32.18	32.18	32.10	32.21	32.09	*0.12*
	11	41.92	41.92	42.03	41.94	41.93	*0.11*	32.26	32.26	32.23	32.33	32.28	*0.10*
LLama3.1-8B-Instruct	5	40.11	40.11	40.10	39.93	39.92	*0.19*	30.33	30.33	30.21	30.37	30.34	*0.16*
	7	40.30	40.30	40.32	40.35	40.31	***0.03***	31.23	31.23	31.18	31.16	31.25	*0.09*
	9	40.15	40.15	40.32	40.34	40.35	*0.20*	31.80	31.80	31.83	31.76	31.89	*0.13*
	11	40.53	40.53	40.47	40.58	40.63	*0.16*	32.04	32.04	31.98	32.07	32.08	*0.10*

Table 3: Performance comparison of different models with various wait-k policies and target start IDs. Δ represents the range of variation in BLEU scores when the start id of target token takes different values. We use bold to indicate the smallest variation and underline to represent the largest variation.

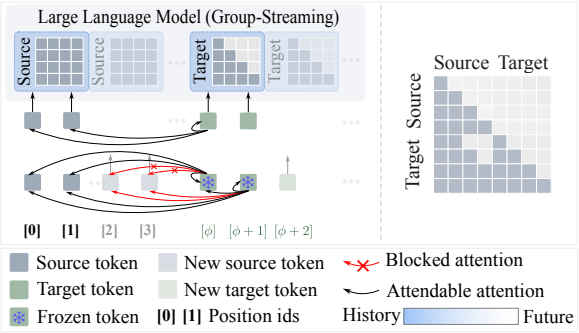


Figure 2: Framework of our *Group-streaming* LLMs. (Left) Positional grouping of source and target tokens in the streaming LLM, avoiding re-encoding. The group start ID ϕ is a hyperparameter. (Right) The attention mask matrix during the training ensures that target tokens can only attend to locally available inputs.

where source and target tokens are independently assigned positional encodings, ensuring only monotonic continuity within each group. Specifically, in our proposed approach, the source position encoding remains consistent with batch processing mode, starting from 0. In contrast, the target position begins from a predefined starting value ϕ.

This approach makes it feasible to prefill source position encodings even without target information and naturally extends to batch processing. In fact, interleaved position encoding can be viewed as a special case of group position encoding, with the distinction that the interleaved mode uses non-uniform positional intervals.

4.3 What is the Impact of Group Position Offset on Model Performance?

This section provides a detailed discussion on the impact of target position offset ϕ.

Setup We evaluate the impact of grouped positional encoding on text translation and automatic speech recognition tasks. For text translation, we use the IWSLT-17 dataset, focusing on En-Fr and En-De translation tasks, with models including Gemma2-2B-Instruct (Team et al., 2024), Phi3-Mini-Instruct (Abdin et al., 2024), and LLaMA3.1-8B-Instruct (Dubey et al., 2024). For ASR, we use the LibriSpeech dataset (Panayotov et al., 2015), with Phi3 as the selected model. Translation performance is assessed using BLEU scores, while ASR performance is evaluated based on WER (Radford et al., 2023). Detailed experimental settings and hyperparameters are provided in the appendix.

Results The streaming text translation task results in Table 3 and streaming ASR task in Table 4 indicate that varying the initial offset of the target-side group position encoding within a reasonable range does not significantly affect the performance of LLMs in streaming scenarios. This suggests that the model is highly robust to the choice of initial group position offset. Specifically, when the offset is set to 0, the source and target positions are fully overlap, whereas an offset of 0.5 results in complete separation. Despite this contrast, both settings yield comparable performance, suggesting that positional overlap appears to have limited impact on

Wait-k	Speech-Text (Target start id ϕ)					
	0	256	512	1024	2048	Δ
1	6.02	6.05	6.04	6.07	6.17	*0.15*
3	4.12	4.10	4.09	4.08	4.19	*0.11*
5	3.52	3.58	3.55	3.59	3.61	***0.09***
7	3.33	3.33	3.38	3.41	3.45	*0.12*

Table 4: Performance of Phi3 with various wait-k policies and target start IDs. Δ represents the range of variation in WER scores when the start id of target token takes different values.

the effectiveness of group position encoding.

4.4 Why Group Position Encoding Works?

The RoPE encodes relative position information via rotation matrices R applied to each token's query and key: $q_n^r = R(n)q_n$ and $k_n^r = R(n)k_n$, where n denotes the position ID. Then the dot product attention score can be written as $Attn(n, \text{cache}) = \sum_i {q_n^r}^T {k_i}^r = \sum_i q_n^T R^T(n) R(i) k_i = \sum_{i=0}^{S+n} q_n^T R(n-i) k_i$, where S is the token length of source input and q_n is query of a target token. The relative position can be written as $\Delta = n - i = \phi + j - i$, where j denotes the index of the target token and ϕ represents the position offset between the first token of the target and that of the source. We split the above dot-product attention into two parts: target-to-target and target-to-source computations:

$$Attn(n, \text{cache}) = \sum_{i=0}^{j} q_n^T R(j-i) k_i + \sum_{i=0}^{S} q_n^T R(\phi + j - i) k_i. \quad (6)$$

The relative position $j - i$ in the first target-to-target term remains consistent across both RoPE and group position encoding. For cross-segment attention in target-to-source, the difference of relative position between RoPE and group position encoding is determined by the position offset ϕ. In original RoPE, ϕ equals the length of the source sequence and varies with input length, whereas in group position encoding, ϕ is predefined as a fixed constant. LLMs are capable of easily learning and internalizing the semantics of the relative offset by fine-tuning. Once the model has correctly understood the meaning of ϕ as a position shift, it can accurately capture and assign position relationships across segments, without requiring explicit differentiation between source and target token IDs.[3]

LLMs can learn the position offset ϕ through simple fine-tuning, so typical values of ϕ do not significantly impact performance. However, when ϕ becomes extremely large, it may lead to discrepancies with the model's pretraining distribution due to the limited context length used during pretraining. Therefore, a reasonable range for ϕ should ensure that the maximum relative distance between the last target token and the first source token remains within the model's pretraining context length.[4]

We recommend using a relatively small ϕ, ideally below the input sentence length, to keep relative position gaps closer to the pretraining distribution, which may facilitate faster convergence and better performance. Notably, when $\phi = 0$, the target starting token is positioned closer to the source starting token and farther from the source ending token. This configuration better reflects the sequential input arrival pattern in streaming scenarios, leading to more stable learning dynamics and enhanced model alignment.

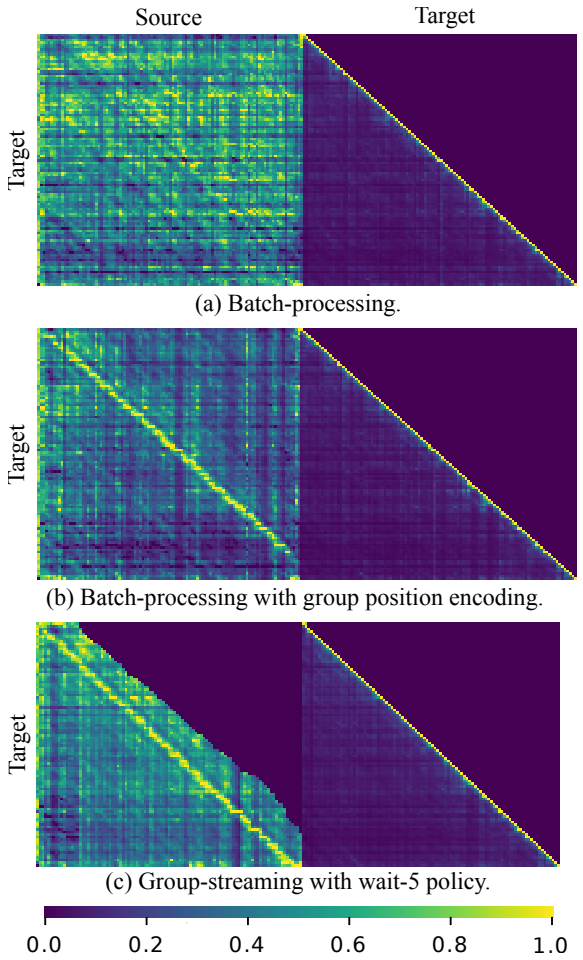
4.5 Visualization of Streaming Attention

Taking text translation as an example, we visualize the extent to which each target token attends to past source information during inference. Notably, we normalize the attention weights column-wise (i.e., across each source token) to the range [0, 1]. This normalization offers two key benefits: (1) it mitigates the influence of tokens with inherently large absolute attention values and highlights the relative importance of attention distribution, making attention strength more interpretable; and (2) it provides a clearer view of *how each source token distributes its attention across different target tokens.*

As shown in Figure 3, under the batch setting, source tokens distribute their attention uniformly across all target tokens, reflecting a globally constrained behavior. In other words, each target token tends to attend equally to the same source token. In contrast, with group position encoding, source tokens tend to assign more attention to target tokens with similar positional indices. That is, source tokens are less likely to attend to future target tokens. This observation supports our earlier finding that re-encoding previously generated target tokens offers limited performance gain in streaming tasks under group position encoding.

[3]The detailed analysis can be found in Appendix D.

[4]We provide additional experiments to demonstrate the potential edge in Appendix D.3.



(a) Batch-processing.

(b) Batch-processing with group position encoding.

(c) Group-streaming with wait-5 policy.

Figure 3: An example of the attention distribution of target tokens, where the attention values of each target token are normalized to emphasize the relative focus. The sample is from IWSLT-17 En-Fr dataset. [5]

Moreover, the results in Figure 3 indicate that employing group position encoding in the batch-processing setting shifts the target tokens' attention to the source context along the diagonal direction when the offset $\phi = 0$. This adjustment encourages target tokens to focus more on the currently available input, making the model's behavior more aligned with the requirements of streaming tasks.

5 Discussion

Why LLMs? We apply the proposed group-streaming approach to mainstream large language models and compare its performance against other decoder-only streaming models to highlight its advantages. To demonstrate the effectiveness of our method, we evaluate it on the En-Fr and En-De translation tasks from the IWSLT-17 dataset, as well as the ASR task from the Librispeech dataset. The baselines for text translation include SimulMask (Raffel et al., 2024) and DST (Guo et al., 2024a), while the baselines for ASR include CAAT (Liu et al., 2021) and Wav2Vec-S (Fu et al., 2024).

As shown in Figure 4, the vertical axis represents task-specific performance metrics—BLEU for translation and WER for ASR—while the horizontal axis indicates the model's average latency (AL and LAAL), measured by the number of waited words in translation and the waiting time in ASR. The results show that group-streaming LLMs consistently outperform specialized decoder-only baselines, typically achieving higher accuracy under the same latency conditions.

Generalization We extend our group position encoding to batch processing. The first bar in Figure 5 represents the model that is specifically trained for batch processing using the original RoPE. Subsequently, we applied group position encoding to the batch processing scenario and fine-tuned the model. The results demonstrate that applying group position encoding introduces no performance degradation for batch processing, confirming its compatibility and generalization across both streaming and batch processing settings.

6 Related Work

Streaming Language and Speech Transformers A typical implementation of Transformer-based streaming tasks adopts an incremental encoding strategy on the encoder side and an incremental decoding strategy on the decoder side (Ma et al., 2021, 2023; Zhang and Feng, 2023). With the rise of large language models, researchers have begun exploring how to adapt decoder-only architectures for streaming tasks. Among these approaches, batch-streaming models based on prompt structures attempt to approximate offline batch processing by re-encoding tokens during streaming inference (Agostinelli et al., 2024; Guo et al., 2024b; Koshkin et al., 2024; Wang et al., 2024). Some studies suggest that position confusion in streaming environments is a key factor necessitating re-encoding in LLMs (Guo et al., 2024a; Raffel et al., 2024). To address this issue, one line of research focuses on modifying the decoder-only architecture to enhance its adaptability to streaming tasks (Guo et al., 2024a; Tsunoo et al., 2024; Chen et al., 2024), while another emphasizes optimizing positional encoding—such as the ALIBI positional

[5] Note that the attention values have been normalized. The values do not represent the actual magnitude of attention.

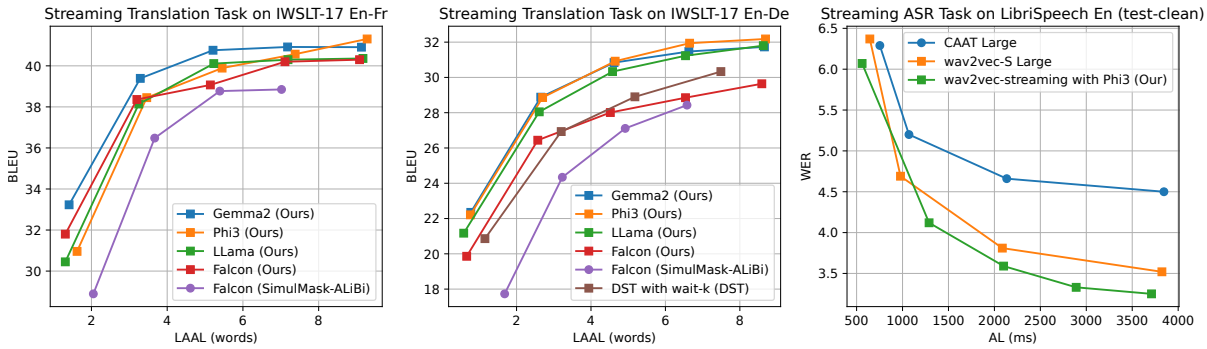


Figure 4: The performance comparison between group position streaming LLMs with other decoder-only models.

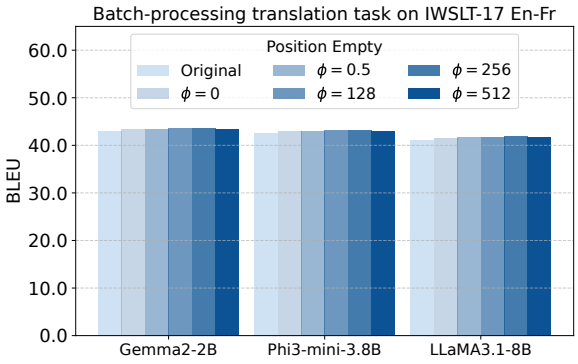


Figure 5: The BLEU performance of batch-processing translation task on IWSLT-17 En-Fr dataset.

encoding—to mitigate the effects of incremental position shifts during streaming decoding (Raffel et al., 2024). In contrast to simulated batch processing, an interleaved-streaming paradigm (Du et al., 2024; Yang et al., 2024) that adheres to temporal order has been explored, wherein input and output tokens are interleaved and encoded sequentially. While significant progress has been made in developing streaming models, existing studies lack a rigorous analysis of the fundamental differences between batch processing and streaming paradigms.

Position Encoding in Transformers Position encoding (Raffel et al., 2020; Press et al., 2022; Su et al., 2024) is a crucial component of Transformer models (Vaswani, 2017), designed to break the permutation-invariant nature of self-attention mechanisms. Recent studies have demonstrated that decoder-only Transformer models can still capture positional information even in the absence of explicit positional encoding (Shen et al., 2018). A plausible explanation is that causal attention masks enforce position-dependent token interactions, implicitly encoding positional information (Haviv et al., 2022; Tsai et al., 2019). Related research has shown that in tasks such as speech modeling (Likhomanenko et al., 2021) and language modeling (Haviv et al., 2022), decoder-only Transformers without positional encoding can achieve performance comparable to standard decoder-based Transformers. Furthermore, other studies (Kazemnejad et al., 2024; Ruoss et al., 2023) have indicated that the generalization ability of Transformers without positional encoding does not degrade significantly when handling varying context lengths. While significant progress has been made in understanding positional encoding in LLMs, existing research has primarily focused on static scenarios. In contrast, the role of positional encoding in streaming scenarios remains underexplored, where the dynamic modeling of positional information may follow different patterns and exert distinct effects.

7 Conclusion

This work provides a systematic analysis of the mismatches that arise in adapting batch-trained LLMs to streaming tasks. We identify input-attention mismatch as the primary bottleneck, while output-attention and position-ID mismatches have negligible impact, challenging the prevailing assumption that position inconsistencies necessitate frequent re-encoding. To clarify this, we conduct the first in-depth analysis of position encoding in streaming settings, showing that preserving strict absolute positions is unnecessary; instead, maintaining relative token order within source and target contexts is more critical. Building on the insights, we propose the *group streaming paradigm*, a simple yet effective strategy that bridges the gap between streaming and batch modes without requiring re-encoding. The approach is model-agnostic and generalizable, achieving strong performance across both cross-lingual and cross-modal streaming tasks.

Limitations

This paper primarily focuses on exploring the optimal paradigm for streaming models and, therefore, does not delve into different waiting policies. The conclusions drawn in this study have only been validated under the wait-k policy. Additionally, our study is confined to streaming tasks in the text and audio modalities, with video streaming left for future investigation.

Acknowledgements

We thank EIT and IDT High Performance Computing Center for providing computational resources for this project. This work was supported by the 2035 Key Research and Development Program of Ningbo City under Grant No.2024Z127.

References

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, et al. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*.

Victor Agostinelli, Max Wild, Matthew Raffel, Kazi Fuad, and Lizhong Chen. 2024. Simul-LLM: A framework for exploring high-quality simultaneous translation with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL 2024)*.

Gerry TM Altmann and Jelena Mirković. 2009. Incrementality and prediction in human sentence processing. *Cognitive science*, 33(4):583–609.

Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. 2020. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in neural information processing systems*, 33:12449–12460.

Mauro Cettolo, Marcello Federico, Luisa Bentivogli, Jan Niehues, Sebastian Stüker, Katsuitho Sudoh, Koichiro Yoshino, and Christian Federmann. 2017. Overview of the iwslt 2017 evaluation campaign. In *Proceedings of the 14th International Workshop on Spoken Language Translation*, pages 2–14.

Junkun Chen, Mingbo Ma, Renjie Zheng, and Liang Huang. 2021. Direct simultaneous speech-to-text translation assisted by synchronized streaming asr. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4618–4624.

Peikun Chen, Sining Sun, Changhao Shan, Qing Yang, and Lei Xie. 2024. Streaming decoder-only automatic speech recognition with discrete speech units: A pilot study. In *Proc. Interspeech 2024*, pages 4468–4472.

Yunfei Chu, Jin Xu, Xiaohuan Zhou, Qian Yang, Shiliang Zhang, Zhijie Yan, Chang Zhou, and Jingren Zhou. 2023. Qwen-audio: Advancing universal audio understanding via unified large-scale audio-language models. *arXiv preprint arXiv:2311.07919*.

Qian Dong, Yaoming Zhu, Mingxuan Wang, and Lei Li. 2022. Learning when to translate for streaming speech. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL 2022)*, pages 680–694.

Zhihao Du, Yuxuan Wang, Qian Chen, Xian Shi, Xiang Lv, Tianyu Zhao, Zhifu Gao, Yexin Yang, Changfeng Gao, Hui Wang, et al. 2024. Cosyvoice 2: Scalable streaming speech synthesis with large language models. *arXiv preprint arXiv:2412.10117*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Biao Fu, Kai Fan, Minpeng Liao, Yidong Chen, Xiaodong Shi, and Zhongqiang Huang. 2024. wav2vec-s: Adapting pre-trained speech models for streaming. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 11465–11480.

Cleotilde Gonzalez, Javier F Lerch, and Christian Lebiere. 2003. Instance-based learning in dynamic decision making. *Cognitive Science*, 27(4):591–635.

Shoutao Guo, Shaolei Zhang, and Yang Feng. 2024a. Decoder-only streaming transformer for simultaneous translation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL 2024)*.

Shoutao Guo, Shaolei Zhang, Zhengrui Ma, Min Zhang, and Yang Feng. 2024b. Sillm: Large language models for simultaneous machine translation. *arXiv preprint arXiv:2402.13036*.

Adi Haviv, Ori Ram, Ofir Press, Peter Izsak, and Omer Levy. 2022. Transformer language models without positional encodings still learn positional information. *arXiv preprint arXiv:2203.16634*.

Zhenyu He, Jun Zhang, Shengjie Luo, Jingjing Xu, Zhi Zhang, and Di He. 2024. Let the code llm edit itself when you edit the code. *arXiv preprint arXiv:2407.03157*.

Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Payel Das, and Siva Reddy. 2024. The impact of positional encoding on length generalization in transformers. *Advances in Neural Information Processing Systems*, 36.

Tom Kocmi and Christian Federmann. 2023. Large language models are state-of-the-art evaluators of translation quality. *arXiv preprint arXiv:2302.14520*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Roman Koshkin, Katsuhito Sudoh, and Satoshi Nakamura. 2024. TransLLaMa: LLM-based simultaneous translation system. In *Findings of the Association for Computational Linguistics: EMNLP 2024*.

Tatiana Likhomanenko, Qiantong Xu, Gabriel Synnaeve, Ronan Collobert, and Alex Rogozhnikov. 2021. Cape: Encoding relative positions with continuous augmented positional embeddings. *Advances in Neural Information Processing Systems*, 34:16079–16092.

Dan Liu, Mengge Du, Xiaoxi Li, Ya Li, and Enhong Chen. 2021. Cross attention augmented transducer networks for simultaneous translation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2024. Visual instruction tuning. *Advances in neural information processing systems*, 36.

Mingbo Ma, Liang Huang, Hao Xiong, Renjie Zheng, Kaibo Liu, Baigong Zheng, Chuanqiang Zhang, Zhongjun He, Hairong Liu, Xing Li, et al. 2019. Stacl: Simultaneous translation with implicit anticipation and controllable latency using prefix-to-prefix framework. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL 2019)*, pages 3025–3036.

Xutai Ma, Yongqiang Wang, Mohammad Javad Dousti, Philipp Koehn, and Juan Pino. 2021. Streaming simultaneous speech translation with augmented memory transformer. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7523–7527. IEEE.

Zhengrui Ma, Shaolei Zhang, Shoutao Guo, Chenze Shao, Min Zhang, and Yang Feng. 2023. Non-autoregressive streaming transformer for simultaneous translation. *arXiv preprint arXiv:2310.14883*.

Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. 2015. Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5206–5210. IEEE.

Matt Post. 2018. A call for clarity in reporting bleu scores. *arXiv preprint arXiv:1804.08771*.

Ofir Press, Noah Smith, and Mike Lewis. 2022. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations (ICLR 2022)*.

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. 2023. Robust speech recognition via large-scale weak supervision. In *International conference on machine learning*, pages 28492–28518. PMLR.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research (JMLR)*, 21(140):1–67.

Matthew Raffel, Victor Agostinelli, and Lizhong Chen. 2024. Simultaneous masking, not prompting optimization: A paradigm shift in fine-tuning LLMs for simultaneous translation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP 2024)*.

Anian Ruoss, Grégoire Delétang, Tim Genewein, Jordi Grau-Moya, Róbert Csordás, Mehdi Bennani, Shane Legg, and Joel Veness. 2023. Randomized positional encodings boost length generalization of transformers. *arXiv preprint arXiv:2305.16843*.

Tao Shen, Tianyi Zhou, Guodong Long, Jing Jiang, Shirui Pan, and Chengqi Zhang. 2018. Disan: Directional self-attention network for rnn/cnn-free language understanding. In *Proceedings of the AAAI conference on artificial intelligence*.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*.

Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. 2019. Transformer dissection: a unified understanding of transformer's attention via the lens of kernel. *arXiv preprint arXiv:1908.11775*.

Emiru Tsunoo, Hayato Futami, Yosuke Kashiwagi, Siddhant Arora, and Shinji Watanabe. 2024. Decoder-only architecture for streaming end-to-end speech recognition. In *Proc. Interspeech 2024*, pages 4463–4467.

A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems (NeurIPS 2017)*.

Minghan Wang, Thuy-Trang Vu, Jinming Zhao, Fatemeh Shiri, Ehsan Shareghi, and Gholamreza Haffari. 2024. Simultaneous machine translation with large language models. In *Proceedings of the 22nd Annual Workshop of the Australasian Language Technology Association*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations (ICLR 2024)*.

Yifan Yang, Ziyang Ma, Shujie Liu, Jinyu Li, Hui Wang, Lingwei Meng, Haiyang Sun, Yuzhe Liang, Ruiyang Xu, Yuxuan Hu, et al. 2024. Interleaved speech-text language models are simple streaming text to speech synthesizers. *arXiv preprint arXiv:2412.16102*.

Jun Zhan, Junqi Dai, Jiasheng Ye, Yunhua Zhou, Dong Zhang, Zhigeng Liu, Xin Zhang, Ruibin Yuan, Ge Zhang, Linyang Li, Hang Yan, Jie Fu, Tao Gui, Tianxiang Sun, Yu-Gang Jiang, and Xipeng Qiu. 2024. AnyGPT: Unified multimodal LLM with discrete sequence modeling. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL 2024)*.

Dong Zhang, Shimin Li, Xin Zhang, Jun Zhan, Pengyu Wang, Yaqian Zhou, and Xipeng Qiu. 2023a. Speechgpt: Empowering large language models with intrinsic cross-modal conversational abilities. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 15757–15773.

Hang Zhang, Xin Li, and Lidong Bing. 2023b. Video-llama: An instruction-tuned audio-visual language model for video understanding. *arXiv preprint arXiv:2306.02858*.

Shaolei Zhang and Yang Feng. 2023. Hidden markov transformer for simultaneous machine translation. *arXiv preprint arXiv:2303.00257*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.

A Different Paradigms on Streaming Tasks

The main text introduces three approaches for applying LLMs to streaming tasks: batch-streaming, interleaved-streaming, and group-streaming. Among them, batch-streaming maximally simulates the batch-processing paradigm in offline scenarios through re-encoding, with the only difference being the availability of local information in a streaming setting. Figure 1 illustrates different paradigms of LLM data processing using ASR as an example.

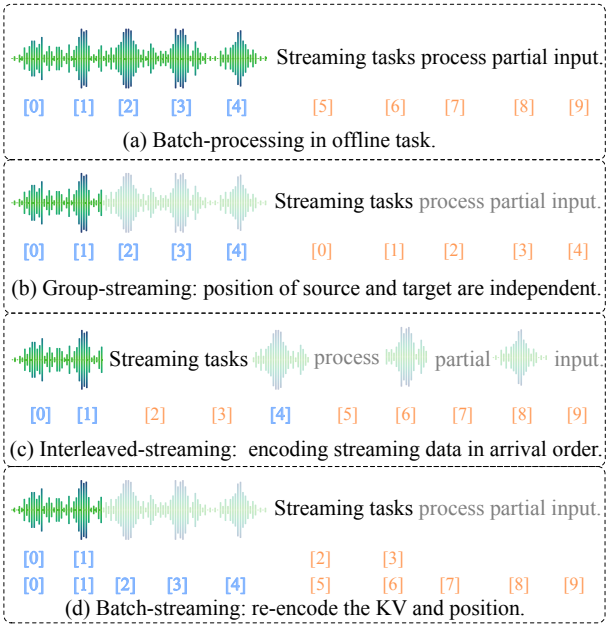


Figure 1: An ASR example for illustration of different paradigms for LLMs processing.

We clarify that re-encoding refers to reprocessing all previously generated target tokens after each new source context is read, before generating the next target token. This is solely for optimizing the generation of the latest token without altering previously output content. We exclude scenarios where re-encoding continuously adjusts already output content, as the final alignment after reading the entire input would be equivalent to batch processing. In this context, re-encoding clearly holds positive value.

B Model Details

B.1 Model Structure

Streaming Text LLM The group-streaming model, as previously introduced, is designed to enforce a strict attention constraint where historically generated tokens are prevented from attending to newly received source tokens, ensuring a clear separation between past and present information. Additionally, the model maintains independent positional encoding for both source and target tokens, preserving structural integrity while facilitating effective streaming processing.

Streaming Speech LLM Figure 2 illustrates the structure of the streaming ASR model proposed in this paper. The model consists of a streaming audio encoder, an MLP projector, and our Group Positional Encoding-based Streaming LLM.

The streaming audio encoder is a variant of Wav2vec2 (Baevski et al., 2020), with the following key modifications: (1) Positional Encoding Adjustment: We replace the convPE in Wav2vec2 with a causal convolution-based positional encoding (causal ConvPE) to enforce directional constraints on the information flow. (2) Structural Optimization: The Transformer Encoder in Wav2vec2 is replaced with a Transformer Decoder to ensure global unidirectional information constraints, thereby enhancing incremental encoding capability. We refer to this modified model as Wav2vec2-Streaming. While it shares some similarities with Wav2vec-S (Fu et al., 2024), the latter employs absolute sinusoidal positional encoding, whereas Wav2vec2-Stream retains causal convolution to improve temporal modeling.

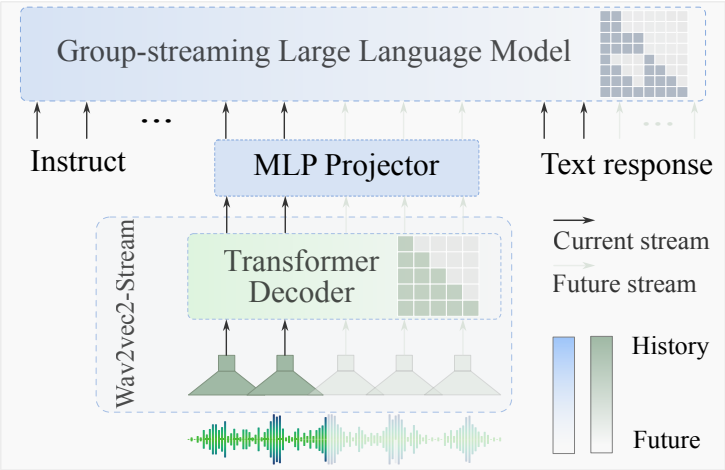


Figure 2: Illustration of our Group-streaming speech Large Language Model, where the group-streaming LLM and the streaming audio encoder are connected through an MLP projector.

Additionally, we have modified Wav2vec2 within the HuggingFace Transformers framework[6] to enable seamless interoperability with existing LLMs. Our code and pretrained weights will be open-sourced for research and application purposes.

Wav2vec2-Stream processes audio data sampled at 16 kHz, where each segment consists of 400 samples, with an 80-sample overlap between consecutive segments. This results in an embedding vector for the LLM approximately every 20 ms, ensuring a fine-grained temporal resolution for streaming speech recognition. Similar to (Chen et al., 2021; Dong et al., 2022) et al., we adopt a fixed-interval audio segmentation approach combined with the wait-k strategy for streaming tasks. In our setup, the time interval is set to 400 ms, ensuring a structured and controlled latency for real-time processing.

Unlike discrete encoding models (Zhang et al., 2023a; Zhan et al., 2024), which require expanding the LLM vocabulary to support speech-text multimodality, we propose a continuous encoding-based speech LLM. In this framework, the output features of the streaming audio encoder are mapped to the LLM space through an MLP projection layer, enabling end-to-end speech understanding and generation. This design is inspired by LLaVA (Liu et al., 2024) but has been specifically optimized for streaming speech tasks.

B.2 Data Format

In this paper, all the large language models we selected are instruction-tuned versions. To fully leverage their instruction-following capabilities, we strictly adhere to the instruction format used during their pretraining phase. Additionally, we design the data format, as shown in Figure 3 and Figure 4, to align with the specific requirements of our tasks.

B.3 Training Method

Training Method of Different Streaming Paradigms The main text analyzes the impact of different LLM paradigms on streaming tasks, covering training and evaluation methods for interleaved-streaming, batch-streaming, and group-streaming. This section provides a detailed explanation of the masking matrix design for different streaming paradigms and introduces the corresponding training methods. We explain these training paradigms in the context of the wait-k reading/writing policy (Ma et al., 2019).

Figure 5 illustrates the attention mask under different training methods, indicating the input order, position IDs, and whether a token is included in the loss calculation. Figure 5 (a) represents the standard LLM causal mask matrix, which enables batch-processing training in offline scenarios using shifted loss computation. Figure 5 (b) also employs a causal mask matrix, but the model's input consists of an interleaved sequence of source and target tokens, with position IDs assigned sequentially. Notably, each word may correspond to multiple tokens, where the first token is generated from the source, while the

[6]https://huggingface.co/facebook/wav2vec2-large-960h-lv60-self

Data format of gemma 2-Instruct

Prompt: '<bos><start_of_turn>userTranslate the following English paragraph to German\n'

Input: 'Just in the last two days, we got the new temperature records in January.<end_of_turn>'

Output: '<start_of_turn>Erst in den letzten beiden Tagen hatten wir neue Januar-Temperaturrekorde.<end_of_turn>'

Data format of Phi3-mini-Instruct

Prompt: '<|system|>Translate the following English paragraph to German:<|end|>'

Input: '<|user|>Just in the last two days, we got the new temperature records in January.<|end|>'

Output: '<|assistant|>Erst in den letzten beiden Tagen hatten wir neue Januar-Temperaturrekorde.<|end|>'

Data format of LLama-3.1-Instruct

Prompt: '<|begin_of_text|><|start_header_id|>system<|end_header_id|>Translate the following English paragraph to German:<|eot_id|>'

Input: '<|start_header_id|>user<|end_header_id|>Just in the last two days, we got the new temperature records in January.<|eot_id|>'

Output: '<|start_header_id|>assistant<|end_header_id|>Erst in den letzten beiden Tagen hatten wir neue Januar-Temperaturrekorde.<|eot_id|>'

Figure 3: Data format of text translation task. An example of translation from English to German.

Data format of gemma 2-Instruct

Prompt: '<bos><start_of_turn>userTranscribe the following English audio to English text\n'

Input: '**SPEECH**<end_of_turn>'

Output: '<start_of_turn>Just in the last two days, we got the new temperature records in January.<end_of_turn>'

Data format of Phi3-mini-Instruct

Prompt: '<|system|>Transcribe the following English audio to English text:<|end|>'

Input: '<|user|>**SPEECH**<|end|>'

Output: '<|assistant|>Just in the last two days, we got the new temperature records in January.<|end|>'

Data format of LLama-3.1-Instruct

Prompt: '<|begin_of_text|><|start_header_id|>system<|end_header_id|>Transcribe the following English audio to English text:<|eot_id|>'

Input: '<|start_header_id|>user<|end_header_id|>**SPEECH**<|eot_id|>'

Output: '<|start_header_id|>assistant<|end_header_id|>Just in the last two days, we got the new temperature records in January.<|eot_id|>'

Figure 4: Data format of ASR task, where the '**SPEECH**' is the audio embedding for input.

remaining tokens are generated from the target. During loss computation, only positions that contribute to target token generation are considered. Figure 5 (c) depicts the batch-streaming mask matrix, which is structurally akin to (Raffel et al., 2024). It maintains the batch-processing input format while adopting interleaved-streaming position encoding, preventing source tokens from attending to target tokens to eliminate input-attention mismatch and ensure streaming consistency.

Figures 5 (d), (e), and (f) represent three different re-encoding scenarios, all of which share the same mask matrix. The core assumption of re-encoding is that as new content is continuously read at the source end, both the historical KV cache and position embedding must be updated accordingly to ensure accurate next-token prediction. Therefore, the training phase should reflect this dynamic updating mechanism. However, existing approaches employ either a causal-mask or prefix-to-prefix training methods (Raffel et al., 2024), leading to a mismatch between training and inference. Specifically, causal-masked training is inherently offline and fails to capture the continuous update of content. While prefix-to-prefix training partially simulates this process, the target token is always the most recent one at each step. As more content is read, its behavior increasingly resembles an offline setting. In streaming scenarios, however, previously generated content cannot be modified, making this approach inadequate for capturing the true nature of re-encoding. To address this discrepancy, the mask matrix design in Figures 5 (d), (e), and (f) ensures consistency between the training and inference processes, effectively aligning the training paradigm with real-world inference dynamics.

Training Method of Streaming ASR Model Due to the lack of a large-scale pre-trained streaming audio encoder, our modified streaming version of Wav2vec2 requires a step-by-step training approach. We adopt a four-stage training strategy to effectively train our proposed speech large language model,

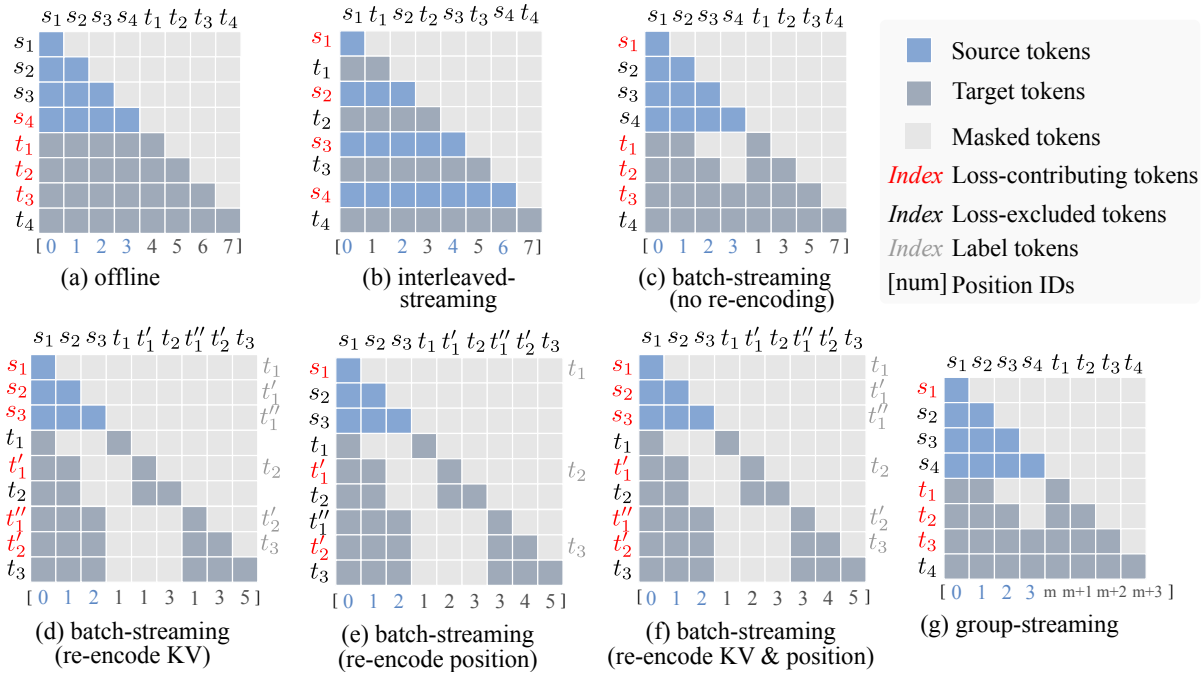


Figure 5: Attention mask matrix of different paradigms.

ensuring a smooth adaptation to streaming scenarios:

1. **Stage 1: Pre-training for Feature Alignment.** In the first stage, we focus on establishing a robust feature alignment between the streaming audio encoder and the LLM. We begin by freezing both Wav2vec2 and the LLM and train the MLP projector using a batch-processing task. The goal is to learn a stable feature transformation that maps the continuous speech representations from Wav2vec2 into a space that aligns with the LLM's token embedding space. This step is crucial for minimizing the modality gap between speech and text representations, ensuring that the LLM can effectively process speech-derived embeddings in later stages.

2. **Stage 2: Streaming Adaptation of Wav2vec2.** We replace Wav2vec2's ConvPE with the causal version used in Wav2vec2-Streaming, enabling directional constraints suitable for streaming processing. In this stage, we jointly train Wav2vec2-Streaming and the projector, allowing the model to adapt to incremental encoding while maintaining alignment with the LLM's input space.

3. **Stage 3: Streaming Adaptation of Wav2vec2.** We replace Wav2vec2's transformer encoder with the transformer decoder from Wav2vec2-Streaming. This modification ensures that the model adheres to global unidirectional constraints. We then continue joint training of Wav2vec2-Streaming and the projector, improving the encoder's ability to generate high-quality speech embeddings in real-time.

4. **Stage 4: Fine-tuning the LLM for Streaming ASR.** In the final stage, we freeze both Wav2vec2-Streaming and the projector, and fine-tune the LLM on a streaming ASR task. This step refines the LLM's ability to generate accurate text outputs from streaming speech representations, optimizing its instruction-following capabilities while maintaining low-latency processing.

C Experiments Details

C.1 Hyperparameters

When verifying grouped position encoding, the model parameters are configured as shown in Table 1. Notably, re-encoding introduces quadratic complexity, increasing the computational cost and resource requirements for both model training and inference. For the mismatch validation experiment, we reduce both the batch size and learning rate by half.

Hyperparameter	Text to Text (Gemma2, Phi3, LLama3.1)	ASR, Stage 1 (Phi3)	ASR, Stage 2 to 4 (Phi3)
Precision	bfloat16	bfloat16	bfloat16
Learning Rate	2e-4	2e-4	2e-4
LR Scheduler	Linear	Linear	Linear
Optimizer	AdamW	AdamW	AdamW
Warmup steps	500	1000	5000
Lora rank	32	64	64
Epochs	2	4	4
Batch size	64	32	64
Wait-k	1,3,5,7,9,11	1,3,5,7,9	1,3,5,7,9

Table 1: Fine-tuning hyperparameters of LLMs in this paper.

C.2 Decoding Strategy

The decoding process for streaming LLM is detailed in Algorithm 1.

Algorithm 1 Streaming decoding with wait-k policy

Input: Source length list S, target length list T, wait-k policy k.

Initialize source KV cache S_{cache}, target KV cache T_{cache}, and past token KV cache P_{cache} as None.
Initialize *action=read*, *is_finished=false*, and $index = 0$.
Initialize *next_token* as the target prompt tokens, and initialize generated tokens for this round *token_list* as an empty list.
while *is_finished* is *false* **do**:
    if *action* is *read*:
        Separate P_{cache} to S_{cache} and T_{cache}.
        Read prompt and $k + index$ words, and save hidden state to source KV cache S_{cache}.
        Merge S_{cache} and T_{cache} to P_{cache}.
        Set *action=write*.
        Set $index = index + 1$.
    elif *action* is *write*:
        Separate P_{cache} to S_{cache} and T_{cache}.
        Calculate and save hidden state to target KV cache T_{cache}.
        Merge S_{cache} and T_{cache} to P_{cache}.
        Project *next_token* as Q, and calculate attention output with KV cache P_{cache}.
        Predict and update the *next_token* based on greedy decoding.
        if *next_token* is the end symbol:
            Set *is_finished* as *true*.
        Add *next_token* to *token_list*.
        if *token_list* forms a complete word:
            Print the word.
            Set *action* as *read*, reset *token_list* as an empty list.
end while
Return: The predict words.

D More Details about Group Position

D.1 Relative Distance of Group Position

Let the source tokens be $X = [x_0, x_1, \ldots, x_{M-1}]$ and target tokens be $Y = [y_0, y_1, \ldots, y_{N-1}]$, where the position IDs are $pos_x = [0, 1, \ldots, M-1]$ and $pos_y = [\phi, \phi+1, \ldots, \phi+N-1]$, respectively. In batch-processing mode, the starting position ID on the target side is given by $\phi = M$. In batch-streaming

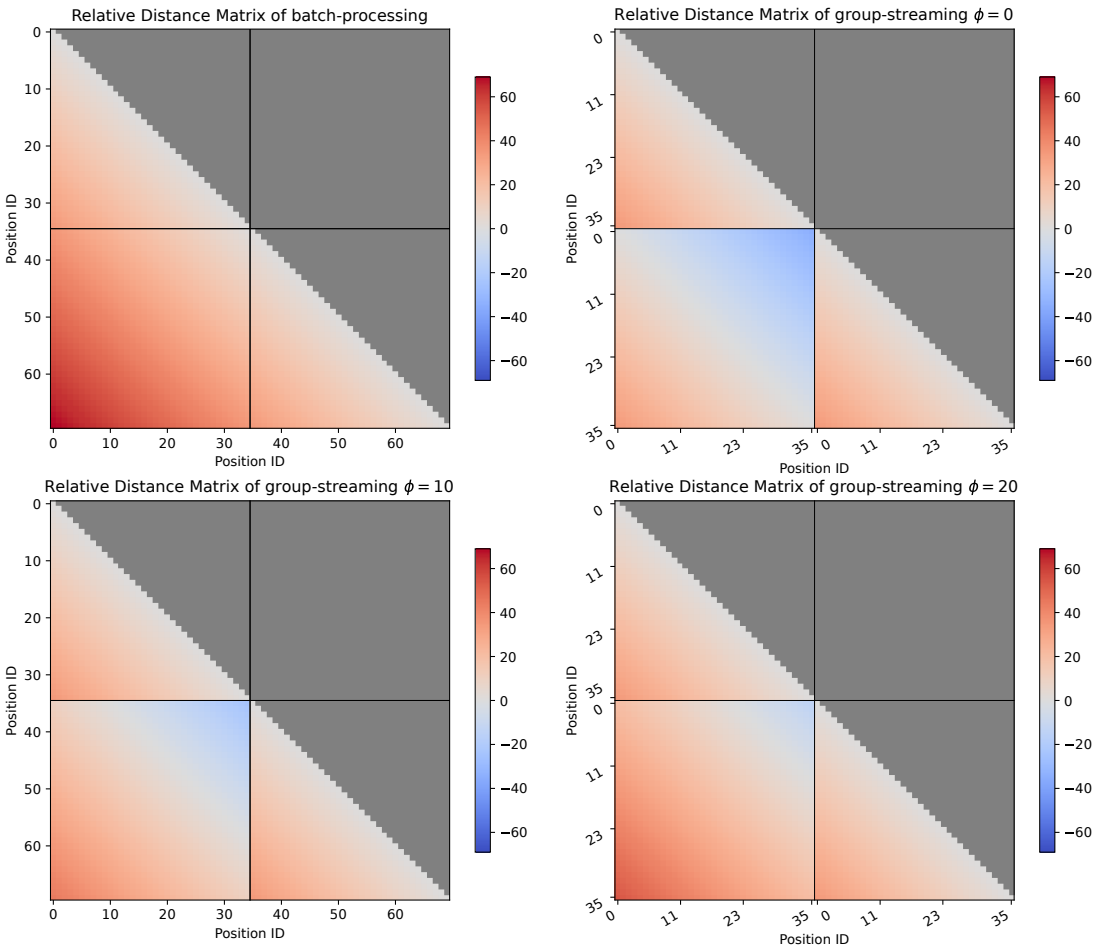


Figure 6: Relative distance matrix of batch-processing mode and group-streaming mode.

mode, the starting position ID on the target side is given by $\phi = 0$.

Define the rotary matrix as $R(m) = diag(R_1(m), R_2(m), \ldots, R_{d/2}(m))$, where m is the position id, d is the model dimension, and

$$R_i(m) = \begin{bmatrix} \cos(m\theta_i) & -\sin(m\theta_i) \\ \sin(m\theta_i) & \cos(m\theta_i) \end{bmatrix}, \quad \theta_i = 10000^{-2i/d}. \tag{1}$$

For the original rotary position embedding (RoPE) (Su et al., 2024), positional information is incorporated into each token's Query (q) and Key (k) through a rotation matrix. This process can be expressed as $q_n^r = R(n)q_n$ and $k_m^r = R(m)k_m$, where n and m denote the respective position IDs. Then, the attention mechanism in RoPE incorporates the rotationally transformed queries and keys, leading to the attention score computation as follows:

$$Attention(n, m) = q_n^{r\,T} k_m^r = q_n^T R^T(n) R(m) k_m = q_n^T R(m-n) k_m. \tag{2}$$

For any two positions n and m within the sequence, their position encoding depends solely on $R(m-n)$, meaning it is determined by their relative distance $m - n$. When k_m and q_n both belong to either source tokens or target tokens, the relative distance is given by $\Delta = m - n$. In this case, the positional encoding results in batch-processing and batch-streaming remain identical. When k_m and q_n belong to source tokens and target tokens, respectively, the relative distance is given by $\Delta = \phi + j - m$, where j denotes the position of q_n as the j-th token on the target side. In this case, the positional encoding results in batch-processing and batch-streaming depend on ϕ.

For batch processing, $\phi = M - 1$ indicates that the target tokens are farther from the source starting token and closer to the source ending token. **In contrast, for batch-streaming, $\phi = 0$ means the target**

starting token is closer to the source starting token and farther from the source ending token. This aligns with the sequential information arrival order in streaming scenarios, making it more suitable for capturing relative positional changes in streaming settings.

Figure 6 illustrates the variation in relative distances under batch-processing and batch-streaming settings. In batch-processing mode, which is typically used in offline scenarios, position IDs are assigned sequentially. Tokens near the diagonal exhibit local positional relationships with smaller relative distances, whereas tokens farther from the diagonal have increasingly larger relative distances, reflecting their positional separation. In batch-streaming mode, the relative positional relationships among tokens within the source and target sequences remain unchanged. However, the relative distance between target and source tokens is influenced by the parameter ϕ, shifting accordingly as ϕ increases. Taking $\phi = 0$ as an example, in a streaming scenario, the target token with position ID 0 first interacts with the source token at position ID 0, resulting in a relative distance of 0. This alignment effectively models the sequential nature of data accumulation in streaming settings, ensuring that the position encoding adapts dynamically to the progressive arrival of information.

D.2 Why Group Position Avoids Confusion

Research by (Shen et al., 2018; Haviv et al., 2022; Tsai et al., 2019) have shown that decoder-only models can learn implicit positional information. In decoder-only architectures, source tokens and target tokens attend to different contexts. As a result, even if their position IDs overlap, the model can still distinguish between source and target based on the content they attend to. As shown in Equation 3:

$$
\begin{aligned}
Attention(n, cache) &= \sum_{i=0} {q_n^r}^T {k_i}^r = \sum_{i=0} q_n^T R^T(n) R(i) k_i = \sum_{i=1} q_n^T R(i-n) k_i \\
&= \begin{cases} \sum_{i=0}^{n} q_{n_s}^T R(i-n) k_{i_s}, & q_{n_s}\ is\ source, \\ \sum_{i=0}^{M-1} q_{n_t}^T R(i-n) k_{i_s} + \sum_{i=0}^{n} q_{n_t}^T R(i-n) k_{i_t}, & q_{n_t}\ is\ target. \end{cases}
\end{aligned} \tag{3}
$$

In both cases, the query token has an ID of n, but since it attends to different content, the model can still distinguish between the source and the target.

D.3 Potential Edge of Group Position

The model can learn the position offset ϕ through simple fine-tuning, so typical values of ϕ do not significantly impact performance. However, when ϕ becomes extremely large, it may lead to discrepancies with the model's pretraining distribution due to the limited context length used during pretraining. Therefore, a reasonable range for ϕ should ensure that the maximum relative distance, specifically, between the last target token and the first source token, does not exceed the model's pretraining context length. For example, Gemma2-2B-Instruct was pretrained with a context length of 8k, which suggests that the maximum suitable value of ϕ is around 6k, as shown in Table 2.

Model	Wait-k	m=0	m=512	m=4k	m=5k	m=6k	m=7k	m=8k	m=10k	m=50k
Gemma2-2B-Instruct (8k)	5	**40.76**	40.68	40.70	40.51	**40.21**	39.83	39.73	39.52	39.37
	9	**40.91**	40.89	40.85	40.81	**40.73**	40.11	39.97	39.78	39.56

Table 2: BLEU performance of Gemma2-2B-Instruct (8k) under different memory sizes m and wait-k settings.

E Full Results of Text Translation Task

This section provides additional results to validate the impact of different initial position IDs on the target side in streaming translation tasks. The results cover three different large language models and two different translation tasks. The full results of the text translation task are shown in Table 3, which includes the accuracy metric BLEU and the latency metric LAAL.

Dataset	Wait-k	Gemma2-2b-Instruct (Target start id ϕ)					
		0	0.5	128	256	512	Δ
En-Fr	5	40.76 (5.21)	40.76 (5.21)	40.70 (5.21)	40.57 (5.20)	40.68 (5.21)	0.19 (0.01)
	7	40.92 (7.18)	40.92 (7.18)	40.85 (7.17)	40.91 (7.18)	40.92 (7.18)	0.07 (0.01)
	9	40.91 (9.14)	40.91 (9.14)	40.90 (9.13)	40.88 (9.13)	41.01 (9.13)	0.09 (0.01)
	11	41.10 (11.09)	41.10 (11.09)	41.14 (11.09)	40.96 (11.09)	41.05 (11.09)	0.18 (0.00)
En-De	5	30.84 (4.62)	30.84 (4.62)	30.90 (4.63)	30.80 (4.62)	30.95 (4.56)	0.15 (0.07)
	7	31.47 (6.63)	31.47 (6.63)	31.44 (6.63)	31.57 (6.63)	31.67 (6.59)	0.23 (0.04)
	9	31.73 (8.66)	31.73 (8.66)	31.87 (8.65)	31.91 (8.65)	31.88 (8.65)	0.18 (0.01)
	11	31.95 (10.70)	31.95 (10.70)	31.98 (10.69)	31.95 (10.69)	31.89 (10.69)	0.09 (0.01)
Dataset	**Wait-k**	**Phi3-mini-Instruct** (Target start id ϕ)					
		0	0.5	128	256	512	Δ
En-Fr	5	39.89 (5.45)	39.89 (5.45)	39.91 (5.44)	40.06 (5.41)	39.87 (5.44)	0.19 (0.03)
	7	40.57 (7.38)	40.57 (7.38)	40.53 (7.37)	40.72 (7.39)	40.71 (7.39)	0.19 (0.02)
	9	41.31 (9.28)	41.31 (9.28)	41.04 (9.29)	41.35 (9.27)	41.44 (9.27)	0.20 (0.02)
	11	41.92 (11.17)	41.92 (11.17)	42.03 (11.17)	41.94 (11.17)	41.93 (11.17)	0.11 (0.00)
En-De	5	30.92 (4.65)	30.92 (4.65)	30.76 (4.64)	30.81 (4.65)	30.86 (4.65)	0.16 (0.01)
	7	31.94 (6.65)	31.94 (6.65)	31.78 (6.64)	31.84 (6.64)	31.78 (6.64)	0.16 (0.01)
	9	32.18 (8.69)	32.18 (8.69)	32.10 (8.68)	32.21 (8.69)	32.09 (8.68)	0.12 (0.01)
	11	32.26 (10.71)	32.26 (10.71)	32.23 (10.73)	32.23 (10.73)	32.28 (10.73)	0.10 (0.02)
Dataset	**Wait-k**	**LLaMA3.1-8b-Instruct** (Target start id ϕ)					
		0	0.5	128	256	512	Δ
En-Fr	5	40.11 (5.23)	40.11 (5.23)	40.10 (5.22)	39.93 (5.23)	39.92 (5.23)	0.19 (0.01)
	7	40.30 (7.19)	40.30 (7.19)	40.32 (7.19)	40.35 (7.20)	40.31 (7.19)	**0.03** (0.01)
	9	40.15 (9.17)	40.15 (9.17)	40.32 (9.16)	40.34 (9.17)	40.35 (9.17)	0.20 (0.01)
	11	40.53 (11.11)	40.53 (11.11)	40.47 (11.11)	40.58 (11.11)	40.63 (11.10)	0.16 (0.01)
En-De	5	30.33 (4.58)	30.33 (4.58)	30.21 (4.57)	30.37 (4.58)	30.34 (4.58)	0.16 (0.01)
	7	31.23 (6.54)	31.23 (6.54)	31.18 (6.54)	31.16 (6.54)	31.25 (6.53)	0.09 (0.01)
	9	31.80 (8.63)	31.80 (8.63)	31.83 (8.63)	31.76 (8.62)	31.89 (8.62)	0.13 (0.01)
	11	32.04 (10.56)	32.04 (10.56)	31.98 (10.55)	32.07 (10.56)	32.08 (10.56)	0.10 (**0.00**)

Table 3: Performance comparison of different models with various wait-k policies and target start IDs. Δ represents the range of variation in BLEU scores and LAAL scores when the start id of target token takes different values. We use bold to indicate the smallest variation. Underline represents the largest variation.

F Model Efficiency

This section compares the computational cost and throughput between re-encoding and our grouped-streaming approach. We conduct a case study on the En–Fr streaming translation task using a filtered subset of the dataset that contains 7.3k sentence-level examples with controlled lengths, amounting to approximately 32k tokens.All experiments are conducted using the Phi-3 Mini Instruct model. Table 4 summarizes the inference time and throughput under different Wait-k settings, highlighting the drastic efficiency gains brought by removing re-encoding.

The results in the figure show that the proposed grouped-streaming paradigm eliminates the need for re-encoding, resulting in significant throughput improvements: over 5 × speedup under the wait-9 setting and more than 11 × speedup under wait-5 setting, compared to the re-encoding baseline.

Wait-k	Inference mode	Time consumption		Throughput	
5	with re-encoding	59.54 h		1.79 tokens/s	
	without re-encoding	**4.38 h**	**↓ 92.6%**	**20.24 tokens/s**	**× 11.3**
9	with re-encoding	28.87 h		3.70 tokens/s	
	without re-encoding	**4.04 h**	**↓ 86.1%**	**21.93 tokens/s**	**× 5.9**

Table 4: Comparison of inference efficiency under different Wait-k values and re-encoding modes.

G Visualization

G.1 Attention Distribution

Figure 7 illustrates the absolute values of the attention matrix, representing the attention magnitude of target tokens to both the input and output. In the left figure, the most attended column corresponds to the attention sink (Xiao et al., 2024), whereas in the right figure, the attention sink has been removed. The absolute attention map highlights each token's attention to historical tokens but makes it difficult to assess how different tokens distribute their attention toward a specific token. To better emphasize the distribution of target tokens' attention toward a given token, we normalize the attention map along columns and apply a gamma transformation to enhance and amplify the relationships. Mathematically, given an attention matrix A, where $A_{i,j}$ represents the attention weight from token j to token i, we normalize each column as follows:

$$A'_{i,j} = \left(\frac{A_{i,j} - \min_i\{A_{i,j}\}}{\max_i\{A_{i,j}\} - \min_i\{A_{i,j}\}} \right). \tag{4}$$

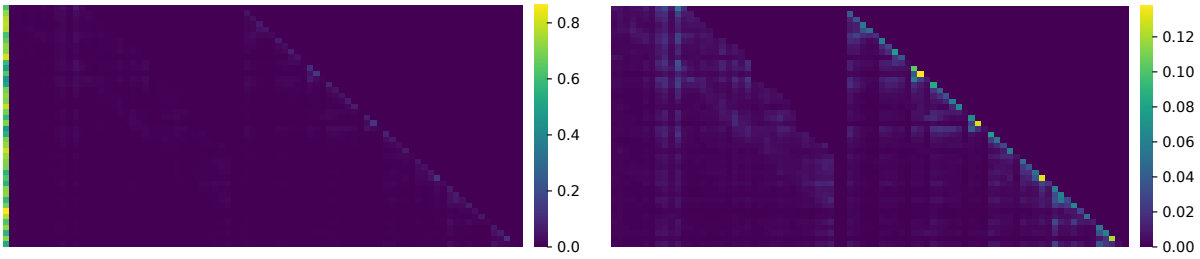


Figure 7: The absolute values of the attention matrix, with the left figure incorporating the attention sink, while the right figure depicts the matrix after the removal of the attention sink.

G.2 Example of Streaming Decoding

Figure 8 is an example of streaming reading and decoding process.

Input:	'<bos><start_of_turn>userTranslate the following English paragraph to German\n**Just in the last two**'
Output:	'**<start_of_turn>Erst**'
Input:	'<bos><start_of_turn>userTranslate the following English paragraph to German\nJust in the last two **days,**'
Output:	'<start_of_turn>Erst **in**'
Input:	'<bos><start_of_turn>userTranslate the following English paragraph to German\nJust in the last two days, **we**'
Output:	'<start_of_turn>Erst in **den**'
Input:	'<bos><start_of_turn>userTranslate the following English paragraph to German\nJust in the last two days, we **got**'
Output:	'<start_of_turn>Erst in den **letzten**'
Input:	'<bos><start_of_turn>userTranslate the following English paragraph to German\nJust in the last two days, we got **the**'
Output:	'<start_of_turn>Erst in den letzten **beiden**'
Input:	'<bos><start_of_turn>userTranslate the following English paragraph to German\nJust in the last two days, we got the **new**'
Output:	'<start_of_turn>Erst in den letzten beiden **Tagen**'
Input:	'<bos><start_of_turn>userTranslate the following English paragraph to German\nJust in the last two days, we got the new **temperature**'
Output:	'<start_of_turn>Erst in den letzten beiden Tagen **hatten**'
Input:	'<bos><start_of_turn>userTranslate the following English paragraph to German\nJust in the last two days, we got the new temperature **records**'
Output:	'<start_of_turn>Erst in den letzten beiden Tagen hatten **neue**'
Input:	'<bos><start_of_turn>userTranslate the following English paragraph to German\nJust in the last two days, we got the new temperature records **in**'
Output:	'<start_of_turn>Erst in den letzten beiden Tagen hatten neue **Januar-**'
Input:	'<bos><start_of_turn>userTranslate the following English paragraph to German\nJust in the last two days, we got the new temperature records in **January**'
Output:	'<start_of_turn>Erst in den letzten beiden Tagen hatten neue Januar-**Temperaturrekorde**'
Input:	'<bos><start_of_turn>userTranslate the following English paragraph to German\nJust in the last two days, we got the new temperature records in January.**<end_of_turn>**'
Output:	'<start_of_turn>Erst in den letzten beiden Tagen hatten neue Januar-Temperaturrekorde.**<end_of_turn>**'

Figure 8: An example on wait-5 reading/writing policy. The bold indicate the most recently content.