NPUsper: Eliminating Redundant Computation for Real-Time Whisper on Mobile NPUs

Hojeong Lee[1,*] **Sihyeon Lee**[2,*] **Sungwon Woo**[2]
Chengpo Yan[1] **Suman Banerjee**[1] **Seyeon Kim**[2]
[1]University of Wisconsin–Madison [2]Korea University
hojeong.lee@wisc.edu xl6047@korea.ac.kr sungwonwoo@korea.ac.kr
cyan46@wisc.edu suman@cs.wisc.edu seyeon625@korea.ac.kr

Abstract

We present *NPUsper*[1], a live transcription system that makes Whisper efficient on mobile NPUs by eliminating redundant computation. To avoid the heavy padding used by prior streaming systems, *NPUsper* detects hallucinated tokens online from temporal patterns in decoder cross-attention, allowing each inference round to process short audio inputs with minimal carryover. For efficient mobile-NPU execution, we propose controlled unrolling, which executes autoregressive decoding as K-step chunk graphs, removing unnecessary KV-cache computation and reducing graph-dispatch overhead. *NPUsper* achieves up to **4.84**$\times$ lower per-word latency, up to **33.2**$\times$ lower time-to-first-token (TTFT), and up to **88.64%** lower average power consumption compared with baselines, while maintaining comparable transcription accuracy. The code is available at `https://github.com/npusper/NPUsper`.

1 Introduction

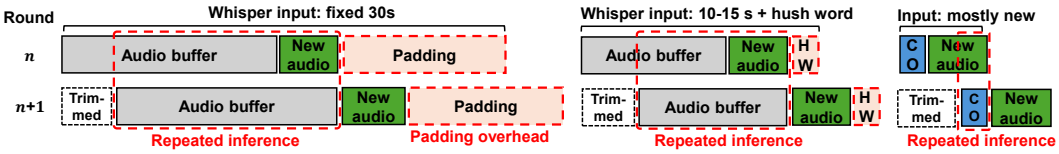
On-device AI inference is becoming increasingly important for mobile applications due to its advantages in privacy, latency, and reliability. Among these applications, real-time speech transcription has emerged as a key capability for intelligent assistants and accessibility services, where timely and accurate transcription is critical to user experience [Apple, Labiausse et al., 2025]. Enabling such functionality directly on mobile devices is particularly desirable, as it avoids the latency and connectivity limitations of cloud-based processing while preserving user privacy.

Recent systems increasingly rely on large foundation models for robust transcription, and Whisper [Radford et al., 2023] has emerged as a *de facto* standard due to its strong accuracy and scalability. As a result, there is growing interest in running Whisper directly on mobile devices for real-time, on-device transcription [Macháček et al., 2023, Wang et al., 2025, 2024, Macháček and Polák, 2025]. To meet the stringent latency and energy requirements of real-time processing, modern mobile platforms are equipped with Neural Processing Units (NPUs) that offer high-throughput and energy-efficient inference. However, fully leveraging mobile NPUs for real-time Whisper remains challenging, due to fundamental mismatches between Whisper's streaming inference pipeline and the execution characteristics of NPUs.

First, existing streaming approaches for Whisper incur substantial redundant computation. Whisper is prone to hallucinations when processing short audio inputs that deviate from its 30-second training distribution [Wang et al., 2025]. To mitigate this, prior systems rely on padding and overlapping audio buffers, effectively reprocessing previously seen inputs to stabilize decoding as illustrated

[*]Equal contribution.
[1]*NPUsper* combines "NPU" and "Whisper," with "NP" also referring to no padding to avoid padding overhead.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).



(a) Whisper-Streaming, SimulStreaming, and Simul-Whisper (b) WhisperFlow (c) Ours

Figure 1: Redundant computation in existing systems. Existing systems construct each Whisper input by buffering previously processed audio with newly arrived audio and padding it to a fixed 30-second window to mitigate hallucinations. WhisperFlow [Wang et al., 2025] reduces padding overhead by replacing long padding with a trained short audio segment, called a hush word, but still retains substantial overlap between consecutive inference inputs. HW denotes hush word and CO denotes audio carryover.

in Figure 1. However, this design treats padding as a necessity, even though its primary role is to suppress hallucinations rather than to improve acoustic coverage. As a result, a large fraction of computation is spent on redundant or artificial inputs, while newly arrived speech accounts for only a small portion of each inference step.

Second, Whisper's autoregressive decoding is poorly matched to the static execution model of mobile NPUs. NPUs rely on precompiled static graphs with fixed tensor shapes, whereas decoding involves dynamically growing key-value (KV) caches. Existing approaches either execute attention over the maximum sequence length at every decoding step, incurring substantial redundant computation, or rely on step-specific graphs, which introduce significant runtime overhead due to frequent graph dispatch.

To address these challenges, we present *NPUsper*, a system for real-time on-device Whisper transcription on mobile NPUs. Our key insight is that the inefficiency of streaming Whisper stems from two structural mismatches: **(i)** hallucination-induced padding overhead caused by unstable temporal grounding under short inputs, and **(ii)** the mismatch between dynamic autoregressive decoding and static-graph NPU execution. *NPUsper* resolves these mismatches through temporal-grounding-aware decoding control and NPU-aware controlled unrolling.

We implement and evaluate *NPUsper* on commercial mobile devices [Samsung, Qualcomm] and show that it achieves up to **4.84**$\times$ lower per-word latency, up to **33.2**$\times$ lower time-to-first-token (TTFT), and up to **88.64%** lower average power consumption compared with baselines, while maintaining comparable transcription accuracy.

2 Background and Related Work

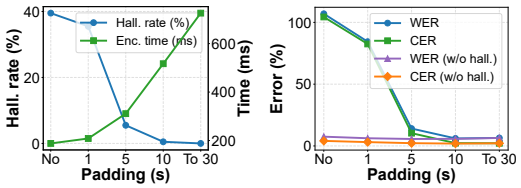
2.1 Live Transcription Systems with Whisper

Whisper. Whisper [Radford et al., 2023] is an encoder–decoder Transformer model widely used for speech transcription due to its strong accuracy and robustness. Although originally designed for offline inference, it has been increasingly adopted in live transcription systems through buffering and specialized decoding strategies [Macháček et al., 2023, Wang et al., 2025, 2024, Macháček and Polák, 2025].

Live transcription systems. Processing continuously arriving speech in real time requires balancing low latency with transcription accuracy [Yu and Deng, 2016]. To achieve this, existing systems repeatedly invoke Whisper on partial inputs and rely on decoding policies such as Local Agreement [Liu et al., 2020] and AlignAtt [Papi et al., 2023] to determine when partial hypotheses are stable enough to emit.

2.2 Neural Processing Units

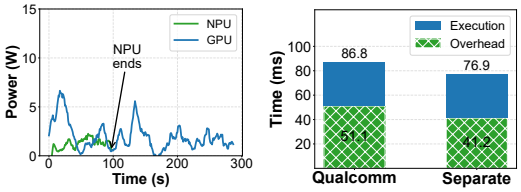
Neural processing units (NPUs) execute models using precompiled static graphs with fixed tensor shapes, enabling high efficiency [Samsung, Meta, Qualcomm] but limiting support for dynamic workloads such as autoregressive decoding.



(a) Padding vs. hallucination/overhead

(b) Padding vs. accuracy

Figure 2: Padding reduces hallucination and improves accuracy, but increases processing overhead.



(a) Power and throughput

(b) Decoding latency breakdown on NPU

Figure 3: NPUs provide faster and more energy-efficient inference, while static-graph execution inefficiency motivates NPU-aware decoding.

Static-graph execution. To exploit these hardware characteristics, NPUs typically execute models using precompiled static graphs, where tensor shapes, memory layouts, and execution schedules are fixed. This enables efficient mapping of operations onto parallel compute units, improves data reuse through on-chip SRAM buffering, and reduces costly memory movement. As a result, static-graph execution allows NPUs to achieve high utilization of compute resources while minimizing runtime overhead and energy consumption [Lee, 2021, Hameed et al., 2010].

Implications for live transcription. This execution model is well-suited to workloads with fixed or predictable tensor shapes, such as vision models with static input resolutions [Jung et al., 2025, Tan and Cao, 2021]. In contrast, live transcription involves continuously arriving audio inputs and incremental decoding, which naturally introduce variability in input sizes and execution patterns [Yu and Deng, 2016]. As a result, efficiently mapping such workloads onto static-graph execution often requires structured input handling or approximation strategies that align dynamic inputs with a limited set of predefined execution configurations [Khomenko et al., 2016].

3 Motivation

3.1 Computational Cost of Padding and Hallucination Risk

Whisper hallucination. Whisper exhibits hallucination behavior, generating text that is not grounded in the input audio [Kalai et al., 2025, Barański et al., 2025, Koenecke et al., 2024]. In speech transcription, this typically manifests as repeated words or phrases [Koenecke et al., 2024, Wang et al., 2025]. As shown in Figure 2(a), hallucinations become more frequent when processing short audio inputs without padding. To mitigate this, existing systems extend the input with padding toward the 30-second training window, which stabilizes decoding. Correspondingly, Figure 2(b) shows that the word error rate (WER) and character error rate (CER) decrease as the input length approaches 30 seconds. This behavior stems from Whisper being trained on fixed 30-second audio segments [Radford et al., 2023, Wang et al., 2025], indicating that padding primarily serves to suppress hallucinations rather than to provide additional acoustic context.

Padding overhead. While padding mitigates hallucinations and improves WER/CER, it incurs significant computational overhead, especially on resource-constrained mobile devices. As shown in Figure 2(a), the encoder processing time increases substantially as the padded input length approaches 30 seconds, reaching around 720 ms, which is 3.8× higher than that without padding. The smaller gap observed in Figure 2(a) is due to the longer input duration in this experiment (approximately 9 seconds), which exceeds the shorter segments typically used in live transcription. We focus on encoding latency in this motivation study because decoding latency depends on both input length and the number of generated tokens, making it difficult to isolate the effect of padding.

Opportunity. As shown in Figure 2(b), when hallucinated tokens generated beyond the reference transcription are removed in offline post-processing (w/o hall. in the figure), Whisper inference without padding achieves WER and CER close to those obtained with full 30-second padding. These observations suggest that hallucinations are fundamentally associated with failures in temporal grounding rather than insufficient acoustic context itself. Therefore, if temporally ungrounded tokens can be detected online during decoding, it becomes possible to eliminate padding overhead while

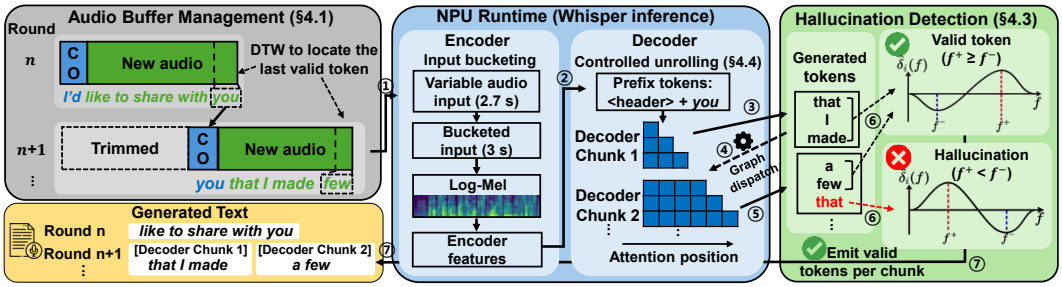


Figure 4: *NPUsper* overview.

preserving transcription accuracy. To realize this opportunity, we analyze cross-attention patterns to detect hallucinations and avoid unnecessary computation in *NPUsper*, detailed in § 4.3.

3.2 Dynamic Speech Processing on Static-graph NPUs

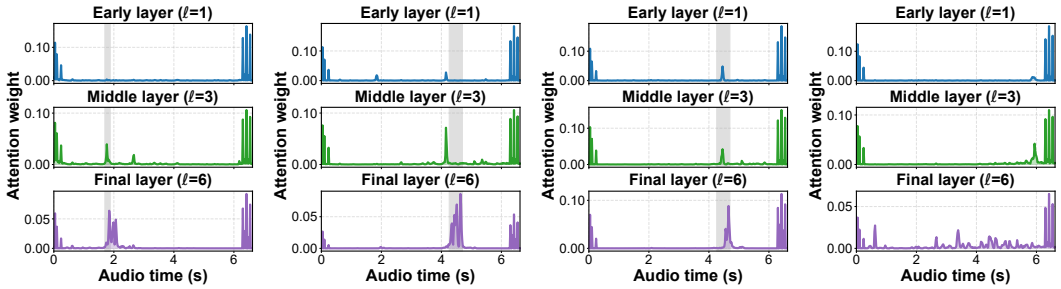
NPU execution mismatch. NPUs offer high inference throughput and energy efficiency, making them attractive for on-device live transcription systems. As shown in Figure 3(a), on the same 200 Whisper inference tasks, the NPU completes inference 3× faster than the GPU and reduces average power consumption from 1.99 W to 1.14 W. However, using NPUs for live transcription introduces an execution mismatch: input length and autoregressive decoding are dynamic, whereas NPUs rely on static graphs with fixed shapes and control flow. While variability in input length can be handled relatively easily using bucketing [Khomenko et al., 2016], autoregressive decoding is more challenging to execute efficiently on NPUs. Specifically, the KV cache grows with each generated token, creating a mismatch between dynamic autoregressive decoding and static-graph execution on NPUs.

KV-cache inefficiency. To enable autoregressive decoding on NPUs, Qualcomm adopts a design that fixes the encoder input to 30 seconds and allocates the KV cache for the maximum decoded sequence length, $M = 200$ [Qualcomm AI Hub]. We focus on the decoding-side inefficiency, as the encoding-side overhead from input padding was discussed in § 3.1. In autoregressive decoding, the KV cache grows incrementally with the number of generated tokens, so early decoding steps require only minimal attention computation. Under Qualcomm's design, however, the NPU must execute attention over all $M = 200$ positions at every decoding step, even when the actual sequence length is much shorter (Figure 6(a)). Similar static-shape designs are commonly used in existing mobile NPU LLM systems [Xu et al., 2025]. As a result, substantial redundant computation is incurred throughout decoding.

Runtime overhead. A straightforward way to reduce KV-cache inefficiency in Qualcomm's fixed-shape design is to compile separate step-specific graphs that match the current KV-cache length. However, this requires dispatching a different graph at each token-generation step, causing substantial runtime overhead (Figure 3(b)). Another alternative is to unroll n decoding steps into a single graph to amortize this overhead. Once dispatched, however, the graph must execute all n steps, even if EOS or a stop condition occurs earlier. This highlights the need for a balanced NPU decoding design that reduces redundant KV-cache computation, minimizes runtime overhead, and preserves early termination (§ 4.4).

4 *NPUsper* System Design

Figure 4 presents an overview of *NPUsper*. At each inference round, *NPUsper* constructs a short Whisper input from newly arrived audio and carry-over context (§ 4.1). The input is mapped to an NPU-compatible bucket, and decoding is executed with controlled unrolling (§ 4.4). Generated tokens are then validated using cross-attention dynamics (§ 4.2 and § 4.3). Valid tokens are emitted immediately, while hallucinated tokens stop decoding and determine the carry-over state for the next round.



(a) Token: the (b) Token: squ (c) Token: -alid (d) Token: .

Figure 5: Layer-wise cross-attention dynamics in Whisper over time for selected tokens from the utterance: "After early nightfall, the yellow lamps would light up here and there, the squalid quarter of the brothels." The shaded region indicates the ground-truth audio span of each token. The final decoder layer shows sharper temporal alignment than earlier layers. Content tokens form well-localized attention peaks within their spoken duration, whereas the punctuation token exhibits more diffuse cross-attention with no clear temporal anchor.

4.1 Audio Buffer Management

The audio buffer manager maintains the incoming audio stream and constructs each input from carry-over audio and newly arrived audio. Unlike prior systems that rely on padding, *NPUsper* constructs inputs using only real audio and validated context. The input is mapped to a discrete bucket for NPU execution. For autoregressive decoding, *NPUsper* initializes the decoder with prefix tokens derived from the carry-over text. *NPUsper* validates generated tokens using cross-attention and emits only valid tokens. When hallucination is detected, decoding stops and the carry-over state is updated using dynamic time warping (DTW) [Sakoe and Chiba, 1978].

4.2 Cross-attention Characteristics in Whisper

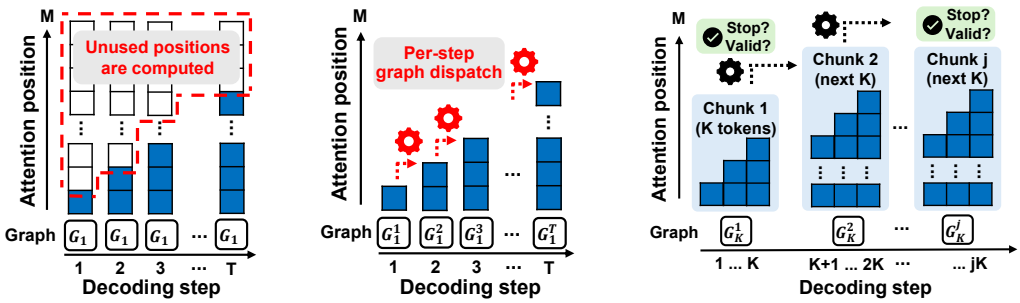
Definition of cross-attention. For the i-th generated token, we define the frame-wise cross-attention at decoder layer l as the head-averaged attention:

$$a_i^{(l)}(f) = \frac{1}{H} \sum_{h=1}^{H} \mathrm{CrossAttn}_i^{(l,h)}(f), \tag{1}$$

where H is the number of attention heads, f denotes the audio frame index, and $\mathrm{CrossAttn}_i^{(l,h)}(f)$ is the softmax-normalized cross-attention weight assigned to frame f at decoder layer l and head h when generating the i-th token. $a_i^{(l)}$ provides a token-level distribution over audio time, which we use to analyze temporal grounding.

Cross-attention analysis. Cross-attention in Whisper indicates which audio frames contribute to each generated token. We use this signal to analyze whether tokens are temporally grounded in the input audio. Figure 5 shows that content tokens exhibit localized attention that follows the temporal progression of the input audio. This property is not specific to Whisper, but arises from the alignment structure of encoder-decoder speech models [Papi et al., 2023, 2024, Waldendorf et al., 2026]. We hypothesize that valid tokens follow a monotonic progression over audio frames, whereas hallucinated tokens violate this temporal consistency. This suggests that cross-attention can serve as a proxy for token-level grounding. Punctuation tokens exhibit diffuse attention over the audio timeline, and subword tokens can introduce noise in temporal alignment. We therefore restrict our analysis to content tokens for reliable detection (§ 4.3).

Layer selection. The temporal structure of cross-attention also varies across decoder layers. Compared with earlier layers, the final decoder layer shows clearer temporal alignment with the audio timeline (Figure 5), making it more suitable for tracking token-level progression. We therefore use the final layer and compare cross-attention across consecutive content tokens.



(a) Fixed full-length graph (b) Step-specific graphs (c) Controlled unrolling (Ours)

Figure 6: Comparison of decoding strategies on NPUs. Fixed full-length graphs reuse the same graph G_1 while computing attention over the maximum KV-cache length M at every step. Step-specific graphs use G_1^r to match the KV-cache length required at each decoding step, but require graph dispatch for every generated token. Controlled unrolling uses G_K^r to generate K tokens per execution, reducing dispatch overhead while avoiding redundant KV-cache computation. Here, G_q^r denotes the r-th graph used in a decoding strategy, where q is the number of tokens generated by that graph.

4.3 Hallucination Detection

Based on observations and insights in § 4.2, we design a hallucination detection mechanism that identifies violations of temporal consistency in cross-attention patterns.

Backward cross-attention shift. *NPUsper* detects hallucinated tokens from cross-attention patterns in the final decoder layer. In Whisper, hallucinated tokens exhibit backward shifts in cross-attention over the audio timeline, rather than following the forward temporal progression of valid tokens.

Token validation. To operationalize this intuition, we define token-level temporal attention dynamics as follows. For each generated token i, we extract the head-averaged cross-attention from the final decoder layer L:

$$\tilde{a}_i(f) = a_i^{(L)}(f). \tag{2}$$

Temporal attention changes are computed only between consecutive content tokens, excluding punctuation and subword tokens to ensure reliable temporal alignment. Let $p(i)$ denote the most recent preceding content token before token i. We define the frame-wise attention difference as $\delta_i(f) = \tilde{a}_i(f) - \tilde{a}_{p(i)}(f)$. Let $\hat{\delta}_i(f)$ denote the filtered attention (Appendix A) difference between token i and its preceding content token. We identify the locations of the positive and negative peaks:

$$f_i^+ = \arg\max_f \hat{\delta}_i(f), \qquad f_i^- = \arg\min_f \hat{\delta}_i(f). \tag{3}$$

Intuitively, f_i^+ indicates the audio frame toward which attention newly concentrates, whereas f_i^- indicates the frame from which attention moves away. A token is considered valid if attention progresses forward along the audio timeline, i.e., $f_i^+ \geq f_i^-$. Otherwise, we mark it as hallucinated:

$$f_i^+ < f_i^-. \tag{4}$$

This condition captures violations of temporal monotonicity in cross-attention, which we treat as a signature of hallucination. This depends only on the relative ordering of attention dynamics and does not require threshold tuning.

Decoding control. During decoding, *NPUsper* stops generation when hallucination is detected or when attention reaches the end of the current audio chunk. In both cases, valid tokens are emitted, and the carry-over state is updated using DTW [Sakoe and Chiba, 1978]. This enables early termination of decoding and avoids unnecessary computation.

4.4 Controlled Unrolling for Decoding on NPUs

Controlled unrolling. Our system unrolls autoregressive decoding into K-step chunk graphs and executes them sequentially on the NPU. This design explicitly bridges the mismatch between dynamic autoregressive decoding and static-graph execution on NPUs. As shown in Figure 6(c), each chunk

graph generates up to K tokens in one NPU execution and returns the cross-attention weights defined in Equation (2). *NPUsper* then applies the token validation in § 4.3 to decide whether to emit tokens, stop decoding, or dispatch the next chunk graph. This enables adaptive computation that scales with the actual decoding length rather than the maximum sequence length.

Chunk-level decoding control. A K-token chunk serves as the unit of both NPU execution and decoding control. After each chunk, *NPUsper* applies the same decoding control conditions described in § 4.3 to decide whether to dispatch the next graph or proceed to the next inference round. Valid tokens are emitted after each chunk, rather than waiting for the entire decoding round to finish. If a hallucinated token is detected, *NPUsper* emits only the preceding valid tokens. This chunk-level design reduces graph dispatch overhead while preserving timely token emission and hallucination handling.

Necessary KV-cache computation. Controlled unrolling keeps only the KV-cache attention computation needed at each decoding step. Let M be the maximum decode length of the fixed full-length graph, P the prefix length at the start of decoding, and $\mathrm{MAC}(s)$ the number of multiply-accumulate operations for self-attention at decoding step s. We use c to denote the per-position self-attention cost, including the query-key dot product and value aggregation across all decoder layers and attention heads. The fixed full-length graph computes attention over all M positions at every step, so $\mathrm{MAC}_{\text{fixed}}(s) = cM$. In contrast, *NPUsper* computes attention only over the $P + s$ positions at decoding step s, so $\mathrm{MAC}_{\text{ours}}(s) = c(P + s)$. If decoding stops after generating T tokens, the resulting self-attention MAC ratio is

$$\frac{\sum_{s=1}^{T} \mathrm{MAC}_{\text{fixed}}(s)}{\sum_{s=1}^{T} \mathrm{MAC}_{\text{ours}}(s)} = \frac{2M}{2P + T + 1}. \tag{5}$$

This reveals a fundamental inefficiency in fixed-length decoding, where computation scales with the maximum sequence length M rather than the actual decoding length T. This inefficiency becomes particularly pronounced in live transcription, where decoding often terminates after only a few tokens. For example, with $M = 200$ and an empty prefix ($P = 0$), stopping at $T = 10$ yields a $36.4\times$ reduction in self-attention MACs, while $T = 20$ still yields a $19\times$ reduction.

5 Results

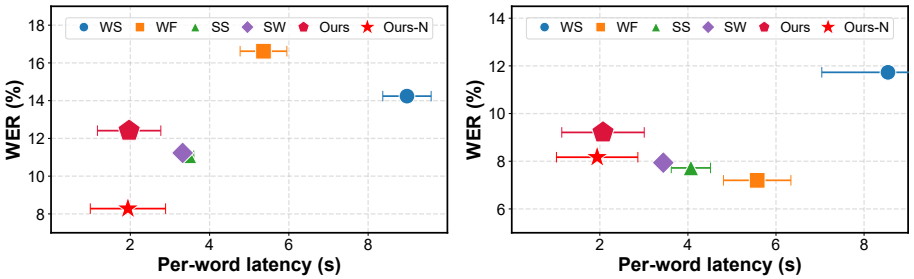
5.1 Evaluation Setup

We implemented and evaluated *NPUsper* and baseline systems on two commercial mobile devices: a Samsung Galaxy S25 [Samsung] and a Snapdragon X Plus laptop [Qualcomm], hereafter referred to as S25 and X Plus, respectively. For a fair comparison, all baselines and our method run on the GPU. We additionally evaluate our method on the NPU, denoted as Ours-N, to measure the benefit of NPU execution. Appendices C and D provide implementation details.

Baselines. We compare *NPUsper* with four Whisper-based live transcription systems: Whisper-Streaming (WS) [Macháček et al., 2023], WhisperFlow (WF) [Wang et al., 2025], SimulStreaming (SS) [Macháček and Polák, 2025], and Simul-Whisper (SW) [Wang et al., 2024]. They differ in Whisper input construction, decoder context management, and decoding policy. WS, SS, and SW feed Whisper with 30-second sliding-window inputs, whereas WF reduces the input to roughly 15 seconds using a trained hush word. For decoding, WS and WF use Local Agreement [Liu et al., 2020], which repeatedly confirms partial hypotheses before emission, while SS and SW use AlignAtt [Papi et al., 2023] to stop decoding based on cross-attention near chunk boundaries. For decoder context, SS retains tokens from dropped audio segments, whereas SW discards them and uses a trained integrate-and-fire (IF) module to detect word truncation [Dong and Xu, 2020].

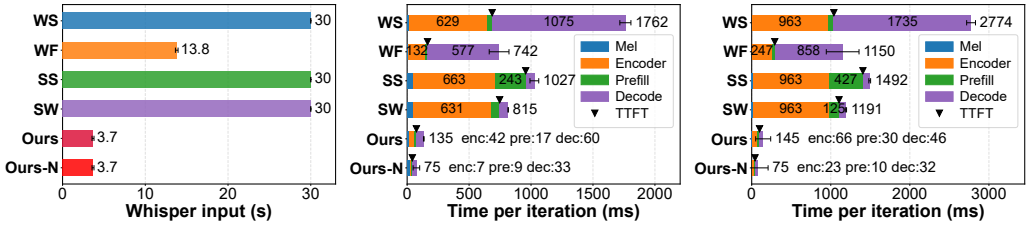
Dataset. We evaluate on two long-form datasets, TED-LIUM 3 [Hernandez et al., 2018] and Meanwhile [Radford et al., 2023]. Because each end-to-end system evaluation runs in real time with the input audio, the main results use two representative TED-LIUM 3 talks, EricMead (7:39 min) and GaryFlake (5:45 min). Additional results on the remaining TED-LIUM 3 talks and Meanwhile dataset are available in Appendix B.

Metrics. Per-word latency (↓) is the average latency between when a word is spoken and when it appears as text to the user [Macháček et al., 2023, Wang et al., 2025]. Word error rate (WER, ↓) is the fraction of incorrectly transcribed words relative to the reference transcript [Radford et al., 2023,



(a) Samsung Galaxy S25 (b) Snapdragon X Plus

Figure 7: Per-word latency and WER comparison. *NPUsper* achieves the lowest per-word latency on both devices with comparable transcription accuracy. Note that a 1 pp WER gap corresponds to only one additional word error per 100 reference words.



(a) Average Whisper input length (b) Samsung Galaxy S25 (c) Snapdragon X Plus

Figure 8: Latency breakdown. *NPUsper* delivers the best latency performance by avoiding redundant computation through hallucination detection during decoding (§ 4.3).

Zhang et al., 2020]. Time-to-first-token (TTFT, ↓) is the delay from the start of inference to the first text output visible to the user [Kudlur et al., 2026].

5.2 Latency and Inference Efficiency

Low-latency transcription. As shown in Figure 7, *NPUsper* achieves the lowest per-word latency with comparable transcription accuracy, with 1.71–4.62× and 1.69–4.84× lower per-word latency than the baselines on S25 and X Plus, respectively. This gain comes from eliminating redundant computation in padded and repeated Whisper inputs, which reduces inference time, and from using AlignAtt [Papi et al., 2023] to emit tokens from a single inference.

Reduced inference cost. Figure 8 shows that *NPUsper* runs Whisper inference 5.52–13.09× and 7.92–19.10× faster than the baselines on S25 and X Plus. *NPUsper* achieves this by avoiding unnecessary padding and processing only newly arrived audio with minimal carryover, resulting in shorter Whisper inputs and lower encoder latency. *NPUsper* also reduces prefill latency by using only the last valid word as the decoder prefix. Appendix B.4 presents the component-wise ablation.

Fast transcript updates. By avoiding redundant computation during each inference, *NPUsper* also reduces the time required to produce the first visible transcript update. On S25, *NPUsper* reduces TTFT by 2.2–12.8× compared with the baselines, while Ours-N further improves this to 3.9–22.8×. On X Plus, *NPUsper* reduces TTFT by 2.9–14.2×, and Ours-N achieves a 6.9–33.2× reduction. This improvement comes from jointly reducing computation cost, hypothesis recomputation, and unnecessary prefill computation.

5.3 Energy and Decoding Efficiency

We measure power on the S25 using a Monsoon High Voltage Power Monitor [Monsoon Solutions, Inc.] and on the Snapdragon X Plus using Qualcomm Profiler [Qualcomm Technologies, Inc.], reporting power after subtracting idle power. (Detailed in Appendix B.2)

Table 1: Energy and decoding efficiency. Power and energy metrics subtract base power; throughput is active-inference tokens/s.

Dev.	Sys.	P (W,↓)	E/inf (J,↓)	E/tok (J,↓)	Tput (tok/s,↑)
S25	WS	4.05±0.10	13.95±1.23	0.26±0.01	29.20±1.10
	WF	3.28±0.09	8.28±0.27	0.15±0.01	83.80±2.90
	SS	3.04±0.28	6.33±0.62	0.61±0.04	10.10±0.80
	SW	2.94±0.01	6.13±0.03	0.56±0.01	13.10±0.20
	Ours	0.56±0.04	1.16±0.09	0.13±0.01	64.40±0.20
	Ours-N	**0.46±0.02**	**0.92±0.05**	**0.10±0.01**	**112.90±0.60**
X Plus	WS	4.73±1.50	16.90±5.37	0.30±0.10	18.02±0.18
	WF	4.82±1.37	11.21±3.18	0.21±0.06	50.75±1.26
	SS	3.27±1.43	6.77±2.95	0.64±0.28	7.25±0.18
	SW	2.82±1.66	5.82±3.42	0.52±0.31	9.41±0.13
	Ours	1.42±1.86	2.92±3.81	0.33±0.43	64.60±0.01
	Ours-N	**0.81±1.75**	**1.65±3.57**	**0.19±0.41**	**144.98±1.09**

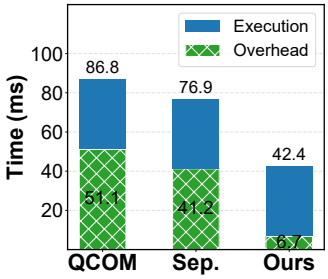


Figure 9: Decoding overhead analysis on S25.

Energy efficiency. Table 1 shows that *NPUsper* achieves the best energy efficiency on both devices. On S25, Ours-N reduces average power to 0.46 W and energy per inference to 0.92 J, achieving 6.3–8.8× lower power than the baselines. On X Plus, Ours-N also achieves the lowest power, energy per inference, and energy per token. This gain comes from eliminating redundant Whisper computation at the system level, while NPU execution provides additional energy savings. Additional results are provided in Appendix B.2.

Token-metric analysis. The energy-per-token and throughput metrics should be interpreted with each decoding policy in mind. WS and WF sometimes show competitive energy per token or throughput, not because they are more efficient, but because Local Agreement [Liu et al., 2020] regenerates many hypothesis tokens within each inference. Moreover, WS and WF failed to process the entire audio (see truncated power traces in Figure 11) because long Whisper inputs and repeated hypothesis recomputation increase per-iteration latency. Slow iterations delay longest-prefix confirmation in Local Agreement, causing the audio buffer to grow beyond what the systems can process in real time. By contrast, SS, SW, and *NPUsper* all use AlignAtt decoding [Papi et al., 2023], making their token-generation metrics more directly comparable. *NPUsper* nevertheless achieves the best energy-per-token and throughput overall.

Decoding efficiency. Figure 9 compares three NPU decoding strategies on S25. Qualcomm's fixed full-length graph incurs the highest latency due to excessive KV-cache computation. Using separate graphs removes this redundant KV-cache work but still suffers from per-token graph dispatch overhead. *NPUsper* reduces both costs by computing only the required KV-cache attention and reducing graph dispatch frequency through controlled unrolling. (Detailed in Appendix B.3.)

6 Conclusion and Limitations

We propose *NPUsper*, an on-device live transcription system that makes Whisper efficient on mobile NPUs by eliminating redundant computation on both audio processing and autoregressive decoding. *NPUsper* removes the need for long padded Whisper inputs by detecting hallucinated tokens online from cross-attention dynamics, allowing each inference round to process only newly arrived audio with minimal carry-over context. For NPU execution, *NPUsper* introduces controlled unrolling, which reduces redundant KV-cache attention computation and graph dispatch overhead while enabling chunk-level token emission and decoding control. *NPUsper* achieves the best per-word latency, inference speed, and energy efficiency among the evaluated live transcription systems.

Limitations. This paper focuses on system-level efficiency rather than modifying the Whisper architecture itself. Prior work [Krichli et al., 2025] converts Whisper's encoder into a causal encoder and applies low-rank adaptation [Hu et al., 2021], but the resulting model is limited to the dataset used for adaptation. We focus on building and evaluating a full end-to-end on-device system with the Whisper base model under mobile memory and compute constraints. However, our hallucination detection method is not limited to the base model, as it relies on temporal alignment in Whisper's decoder cross-attention. Appendix A.4 demonstrates its generalizability across different Whisper model sizes. We leave the design of efficient on-device systems using large speech models such as Canary [Sekoyan et al., 2025] and SeamlessM4T [Barrault et al., 2023] for future work.

Acknowledgments and Disclosure of Funding

This work was supported in part by the U.S. National Science Foundation under Awards 2625580, 2554552, 2504963, 2312716, 2231668, 2112562, and 2107060, and by the Institute of Information & Communications Technology Planning & Evaluation (IITP)-ICT Creative Consilience Program grant funded by the Korea government (MSIT) (IITP-2026-RS-2020-II201819). Seyeon Kim is the corresponding author.

References

Apple. Siri. `https://www.apple.com/siri/`.

Tom Labiausse, Laurent Mazaré, Edouard Grave, Alexandre Défossez, and Neil Zeghidour. High-fidelity simultaneous speech-to-speech translation. In *Forty-second International Conference on Machine Learning*, 2025.

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

Dominik Macháček, Raj Dabre, and Ondřej Bojar. Turning whisper into real-time transcription system. In *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 17–24, 2023.

Rongxiang Wang, Zhiming Xu, and Felix Xiaozhu Lin. Whisperflow: speech foundation models in real time. In *Proceedings of the 23rd Annual International Conference on Mobile Systems, Applications and Services*, pages 169–182, 2025.

Haoyu Wang, Guoqiang Hu, Guodong Lin, Wei-Qiang Zhang, and Jian Li. Simul-whisper: Attention-guided streaming whisper with truncation detection. *arXiv preprint arXiv:2406.10052*, 2024.

Dominik Macháček and Peter Polák. Simultaneous translation with offline speech and llm models in cuni submission to iwslt 2025. In *Proceedings of the 22nd International Conference on Spoken Language Translation (IWSLT 2025)*, pages 389–398, 2025.

Samsung. Galaxy S25. `https://www.samsung.com/us/smartphones/galaxy-s25/`.

Qualcomm. Snapdragon PCs & Laptops Product List. `https://www.qualcomm.com/laptops/products`.

Dong Yu and Lin Deng. *Automatic speech recognition*, volume 1. Springer, 2016.

Danni Liu, Gerasimos Spanakis, and Jan Niehues. Low-latency sequence-to-sequence speech recognition and translation by partial hypothesis selection. *arXiv preprint arXiv:2005.11185*, 2020.

Sara Papi, Marco Turchi, and Matteo Negri. Alignatt: Using attention-based audio-translation alignments as a guide for simultaneous speech translation. *arXiv preprint arXiv:2305.11408*, 2023.

Meta. Meta quest. `https://www.meta.com/quest/`.

Kyuho J Lee. Architecture of neural processing unit for deep neural networks. In *Advances in computers*, volume 122, pages 217–245. Elsevier, 2021.

Rehan Hameed, Wajahat Qadeer, Megan Wachs, Omid Azizi, Alex Solomatnikov, Benjamin C Lee, Stephen Richardson, Christos Kozyrakis, and Mark Horowitz. Understanding sources of inefficiency in general-purpose chips. In *Proceedings of the 37th annual international symposium on Computer architecture*, pages 37–47, 2010.

Chanyoung Jung, Jeho Lee, Gunjoong Kim, Jiwon Kim, Seonghoon Park, and Hojung Cha. Aria: Optimizing vision foundation model inference on heterogeneous mobile processors for augmented reality. In *Proceedings of the 23rd Annual International Conference on Mobile Systems, Applications and Services (MobiSys' 25)*, 2025.

Tianxiang Tan and Guohong Cao. Efficient execution of deep neural networks on mobile devices with npu. In *Proceedings of the 20th International Conference on Information Processing in Sensor Networks (Co-Located with CPS-IoT Week 2021)*, pages 283–298, 2021.

Viacheslav Khomenko, Oleg Shyshkov, Olga Radyvonenko, and Kostiantyn Bokhan. Accelerating recurrent neural network training using sequence bucketing and multi-gpu data parallelization. In *2016 IEEE First International Conference on Data Stream Mining & Processing (DSMP)*, pages 100–103. IEEE, 2016.

Adam Tauman Kalai, Ofir Nachum, Santosh S Vempala, and Edwin Zhang. Why language models hallucinate. *arXiv preprint arXiv:2509.04664*, 2025.

Mateusz Barański, Jan Jasiński, Julitta Bartolewska, Stanisław Kacprzak, Marcin Witkowski, and Konrad Kowalczyk. Investigation of whisper asr hallucinations induced by non-speech audio. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2025.

Allison Koenecke, Anna Seo Gyeong Choi, Katelyn X Mei, Hilke Schellmann, and Mona Sloane. Careless whisper: Speech-to-text hallucination harms. In *Proceedings of the 2024 ACM conference on fairness, accountability, and transparency*, pages 1672–1681, 2024.

Qualcomm AI Hub. Whisper-base. `https://aihub.qualcomm.com/iot/models/whisper_base`.

Daliang Xu, Hao Zhang, Liming Yang, Ruiqi Liu, Gang Huang, Mengwei Xu, and Xuanzhe Liu. Fast on-device llm inference with npus. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pages 445–462, 2025.

Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*, 26(1):43–49, 1978.

Sara Papi, Marco Gaido, Matteo Negri, and Luisa Bentivogli. Streamatt: Direct streaming speech-to-text translation with attention-based audio history selection. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3692–3707, 2024.

Jonas Waldendorf, Bashar Awwad Shiekh Hasan, and Evgenii Tsymbalov. Detecting hallucinations in speechllms at inference time using attention maps. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 43276–43289, 2026.

Linhao Dong and Bo Xu. Cif: Continuous integrate-and-fire for end-to-end speech recognition. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6079–6083. IEEE, 2020.

François Hernandez, Vincent Nguyen, Sahar Ghannay, Natalia Tomashenko, and Yannick Esteve. Ted-lium 3: Twice as much data and corpus repartition for experiments on speaker adaptation. In *International conference on speech and computer*, pages 198–208. Springer, 2018.

Qian Zhang, Han Lu, Hasim Sak, Anshuman Tripathi, Erik McDermott, Stephen Koo, and Shankar Kumar. Transformer transducer: A streamable speech recognition model with transformer encoders and rnn-t loss. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7829–7833. IEEE, 2020.

Manjunath Kudlur, Evan King, James Wang, and Pete Warden. Moonshine v2: Ergodic streaming encoder asr for latency-critical speech applications. *arXiv preprint arXiv:2602.12241*, 2026.

Monsoon Solutions, Inc. High Voltage Power Monitor. `https://www.msoon.com/high-voltage-power-monitor`.

Qualcomm Technologies, Inc. Qualcomm Profiler. `https://www.qualcomm.com/developer/software/qualcomm-profiler`.

Tomer Krichli, Bhiksha Raj, and Joseph Keshet. Carelesswhisper: Turning whisper into a causal streaming model. *arXiv preprint arXiv:2508.12301*, 2025.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Monica Sekoyan, Nithin Rao Koluguri, Nune Tadevosyan, Piotr Zelasko, Travis Bartley, Nikolay Karpov, Jagadeesh Balam, and Boris Ginsburg. Canary-1b-v2 & parakeet-tdt-0.6 b-v3: Efficient and high-performance models for multilingual asr and ast. *arXiv preprint arXiv:2509.14128*, 2025.

Loïc Barrault, Yu-An Chung, Mariano Cora Meglioli, David Dale, Ning Dong, Paul-Ambroise Duquenne, Hady Elsahar, Hongyu Gong, Kevin Heffernan, John Hoffman, et al. Seamlessm4t: Massively multilingual & multimodal machine translation. *arXiv preprint arXiv:2308.11596*, 2023.

Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. 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, 2015.

Stefanos Laskaridis, Kleomenis Katevas, Lorenzo Minto, and Hamed Haddadi. Melting point: Mobile evaluation of language transformers. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, pages 890–907, 2024.

GGML. whisper.cpp. `https://github.com/ggml-org/whisper.cpp`.

Keith Ito and Linda Johnson. The lj speech dataset. `https://keithito.com/LJ-Speech-Dataset/`, 2017.

A Hallucination Detection Analysis

A.1 Filtered Attention-Difference Signal

Filtered attention. For each generated token i, we compare its final-layer cross-attention with that of the most recent preceding content token $p(i)$. The raw attention-difference signal is defined as

$$\delta_i(f) = \tilde{a}_i(f) - \tilde{a}_{p(i)}(f), \tag{6}$$

where f denotes the audio frame index. To reduce spurious spikes near the beginning and end of the audio as shown in Figure 5, we apply a median filter followed by a moving-average filter:

$$\hat{\delta}_i(f) = \mathrm{MA}_{w_a}\left(\mathrm{Med}_{w_m}(\delta_i)\right)(f), \tag{7}$$

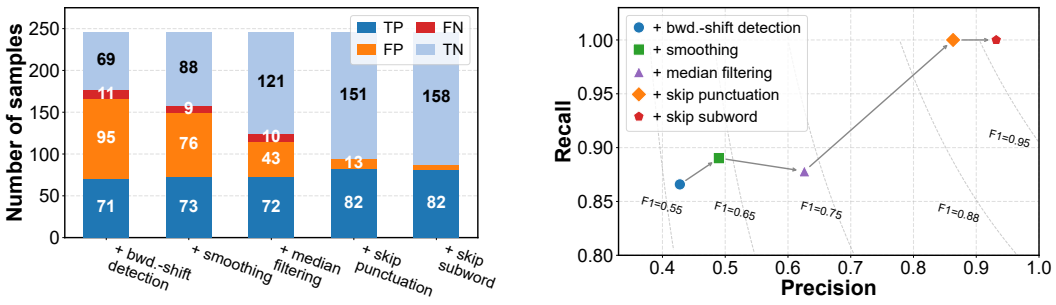
where Med_{w_m} denotes a median filter with window size w_m, and MA_{w_a} denotes a moving-average filter with window size w_a. We use the filtered signal $\hat{\delta}_i(f)$ to locate the positive and negative peaks used for hallucination detection.

A.2 Component Ablation Study

Effect of token-validation components. Table 2 analyzes the effect of each component in our token validation method using 250 samples from the LibriSpeech dataset [Panayotov et al., 2015]. Without padding, Whisper produces a high hallucination rate. Backward-shift detection substantially reduces hallucinations, while smoothing and median filtering reduce undesired backward-shift detections. Skipping punctuation and subword tokens further improves robustness by avoiding unreliable temporal anchors. The full configuration eliminates hallucinations in this study and achieves the lowest undesired backward-shift detection rate. In *NPUsper*, even if a rare undesired backward shift stops decoding early, the remaining audio is retained as carryover and processed together with newly arrived audio in the next inference round.

Table 2: Ablation study of token validation components. Hall. denotes hallucination rate. Bwd. Shift denotes backward attention shift detection rate. Undesired Bwd. denotes undesired backward-shift detections.

Method variant	Padding	Hall. (%)	Bwd. Shift (%)	Undesired Bwd. (%)
30s padded baseline	30s	0.0	–	–
No-padding baseline	None	34.4	–	–
+ backward-shift detection	None	4.4	70.8	75.7
+ smoothing	None	2.8	65.2	82.2
+ median filtering	None	1.2	51.6	67.4
+ skip punctuation tokens	None	0.0	39.6	19.2
+ skip punctuation and subword tokens	None	0.0	36.4	9.9



(a) Component-level outcome breakdown

(b) Precision and recall

Figure 10: Additional analysis of hallucination detection. The stacked bar plot summarizes the outcome distribution for each token-validation variant, while the precision–recall plot characterizes detection quality under different filtering and token-skipping choices.

Table 3: Filter parameter selection. Hall., Bwd. Shift, and Undesired Bwd. are reported in percentage.

Config. (MA, Med.)	Hall.	Bwd. Shift	Undesired Bwd.
(0, 0)	4.4	70.8	75.7
(5, 0)	3.6	62.0	83.2
(10, 0)	2.8	65.2	82.2
(15, 0)	3.2	67.6	84.0
(10, 3)	2.4	60.0	74.7
(10, 5)	1.2	59.2	73.0
(10, 7)	1.2	51.6	67.4

Ablation visualization. Figure 10 provides a complementary view of the same ablation. The stacked bars (Figure 10(a)) show that each component progressively reduces false or undesired detections while preserving the ability to catch hallucinated tokens. The precision–recall plot (Figure 10(b)) further shows that token filtering improves detection precision while maintaining high recall, indicating that punctuation and subword tokens are unreliable temporal anchors for hallucination detection.

A.3 Filter Parameter Selection

Parameter selection rationale. We choose the moving-average and median-filter window sizes in two steps. Table 3 reports representative configurations used for this selection. First, without median filtering, we vary the moving-average window and find that $w_a = 10$ reduces the hallucination rate among the tested smoothing settings. Second, with $w_a = 10$ fixed, we add a median filter to remove spurious attention peaks in the attention-difference signal, which are visible in Figure 5. A median-filter width of $w_m = 7$ removes most of these peaks and reduces undesired backward-shift detections while maintaining a low hallucination rate. We therefore use $(w_a, w_m) = (10, 7)$ in our implementation. Compared with this parameter selection, the larger improvement comes from the backward-shift criterion and proper handling of punctuation and subword tokens, as shown in Table 2 and Figure 10.

A.4 Model Selection and Generalizability

Model selection. Our end-to-end system evaluation focuses on the Whisper base model. This choice reflects the memory and compute constraints of mobile devices, where the goal is to build a complete real-time on-device transcription system rather than to maximize transcription accuracy with a larger model. We exclude Whisper tiny because its transcription quality is substantially weaker and it is rarely used as the main model in recent Whisper-based streaming systems [Wang et al., 2025]. We also exclude Whisper large from this analysis in this section because it caused an out-of-memory error even on the Galaxy S25, which is a high-end mobile device.

Cross-model evaluation. Although our end-to-end system uses Whisper base, the proposed hallucination detection method (§ 4.3) is not specific to the base model. Its generality comes from relying on temporal alignment in decoder cross-attention, which is consistently observed across different Whisper model sizes. To evaluate this property, we reuse the configuration chosen for Whisper base on Whisper small and medium, including the same filtering and token-skipping parameters. For each model size, we compare a no-detection baseline against our method.

Generalizability. Table 4 shows that hallucination detection remains effective beyond Whisper base. On Whisper small, our method reduces the hallucination rate from 24.0% to 2.0% for both selected layers. Undesired backward-shift remains modest at 5.2% and 3.2%, respectively. On Whisper medium, our method reduces the hallucination rate from 36.4% to 6.8%, with 6.9% undesired backward-shift. These results indicate that temporal alignment in decoder cross-attention provides a general signal for detecting hallucinated tokens across different Whisper model sizes.

Deployment feasibility. Although our hallucination detection method remains effective, Whisper medium is less practical for real-time deployment because its encoder alone takes more than 4 s on the Snapdragon X Plus laptop, which exceeds the short inference interval in live transcription systems. This makes Whisper medium difficult to use as a full real-time on-device system, despite its potential for higher transcription accuracy. We leave the construction of fully optimized end-to-end

Table 4: Generalizability of hallucination detection across Whisper model sizes. Hall. denotes hallucination rate. Bwd. denotes backward attention shift detection rate. Undesired denotes undesired backward-shift detections. All "Our method" rows use the same configuration selected for Whisper base.

Model	Config.	Layer	Hall. (%)	Bwd. (%)	Undesired (%)
Small	No detection	–	24.0	–	–
	Our method	5	2.0	5.2	7.7
	Our method	7	2.0	3.2	12.5
Medium	No detection	–	36.4	–	–
	Our method	8	6.8	28.8	6.9

systems for larger Whisper model sizes to future work, including model-size-specific hyperparameter tuning for hallucination detection.

A.5 Overhead Analysis

Runtime overhead. We measure the runtime overhead of the hallucination detection pipeline on the Galaxy S25. The total analysis overhead is less than 1 ms per inference round, including cross-attention tensor transfer and analysis, median and moving-average filtering, punctuation and subword identification, and DTW [Sakoe and Chiba, 1978].

Memory overhead. Storing the cross-attention scores for the K generated tokens requires 86.8 KB of temporary memory per inference round on average, with a peak of 116.9 KB. This temporary memory is released after each inference round and does not accumulate across rounds.

A.6 Token Filtering and Multilingual Analysis

Token identification. We exclude punctuation tokens and subword continuations from temporal attention comparison because they provide unreliable temporal anchors. As illustrated in Figure 5, punctuation tokens exhibit diffuse cross-attention, while subword continuations often attend to overlapping frames within the same word. Our implementation identifies punctuation tokens using a byte-level heuristic. After removing at most one leading ASCII space, a token is classified as punctuation if the remaining string is one or two bytes long and none of its bytes is classified as alphanumeric by `isalnum`. Empty strings and single-space tokens are also excluded. For English, Spanish, French, German, and Italian, a token is treated as a subword continuation if its decoded text does not begin with an ASCII space and contains at least one byte classified as alphabetic by `isalpha`. These checks operate on individual bytes rather than Unicode characters. Excluded tokens are skipped when computing temporal attention differences and do not update the reference attention used for subsequent comparisons. As shown in Table 2, excluding punctuation and subword tokens substantially reduces undesired backward-shift detections.

Multilingual analysis. We further evaluate the same token-filtering strategy on Spanish using the same hallucination-detection configuration as in the English evaluation. As shown in Table 5, filtering punctuation tokens reduces the undesired backward-shift detection rate from 81.1% to 57.7%, and additionally filtering subword continuations further reduces it to 33.8%. We observe similar trends for French, German, and Italian under the same system configuration, although optimal performance may require language-specific parameter tuning.

B Additional Results

B.1 Transcription Quality

Word and character errors. Table 6 reports additional transcription quality results. All systems are evaluated with the Whisper base model, and WER/CER are computed per sample and reported as mean$\pm$standard deviation. Full WER/CER compare each system output against the complete reference transcript. For Whisper-Streaming and WhisperFlow, some runs terminate before processing the full audio, which can inflate full-reference WER/CER due to incomplete coverage. We therefore

Table 5: Token-filtering analysis on Spanish. Hall. denotes hallucination rate, Bwd. Shift denotes backward attention shift detection rate, and Undesired Bwd. denotes undesired backward-shift detections.

Method variant	Hall. (%)	Bwd. Shift (%)	Undesired Bwd. (%)
30s padded baseline	0.4	–	–
No-padding baseline	28.4	–	–
+ backward-shift detection	2.0	77.2	89.6
+ smoothing	2.4	67.6	85.8
+ median filtering	2.4	59.2	81.1
+ skip punctuation tokens	0.0	38.8	57.7
+ skip punctuation and subword tokens	2.4	32.0	33.8

additionally report truncated WER/CER for these cases, comparing each output only against the reached reference prefix.

Table 6: Additional per-sample full and truncated WER/CER across systems. WS, WF, SS, and SW denote Whisper-Streaming, WhisperFlow, SimulStreaming, and Simul-Whisper, respectively. Values are percentages and reported as mean±population standard deviation over samples.

Dataset	**System**	**Full WER**	**Full CER**	**Trunc. WER**	**Trunc. CER**
Meanwhile	WS	30.69±24.23	24.89±25.07	11.79±6.32	5.29±8.62
	WF	25.13±22.95	18.69±23.89	16.64±19.56	9.58±20.72
	SS	12.06±4.00	5.40±1.89	–	–
	SW	12.93±4.49	5.85±2.20	–	–
	Ours	16.36±11.21	8.44±6.29	–	–
TED-LIUM 3	WS	91.15±5.16	90.09±5.73	11.04±3.56	2.81±1.08
	WF	90.25±18.95	89.63±19.70	32.73±41.22	29.04±43.47
	SS	11.26±3.34	3.36±1.11	–	–
	SW	11.40±2.90	3.53±1.12	–	–
	Ours	14.18±2.94	5.46±1.26	–	–

B.2 Power Measurement

Measurement setup. We measured full-system battery-rail power on the Galaxy S25 using a Monsoon High Voltage Power Monitor [Monsoon Solutions, Inc.], configured as an external supply for the phone battery rail [Laskaridis et al., 2024]. The S25 traces used for Table 1 were captured with $V_{\text{out}} = 4.42$ V. The Monsoon capture process starts before each live-transcription run and samples the main rail current and voltage at $f_s = 5000$ Hz. The workload script writes start and end marker files immediately before and after the transcription command; all S25 power statistics are computed over the Monsoon samples whose timestamps fall inside this marker-delimited command window. This measures the complete phone system during the workload, not an isolated GPU or NPU rail.

Power and energy computation. For each command-window sample i, we compute instantaneous power as

$$p_i = I_i V_i, \quad \bar{P}_{\text{raw}} = \frac{1}{M}\sum_{i=1}^{M} p_i, \quad E_{\text{raw}} = \frac{1}{f_s}\sum_{i=1}^{M} p_i, \tag{8}$$

where I_i is in mA, V_i is in V, p_i is in mW, and M is the number of samples in the command window. Thus, S25 energy is obtained by discrete integration over Monsoon samples rather than by manually multiplying a reported average by wall-clock time. We separately measure an idle baseline with the same Monsoon setup and subtract its command-window mean power, $P_{\text{idle}} = 362.2$ mW, from each S25 workload:

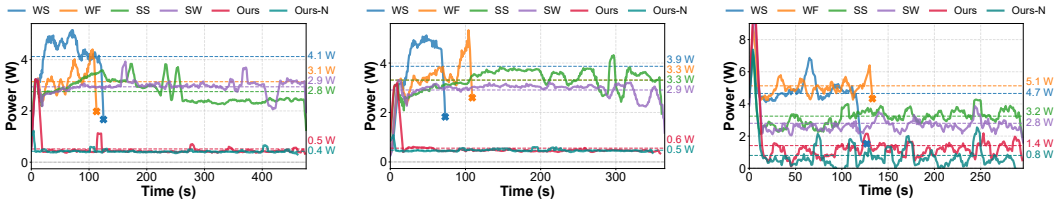
$$P_{\text{work}} = \bar{P}_{\text{raw}} - P_{\text{idle}}, \quad E_{\text{work}} = E_{\text{raw}} - P_{\text{idle}}\frac{M}{f_s}. \tag{9}$$

Reported metrics. Table 1 reports P_{work} and energy metrics derived from E_{work}, converted to W and J. Energy per inference is E_{work}/N, where N is the number of inference iterations in the run logs. Energy per token divides E_{work} by the total number of generated tokens, and throughput is

computed as generated tokens per active inference time, where active time is the sum of mel, encoder, decoder-prefill, and decoder execution time.

Snapdragon X Plus measurement. For the Snapdragon X Plus laptop, we measured power using Qualcomm Profiler [Qualcomm Technologies, Inc.] and applied the same base-power subtraction principle to the profiler traces. Because the S25 and X Plus use different power measurement interfaces, their traces have different noise characteristics. We therefore use the traces in Figure 11 to show qualitative results and report the averaged values in Table 1 for quantitative comparison. The numerical values in Table 1 are computed from the raw command-window summaries.

Power traces. Figure 11 shows the power traces used for Table 1. *NPUsper* consistently maintains much lower power than the baselines because it avoids long padded inputs and repeated hypothesis recomputation. All systems show an initial power peak when transcription starts, reflecting the startup cost of launching the live transcription pipeline. The WS and WF traces on both devices are truncated because these systems terminated before processing the entire audio. This behavior is caused by high per-iteration latency from long Whisper inputs and hypothesis recomputation by Local Agreement.



(a) EricMead on S25 (b) GaryFlake on S25 (c) GaryFlake on X Plus

Figure 11: Power consumption. *NPUsper* achieves the lowest power consumption by reducing redundant computation on both GPU and NPU. The $\times$ markers indicate when each system terminates.

B.3 Controlled Unrolling on NPUs

Cross-device consistency. In addition to the S25 results in Figure 9, we provide supplementary X Plus results in Figure 12. The X Plus results show the same overall trend as S25: the fixed full-length graph has the highest decoding overhead, the separate-graph approach reduces redundant KV-cache computation but still incurs graph-dispatch overhead, and our controlled-unrolling design achieves the lowest overall decoding overhead.

Across decoding lengths. We also evaluate whether this trend remains consistent as the number of generated tokens changes. Figure 13 reports the S25 decoding breakdown for shorter, medium, and longer decoding cases. Across all token-count settings, the relative behavior of the three strategies remains unchanged. The fixed full-length graph remains the least efficient, the separate-graph approach improves over it but retains dispatch overhead, and our method consistently provides the lowest decoding overhead.

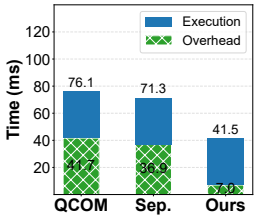
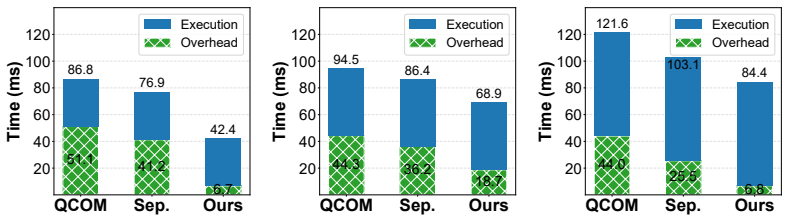


Figure 12: Decoding overhead on X Plus.



(a) 16 tokens (b) 23 tokens (c) 30 tokens

Figure 13: Decoding overhead analysis on S25 as the number of generated tokens varies.

Power and energy. We further compare the power and energy efficiency of the three NPU decoding strategies. As shown in Table 7, controlled unrolling achieves the lowest average power, energy per inference, and energy per token on both devices, while also providing the highest throughput. The

step-specific design shows substantially lower throughput because it requires the largest number of precompiled graphs, which cannot all remain resident in memory and must therefore be loaded on demand. Overall, controlled unrolling improves decoding efficiency without introducing additional power or energy overhead.

Table 7: Power and energy efficiency of NPU decoding strategies. Power and energy values are reported after subtracting idle power.

Device	Strategy	Power (W) ↓	E/inf. (J) ↓	E/token (J) ↓	Tput. (tok/s) ↑
S25	Fixed one graph	0.55	1.10	0.11	96.62
	Step-specific graphs	0.85	1.70	0.17	21.39
	Controlled unrolling	**0.46**	**0.92**	**0.10**	**112.90**
X Plus	Fixed one graph	1.00	2.02	0.21	135.04
	Step-specific graphs	1.98	4.01	0.41	10.68
	Controlled unrolling	**0.81**	**1.65**	**0.19**	**144.98**

Deployment overhead. Controlled unrolling requires multiple precompiled chunk graphs, resulting in moderate graph-storage and runtime-memory overhead. Table 8 compares these overheads across the three decoding strategies. On both devices, controlled unrolling maintains substantially lower storage and runtime-memory footprints than the step-specific design, while providing the decoding-efficiency benefits described above.

Table 8: Graph-storage and runtime-memory overhead of NPU decoding strategies.

Device	Strategy	Storage (MB)	Avg. Mem. (MB)	Peak Mem. (MB)
S25	Fixed one graph	151.7	45.4	45.9
	Step-specific graphs	316.3	61.9	62.7
	Controlled unrolling	220.1	52.1	52.7
X Plus	Fixed one graph	152.1	272.3	273.5
	Step-specific graphs	322.0	552.7	554.0
	Controlled unrolling	223.4	359.4	365.3

B.4 End-to-End Component Ablation

Component ablation. Table 9 reports an end-to-end ablation of the three key design choices in *NPUsper*: removing padding, hallucination detection, and controlled unrolling. Removing padding substantially reduces Mel-spectrogram and encoder latency, but causes severe accuracy degradation without hallucination detection. Adding hallucination detection recovers transcription accuracy while retaining the latency benefit of short inputs. Finally, controlled unrolling further reduces decoding latency by reducing redundant KV-cache computation and graph-dispatch overhead. The same overall trend is observed on both devices.

Table 9: End-to-end component ablation. Latency values (ms) are averaged over inference rounds.

Dev.	Padding	Hall. det.	Controlled unrolling	Input (s)	Mel	Enc.	Prefill	Decode	Total	WER
S25	30s	✗	✗	30.000	86.30	69.31	39.37	75.44	270.42	9.93
	None	✗	✗	2.709	20.72	5.66	18.99	65.54	110.91	99.03
	None	✓	✗ (Fixed)	3.072	25.45	6.05	17.55	43.75	92.81	6.50
	None	✓	✗ (Step-specific)	3.119	20.38	7.21	19.57	39.93	441.60	7.03
	None	✓	✓ (Ours)	3.094	26.45	6.53	8.99	34.62	76.60	6.50
X Plus	30s	✗	✗	30.000	67.79	75.53	24.71	42.13	210.15	10.98
	None	✗	✗	3.145	8.05	8.82	16.67	41.72	75.26	90.77
	None	✓	✗ (Fixed)	3.054	8.25	4.71	15.86	25.68	54.49	7.29
	None	✓	✗ (Step-specific)	3.019	8.34	7.34	22.13	40.22	788.64	6.59
	None	✓	✓ (Ours)	3.056	7.96	5.43	8.83	22.18	44.41	7.56

B.5 Supplementary CPU-only System Comparison

CPU-only setup. Table 10 reports a supplementary CPU-only system comparison on Galaxy S25 and Snapdragon X Plus. All systems use the same 2 s inference interval and run without GPU acceleration. This experiment is not intended as our main comparison setting. Instead, it characterizes how each system behaves when real-time streaming must be sustained using CPU execution alone.

Table 10: CPU-only realtime results on GaryFlake and EricMead. All values are reported as averages over the two samples. PWL denotes per-word latency. Input is the encoder input length, Prep. is the mel-spectrogram computation time, Enc. is encoder execution time, and Dec. includes both prefill and autoregressive decoding time.

Dev.	Sys.	Full WER (%)	Trunc. WER (%)	PWL (s)	Input (s)	Prep. (ms)	Enc. (ms)	Dec. (ms)
S25	WS	89.71	11.34	10.09	30.00	23.4	2141.7	852.8
	WF	90.51	56.71	5.40[2]	12.81	24.6	854.3	430.0
	SS	58.02	–	95.38	30.00	168.0	2832.6	1142.2
	SW	100.00	–	–	30.00	49.4	2814.1	191.6
	Ours	13.43	–	2.00	3.67	5.6	161.6	37.1
X Plus	WS	88.19	6.17	4.91	30.00	9.6	690.7	306.0
	WF	85.44	17.63	3.92	13.99	8.3	230.3	154.4
	SS	7.79	–	3.31	30.00	19.4	698.0	336.1
	SW	7.69	–	3.08	30.00	17.4	640.7	99.0
	Ours	9.62	–	1.99	3.76	3.2	52.3	18.3

Mobile CPU bottlenecks. The CPU-only results highlight why removing unnecessary computation is important for on-device streaming inference. On the S25 CPU, systems that repeatedly use a fixed 30s encoder input incur substantial encoder overhead: WS, SS, and SW spend 2141.7 ms, 2832.6 ms, and 2814.1 ms, respectively, on encoder execution alone. These costs already exceed the 2 s inference interval before accounting for preprocessing and decoding. SS further spends 1142.2 ms in the decoder, which is the largest decoder cost among the S25 runs, mainly because it performs the most prefill-token computation. As a result, SS completes but falls far behind the real-time latency performance, while SW fails to produce a usable transcript.

Effect of removing redundant computation. *NPUsper* avoids much of this overhead by using online hallucination detection to remove unnecessary reprocessing for mobile execution. Its average encoder input length is only 3.67s on S25, reducing encoder execution to 161.6 ms and decoder execution to 37.1 ms. This allows *NPUsper* to complete each CPU-only streaming update within the inference interval even on the mobile CPU. The X Plus results show the same trend under a stronger CPU: all systems become less constrained, but *NPUsper* still uses substantially shorter encoder inputs and lower encoder/decoder latency than the baselines.

Scope. Note that these results are not intended as the main system comparison, but as a supplementary evaluation showing that on-device AI systems are highly sensitive to redundant computation. Eliminating those redundant computations for encoder and decoder is essential for sustaining real-time streaming on mobile processors. Jointly pipelining the CPU, GPU, and NPU remains an interesting direction for future work.

C GPU Implementation Details

Implementation and GPU setup. All GPU implementations were built on `whisper.cpp` [GGML], with GGML used as the tensor computation backend. For GPU-accelerated execution, we used the GGML OpenCL backend on Qualcomm Adreno GPUs. Experiments were conducted on two devices: a Samsung Galaxy S25 [Samsung] equipped with a Qualcomm Adreno 830 GPU, and an ASUS Vivobook S 15 Qualcomm powered by Snapdragon X Plus with a Qualcomm Adreno X1-45 GPU. Both platforms used OpenCL 3.0 and were compiled in release mode with Adreno-optimized OpenCL kernels enabled. Unless otherwise stated, all experiments used the `ggml-base.bin` Whisper

[2]WF per-word latency is computed from GaryFlake only because the EricMead run did not progress long enough for word-latency matching.

model, and streaming inference was performed with a 2 s inference round interval. The motivation experiments in § 3 were conducted on the Snapdragon X Plus using 200 samples from the LibriSpeech dataset [Panayotov et al., 2015].

D NPU Implementation Details

D.1 Graph Construction

ONNX/QNN setup. For NPU execution, we use ONNX graphs executed with the ONNX Runtime QNN execution provider and the QAIRT/QNN HTP backend. The NPU implementation follows the QAI Hub Whisper graph construction: transformer linear layers are represented as 1×1 convolution operators for QNN execution, while core Whisper operations such as attention, layer normalization, GELU, and encoder front-end convolutions are preserved.

Table 11: Effect of QAI Hub linear-to-1×1-convolution lowering on S25 NPU latency. Both variants use the same ONNX Runtime QNN/QAIRT stack and ONNX I/O contract.

Metric	QAI Hub-lowered	HF-direct Linear	Reduction
Encoder `Session::Run`	47.618 ms	51.195 ms	7.0%
Decoder `Session::Run` sum	139.417 ms	158.354 ms	12.0%
Decoder total	184.570 ms	205.664 ms	10.3%
Preprocess + encoder + decoder	235.060 ms	259.955 ms	9.6%

Lowering effect. This lowering improves S25 NPU latency under the same runtime stack, so we use the lowered QNN graphs for our NPU implementation.

D.2 Input and Model Bucketing

Bucket selection. The NPU runtime uses input and model bucketing to connect variable-length streaming audio with the static-shape execution model of Qualcomm QNN. Rather than compiling a separate graph for every possible window length or padding every input to Whisper's 30s context, *NPUsper* selects the smallest supported bucket that can contain the current effective audio window. Given an effective audio duration d, the selected bucket is

$$\mathcal{B} = \{3, 4, 5, 6, 30\}\ \text{s}, \qquad b(d) = \min\{\, \beta \in \mathcal{B} \mid d \le \beta \,\}.$$

The 3–6s buckets are sufficient for most audio-buffer lengths in our experiments, while the 30s graph is kept as a rare fallback for unstable segments that require reinference with the native Whisper context.

D.3 Controlled Unrolling Decoder Graphs

Chunked decoder execution. The decoder is executed as a sequence of small static graphs rather than as one graph that always runs the full decoding loop. This keeps each NPU graph shape-specialized while allowing the runtime to stop early when no further tokens are needed. For each bucket, the decoder supports up to 30 output tokens. After each chunk graph, the system emits valid tokens and dispatches the next graph only if decoding must continue.

Chunk schedule and prompt prefill. For each bucket, the decoder is implemented as a controlled-unrolling chain with a maximum decode length of 30 tokens. The default schedule is [4,6,5,5,5,5], where the first chunk handles Whisper's four special tokens. When carryover text is used as a decoder prompt, we use the prefill schedule [6,4,5,5,5,5] so that prompt and special tokens are handled within the first two chunks. Since both schedules cover 10 positions before the later chunks, the remaining [5,5,5,5] graphs are shared across the default and prefill paths. Each chunk returns logits, cross-attention maps, and updated self-KV cache tensors, and chunk wrappers within each bucket are linked into a weight-shared QNN context binary to minimize decoder weights.

Chunk-size selection. We select the repeated later chunk size while keeping the first two chunks fixed as [4,6] for the default path and [6,4] for the prompt-prefill path. We compare later chunk sizes of 3, 5, and 7, corresponding to default schedules of [4,6,3,3,3,3,3,3], [4,6,5,5,5,5], and [4,6,7,7,7], respectively. This comparison captures the trade-off between graph-dispatch overhead and unused

decoder computation: smaller chunks require more graph launches, while larger chunks execute more decoder positions than may be needed.

Selected chunk size. Table 12 shows that 3-token chunks incur excessive decoder calls and graph switches, while 7-token chunks perform more unnecessary decoder-position computation. The 5-token setting best balances graph-dispatch overhead and unused decoder work, achieving the lowest per-word latency. We therefore use [4,6,5,5,5,5] for the default path and [6,4,5,5,5,5] for the prompt-prefill path.

Table 12: Chunk-size selection with fixed first two chunks. Per-word latency is measured as the difference between a word emission time and the end time of its matched ground-truth word. Extra work/token is a proxy for unused decoder computation, measured as the number of decoder token positions executed per emitted transcript token.

Chunk size	Per-word latency (s)	Calls/token	Switches/token	Extra work/token
3	1.994	0.452	0.575	1.827
5	1.943	0.389	0.510	1.946
7	2.015	0.356	0.475	2.040

D.4 Cross-Attention Masking

Padded source positions. In the bucketed NPU runtime, the number of encoder frames attended by the decoder is fixed by the selected bucket rather than by the exact audio duration. When the streaming window is shorter than the bucket, the encoder output contains real audio positions followed by padded positions introduced only to match the static bucket shape. If left unmasked, these padded positions would still participate in the cross-attention softmax and could change the attention distribution over real audio frames.

Masking padded positions. Each decoder chunk receives a cross-attention mask constructed from the non-padded audio length recorded during bucket selection. The mask leaves the columns corresponding to real audio unchanged and adds a finite negative bias of -100 to the padded columns before the softmax. We use a finite value instead of infinity to avoid numerical issues under reduced-precision NPU execution. The final-layer cross-attention maps returned to the host are also cropped to the real-audio region before hallucination detection and DTW-based carryover updates. As a result, the compiled NPU graph keeps a fixed bucket shape, while token validation and streaming decisions are computed only from real audio.

Mask-removal case study. Figure 14 shows the token-level effect of removing the cross-attention mask on the LJ-Speech utterance LJ001-0021 [Ito and Johnson, 2017]. With the same encoder input, decoder weights, and decoding state, the unmasked decoder leaves the masked top-1 trajectory after step 10. At the first failure point, the masked token is absent from the unmasked top-3 candidates; in the following steps, it is demoted below an incorrect unmasked top-1 token.

D.5 Self-Attention Cache Handling

Static cache interface. Controlled unrolling executes multiple autoregressive decoding steps inside one NPU graph, but the first chunk would normally start with an empty self-KV cache. This zero-length cache is inconvenient for QNN compilation and would make the first chunk graph a special case. We avoid this by reserving one leading dummy KV slot, so every chunk starts with a non-empty cache and follows the same read–append–return cache interface.

Dummy-slot masking. The dummy slot is only a structural placeholder. It is masked as an invalid self-attention source position, and the host never treats it as a generated token. Thus, the cache interface remains static for NPU execution while token emission, EOS checks, hallucination detection, and prompt carryover operate only on real generated tokens.

D.6 Software Stack

NPU runtime stack. We implemented the bucketed NPU runtime using ONNX Runtime with Qualcomm's QNN HTP backend. Encoder and decoder graphs were exported to ONNX and compiled

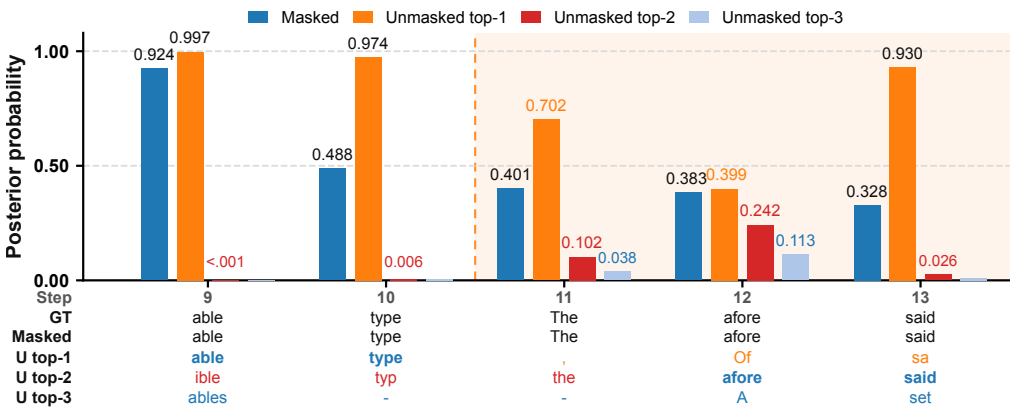


Figure 14: Effect of removing the cross-attention source-axis mask on LJ001-0021, decoder steps 9–13. Bars show the masked top-1 posterior and the unmasked top-1, top-2, and top-3 posteriors. At step 11, the masked token is absent from the unmasked top-3 candidates; at steps 12–13, it appears only as the second-ranked unmasked candidate.

offline with Qualcomm AI Runtime SDK (QAIRT) `2.37.1.250807`. The same runtime, NPU backend, and offline compiler stack were used across both target devices. The target-specific details are the application build environments: Visual Studio 2022 ARM64/MSVC for the Snapdragon X Plus laptop and Android NDK r26d with clang 17 for the Galaxy S25.

Table 13: Shared NPU software stack.

Item	**Configuration**
Runtime	ONNX Runtime QNN `1.23.2`
NPU backend	QAIRT/QNN HTP `2.37.1.250807`
Offline compiler	`qairt-converter`, `qnn-context-binary-generator`
Targets	Snapdragon X Plus laptop; Samsung Galaxy S25
Builds	VS2022 ARM64/MSVC; Android NDK r26d, clang 17, `android-33`

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the main contributions and scope of the paper.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses the main limitations of the proposed system.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: The paper does not present theoretical results or formal proofs.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper describes the system design, implementation, datasets, baselines, and evaluation metrics.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The paper provides code and uses publicly available datasets and models.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specifies the datasets, model, hardware platforms, inference backends, baselines, and metrics used in the experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The paper reports standard deviation error bars for the main experimental results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper reports the hardware platforms, accelerators, model, and inference backends used for the experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work uses public datasets and models and does not involve private user data or human-subject experiments.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The paper discusses privacy benefits and potential risks from transcription errors.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release a new high-risk model or dataset requiring safeguards.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper cites the existing assets used in the evaluation, and the released artifact documents their license information.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The paper releases a system implementation and scripts for reproducing the experiments.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core methods do not involve LLMs as an important, original, or non-standard component.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.