

sEMG Typing Transcription with Conformer Encoding and Beam-Aware Large Language Model Correction

Yuanxi Wang¹, Lun Shu², and Yasuharu Koike²

¹School of Engineering, Department of Information and Communications Engineering,
Institute of Science Tokyo, Yokohama, Japan

²Institute of Integrated Research, Institute of Science Tokyo, Yokohama, Japan

wang.y.9e10@m.isct.ac.jp shulun.titech@gmail.com koike.y@first.iir.isct.ac.jp

Abstract

Surface electromyography (sEMG) offers a promising path toward private and wearable text input, but accurate typing transcription remains challenging due to strong cross-user and cross-session variability. We propose a beam-aware sEMG typing framework that connects an improved Connectionist Temporal Classification (CTC)-based frontend with a large language model (LLM) corrector through beam-ordered candidate lists. As a standalone sEMG decoder, our CNN-Conformer frontend achieves the best reported performance among compared published emg2qwerty systems under the standard full-session frontend protocol, reaching 19.94% generic character error rate (CER) and 3.43% personalized CER. For LLM correction, we use a separate windowed hypothesis-export protocol, where the frontend’s top-10 CTC hypotheses are provided to a fine-tuned LLM together with a rank-1 baseline. Under this within-protocol comparison, the LLM reduces session-level CER from 22.56% to 19.65% in the generic setting and from 3.42% to 2.01% after personalization. The top-10 oracle results further show that the beam contains recoverable information beyond the rank-1 output. These results demonstrate that a stronger sEMG frontend and beam-aware LLM correction can be combined effectively for sEMG-to-text transcription.

1 Introduction

Decoding typed text from wrist surface electromyography (sEMG) is a sensor-to-text transcription problem: the input is continuous neuromotor activity, and the output is the character sequence typed by the user. In the emg2qwerty dataset, users type prompted text on a QWERTY keyboard while wearing two 16-channel sEMG wristbands, and the task is to recover the typed character sequence from dual-wrist sEMG (Sivakumar et al., 2024). This setting is promising for private and wearable

text input in augmented reality and virtual reality (AR/VR), but remains difficult because sEMG varies strongly across users and sessions, and the decoder must handle continuous character-level transcription with keyboard-specific events such as backspace.

Recent work on emg2qwerty has mainly focused on improving the first-pass sEMG decoder. The original baseline uses automatic speech recognition (ASR) style Connectionist Temporal Classification (CTC) training with a backspace-aware beam decoder and a character-level language model (Sivakumar et al., 2024). Later systems improve cross-user transcription through raw-sEMG Transformer scaling (Mehlman et al., 2025) and bilateral encoder designs (Hadidi et al., 2025). These methods show that stronger frontends are essential for robust sEMG typing. However, generic cross-user transcription remains much harder than personalized transcription, and rank-1 decoder outputs still contain character substitutions, word-boundary errors, and local spelling mistakes.

This motivates language-level correction after first-pass decoding. Large language models (LLMs) provide strong lexical and contextual priors, but unlike ASR post-corrections, sEMG typing requires exact recovery of the typed sequence, including punctuation, spacing, spelling choices, and possible non-standard strings. If the LLM only receives a noisy rank-1 transcript, especially in the generic setting, it may produce fluent text that is not supported by the signal decoder. Correction therefore needs access to candidate alternatives produced from the same sEMG input.

We use the CTC beam as this candidate interface between the sEMG frontend and the LLM. Instead of correcting only the rank-1 output, the LLM receives beam-ordered candidate lists generated from the same local windows together with a separate rank-1 baseline. Candidate alternatives can contain useful characters or word fragments that are miss-

ing from the best beam output. The LLM can then use these candidate alternatives together with the rank-1 baseline to correct the frontend output.

Our system has two stages. First, a compact raw-sEMG convolutional neural network (CNN)–Conformer frontend is trained with CTC to produce frame-level character probabilities. It uses split-and-share bilateral encoding and keyboard-state auxiliary supervision to improve first-pass transcription and produce beam-ordered CTC candidate lists. Second, a beam-aware LLM corrector receives the top-10 beam candidates from non-overlapping three consecutive windows together with a separate rank-1 baseline and is fine-tuned to output the corrected chunk transcript. This chunk-level design provides local context while keeping the prompt length practical for online text input.

Empirically, our CNN–Conformer frontend provides a stronger first-pass sEMG decoder, while the beam-aware LLM corrector further improves local transcription by using alternative CTC hypotheses rather than relying only on the rank-1 output. Ablations show that multiple beam candidates, adjacent-window context, and task-specific fine-tuning are all important for reliable correction. These results suggest that exposing CTC beam hypotheses is an effective way to connect sEMG decoding with language-level correction.

Our contributions are:

- We propose a beam-aware sEMG typing framework that connects a CTC-based signal frontend with an LLM corrector through beam-ordered CTC candidate lists generated from the same sEMG input.
- We propose a CNN–Conformer sEMG encoder as hypothesis generator, achieving state-of-the-art first-pass transcription performance on emg2qwerty while producing beam-ordered candidate lists for correction.
- We show that hypothesis-based LLM correction further improves transcription accuracy by reducing character and word level errors, and analyze how beam size, local context, and fine-tuning affect correction performance.

2 Related Work

sEMG typing connects neuromotor signal modeling with text transcription. We review related work on sEMG recognition benchmarks, recent

emg2qwerty transcription systems, and language-model correction methods that use decoder uncertainty.

sEMG Benchmarks and Sensor-to-Text Transcription

Public sEMG benchmarks have traditionally focused on gesture classification, pose estimation, or motor-control-oriented decoding. Datasets such as NinaPro (Atzori et al., 2014), CSL-HDEMG (Ammar et al., 2015), CapgMyo (Du et al., 2017), putEMG (Kaczmarek et al., 2019), and emg2pose (Salter et al., 2024) typically map an electromyography (EMG) segment to a fixed gesture label, action class, or hand-pose estimate. Recent multi-user benchmarks further emphasize discrete recognition, out-of-distribution generalization, and cross-user or cross-session adaptation (Eddy et al., 2024; Yang et al., 2024). Wrist-sEMG typing poses a different sequence transduction problem: continuous signals must be converted into ordered character sequences while preserving local timing and keyboard-specific editing events such as backspace. This makes the task closer to ASR-style transcription than to conventional gesture classification, and motivates treating sEMG typing as a sensor-to-text transcription problem (Sivakumar et al., 2024). A related line of work studies noisy text entry in augmentative and alternative communication (AAC), where neural noisy-channel language models help recover intended text from noisy brain-computer interface typing signals (Dong et al., 2019).

sEMG Typing Transcription on the emg2qwerty Dataset

The emg2qwerty benchmark established the first large-scale public setting for wrist-sEMG typing, with 1,135 sessions from 108 users and baselines for both generic and personalized decoding (Sivakumar et al., 2024). Its baseline uses a Time Depth Separable ConvNet (TDS) model with CTC training, a 6-gram character language model (LM), and a backspace-aware beam-search decoder. However, the generic model remains above 50% character error rate (CER) on unseen users, while finetuned personalized models are much stronger, highlighting severe cross-user shift. SplashNet improves cross-user transcription through Rolling Time Normalization (RTN), aggressive channel masking, and a split-and-share encoder that reflects the bilateral structure of typing (Hadidi et al., 2025). Beyond typing-specific architectures, Mehlman et al. (2025) show that raw-sEMG Transformer models can scale effectively and be distilled into

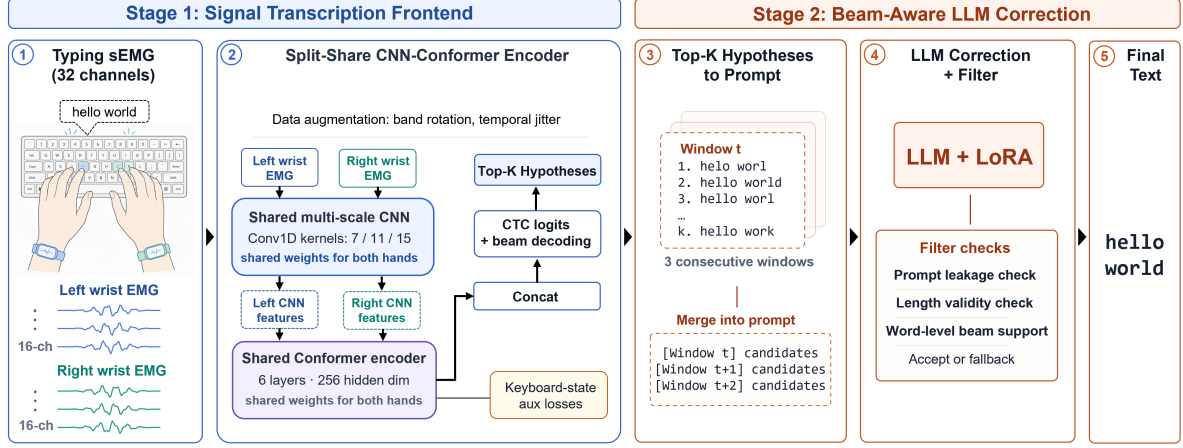


Figure 1: Overview of the proposed beam-aware sEMG transcription pipeline. The CNN–Conformer frontend maps raw sEMG to CTC logits and decodes each window into a beam-ordered CTC candidate set. The CTC hypothesis interface exports top-20 beam hypotheses from consecutive windows and passes the top-10 candidates from each window to the LLM corrector. The LLM adapted with low-rank adaptation (LoRA) then performs local correction using beam-ordered candidate lists together with the concatenated rank-1 baseline.

much smaller models with little loss, suggesting that stronger raw-signal encoders are a promising direction for robust sEMG modeling.

MyoText introduces a hierarchical sEMG-to-text pipeline that predicts finger activations, applies ergonomic QWERTY typing priors to infer possible letters, constructs candidate words, and uses a fine-tuned T5 decoder for sentence reconstruction (Chowdhury et al., 2026). By reducing the decoding search space, MyoText reports strong results on a 30-user subset of emg2qwerty, with 85.4% finger-classification accuracy, 5.4% CER, and 6.5% word error rate (WER). This demonstrates the benefit of combining signal-related structure with language-level decoding. Our work studies a complementary setting: we keep the continuous, open-vocabulary, character-level CTC formulation used in prior emg2qwerty systems. Because MyoText uses a different subset and intermediate supervision, its numbers are not directly comparable to the official full-session emg2qwerty benchmark.

Hypothesis-Aware Language Model Correction

Language models have long been integrated into ASR through beam decoding and rescoring, while recent post-correction methods use neural sequence editors, pretrained encoder–decoder models, or LLMs to repair recognition errors (Leng et al., 2021; Ma et al., 2023; Yang et al., 2023; Hu et al., 2024; Ma et al., 2025). Compared with 1-best correction, N-best correction exposes multiple de-

coder hypotheses to the correction model, providing richer alternatives from the ASR search space and often improving robustness (Ma et al., 2023; Hu et al., 2024; Ma et al., 2025). At the same time, generative correction can introduce unsupported rewrites: an LLM may produce fluent text that is not supported by the underlying signal, especially when the decoded transcript is noisy or out of domain (Hu et al., 2024; Ma et al., 2025). Our work adapts this hypothesis-aware correction paradigm to sEMG typing. Instead of correcting only the rank-1 CTC output, the LLM receives top-10 CTC hypotheses from three adjacent sEMG windows. These signal-derived candidates serve as an explicit interface between the sEMG decoder and the LLM, allowing the corrector to select, combine, and minimally revise alternatives produced by the frontend.

3 Method

3.1 Task Definition and System Overview

We study character-level transcription from dual-wrist sEMG using the public emg2qwerty dataset (Sivakumar et al., 2024). The dataset contains 1,135 typing sessions from 108 users, with synchronized left- and right-wrist sEMG signals and timestamped keystroke annotations. In each session, users type prompted English text on a QWERTY keyboard; the prompts consist of sampled words and Wikipedia sentences, and the annotations record the actual typed keystrokes, including

corrections such as backspace. Following the official split, 100 users are used for training and validation, and 8 held-out users are used for evaluation.

For each window, the input is a raw sEMG segment $X \in \mathbb{R}^{T \times 2 \times 16}$, where the two bands correspond to the left and right wrists and each band contains 16 electrode channels. The target is a character sequence $Y = (y_1, \dots, y_U)$ over the keyboard character set. The goal is to recover the typed text from continuous raw sEMG signals.

Our system follows a two-stage pipeline, as shown in Figure 1. The first stage is a CNN–Conformer CTC frontend that maps raw sEMG windows to frame-level character posteriors and decodes them with beam search. The second stage is a beam-aware LLM corrector that receives local CTC hypothesis sets and produces a corrected transcript. The connection between the two stages is described in Section 3.3.

3.2 Signal Transcription Frontend

3.2.1 Raw EMG Input and Preprocessing

The frontend operates on raw dual-wrist sEMG sampled at 2000 Hz. Each training example uses a 4-second core window with 900 ms past context and 100 ms future context, following the emg2qwerty baseline. We apply the same band rotation and temporal alignment jitter augmentations to improve robustness to armband placement shifts and small left-right timing offsets.

3.2.2 Multi-Scale Raw EMG Frontend

We use a multi-scale convolutional frontend to encode local EMG patterns from the time-domain signal. The first layer contains three parallel 1D convolution branches with kernel sizes 7, 11, and 15, followed by instance normalization and a rectified linear unit (ReLU). Their outputs are concatenated and fused with a 1×1 convolution. Two additional convolution layers downsample the signal with a total stride of 20, producing a roughly 100 Hz feature sequence for the Conformer encoder.

3.2.3 Split-and-Share Conformer Encoder

We use a split-and-share encoder inspired by SplashNet (Hadidi et al., 2025). The left and right wrist streams are processed separately with shared CNN–Conformer weights, which encourages a common wrist-level representation while preserving hand-specific information before fusion. Compared with SplashNet, we apply split-and-share encoding directly to raw EMG and replace the

TDSCConv (Hannun et al., 2019) encoder with a Conformer encoder (Gulati et al., 2020). The two encoded streams are concatenated and projected to character logits for CTC training. Unless otherwise stated, the Conformer uses 6 layers, hidden size 256, 16 attention heads, feed-forward size 1024, kernel size 15, and dropout 0.2.

3.2.4 CTC Training Objective and Keyboard-State Auxiliary Supervision

We train the main transcription branch with the standard CTC objective (Graves et al., 2006). To provide local typing-state supervision, we add auxiliary heads that predict hand-specific keyboard-position classes and binary typing activity from downsampled encoder frames.

The total loss is:

$$\mathcal{L} = \mathcal{L}_{\text{CTC}} + \lambda_{\text{pos}} \mathcal{L}_{\text{pos}} + \lambda_{\text{act}} \mathcal{L}_{\text{act}}. \quad (1)$$

Here, $\mathcal{L}_{\text{pos}}$ combines the left and right hand position losses, and $\mathcal{L}_{\text{act}}$ is the typing-activity loss. The auxiliary heads are used only during training. We set $\lambda_{\text{pos}} = 0.29$ and $\lambda_{\text{act}} = 0.01$, which were selected based on a hyperparameter sweep.

3.2.5 CTC Beam Search

At inference time, we use a backspace-aware CTC beam-search decoder following the emg2qwerty baseline. The decoder combines CTC prefix scoring with an external 6-gram character (Heafield, 2011) language model and explicitly handles backspace tokens, which are modeled as an explicit CTC label (Appendix A.4). For standard frontend benchmarking, we follow prior work and decode the test session as a whole. For LLM correction, the same decoder is applied to local windows so that the top-20 beam hypotheses can be exported continuously.

3.3 CTC Hypothesis Interface

The CTC beam serves as the interface between the sEMG frontend and the LLM corrector. For each 4-second window, we export the top-20 CTC beam hypotheses as a beam-ordered candidate list. Before export, backspace operations are applied to the hypotheses and the backspace tokens are removed, so that the LLM receives the displayed text (Appendix A.4). Candidates are presented in descending beam order, but we do not expose numerical rank labels or beam scores to the LLM. In addition, we provide the concatenation of the

rank-1 hypotheses from the three windows as a separate rank-1 baseline for correction.

For LLM correction, we group three non-overlapping consecutive windows into one chunk. The prompt contains the top-10 hypotheses from each window as beam-ordered candidate lists, together with the concatenated rank-1 baseline. The LLM is trained to produce the corrected transcript for the whole chunk, allowing it to use nearby context while keeping the prompt short enough for local correction.

Although correction is performed chunk by chunk, evaluation is conducted after reconstructing full sessions. We concatenate the chunk outputs in temporal order and compute CER/WER on the reconstructed session-level sequences. This reduces boundary artifacts from fixed window segmentation while preserving the local inference setting used by the LLM.

3.4 Beam-Aware LLM Correction

3.4.1 Training Objective and Fine-Tuning

Given the chunk-level input described in Section 3.3, the LLM is trained to generate the reference transcript for the whole chunk. We use causal language modeling loss only on the assistant response, masking all prompt tokens.

We fine-tune instruction-tuned Qwen models with low-rank adaptation (LoRA) adapters (Hu et al., 2022) while keeping base weights frozen. Our main model is Qwen3.5-9B (Qwen Team, 2026), and smaller variants are used for ablations. We use rank $r = 16$, $\alpha = 32$, dropout 0.05, maximum sequence length 4096, and deterministic decoding with temperature 0.0. Reasoning-mode generation is disabled, and all models are trained and evaluated with direct text outputs.

3.4.2 Training Data Mixing

Training beams are much cleaner than held-out test beams, which can cause a mismatch for LLM fine-tuning. We therefore construct LLM training data from three beam pools generated by different checkpoints of the same frontend: standard, degraded, and severely degraded, with CERs of 6.21%, 13.25%, and 24.20%. During training, chunks are sampled from these pools with probabilities 0.2, 0.3, and 0.5, respectively.

3.4.3 Prompt Constraints

The prompt constrains the LLM to act as a beam-aware corrector. It instructs the model to output

only the corrected text for all windows concatenated, use the rank-1 baseline as the editing starting point, and modify it only when other beam candidates provide clear evidence. The model is also instructed to preserve punctuation, spacing, and angle-bracketed tokens, avoid adding or deleting unsupported characters, and copy the rank-1 baseline when the beam is ambiguous.

At generation time, we further enforce this behavior with deterministic decoding and by blocking common prompt-format strings such as window headers, bullet markers, and boundary tags. This keeps the correction output anchored to the beam candidates and prevents the model from echoing the prompt structure.

During fine-tuning, we randomly shuffle the candidate list within each window as data augmentation while keeping the rank-1 baseline fixed. This prevents the model from relying only on candidate position and encourages it to compare alternatives against the rank-1 baseline. At inference time, candidates are shown in their original beam order, but no explicit rank labels or beam scores are provided.

3.5 Beam-Aware Output Filter

We apply a lightweight output filter to catch rare invalid generations. The filter rejects outputs with prompt leakage, assistant-format artifacts, extreme length expansion or collapse, or low-support rewrites. In the last case, the candidate is accepted only if it remains sufficiently close to the rank-1 baseline or if its word forms are sufficiently supported by the top- k candidate vocabulary; otherwise the output falls back to rank-1. Detailed filter rules and thresholds are described in the Appendix A.8. In practice, the filter mainly acts as a safeguard for smaller or untuned LLMs, while the fine-tuned main corrector rarely triggers fallback.

4 Results

4.1 Experimental Setup and Evaluation Protocol

We evaluate all models on the default emg2qwerty user split. In the generic setting, a single model is trained on 100 users and tested zero-shot on 8 held-out users. In the personalized setting, the generic frontend is further fine-tuned separately for each held-out user and evaluated on that user’s held-out sessions. The LLM corrector is trained only on the 100-user training split and is shared across both generic and personalized frontend outputs.

Method	Params	Generic CER (%) ↓		Personalized CER (%) ↓	
		Greedy	Beam	Greedy	Beam
emg2qwerty baseline (Sivakumar et al., 2024)	5.29M	55.38±4.10	51.78±4.61	11.28±4.45	6.95±3.61
SplashNet (Hadidi et al., 2025)	2.58M	44.78	35.67±6.79	12.39	5.51±2.81
Tiny Transformer (Mehlman et al., 2025)	2.2M	35.9	–	9.7	–
Small Transformer (Mehlman et al., 2025)	5.4M	35.2	–	7.9	–
Large Transformer (Mehlman et al., 2025)	109M	30.5	–	6.8	–
Ours: CNN–Conformer	9.8M	33.40±6.16	24.14±7.31	8.01±3.24	4.75±2.27
+ Multi-scale CNN	9.8M	32.59±5.96	23.70±7.28	8.21±3.19	4.76±2.32
+ Split-and-share encoder	9.8M	30.58±6.57	20.92±7.72	6.59±2.29	3.46±1.33
+ Keyboard-state auxiliary supervision	9.8M	29.58±5.80	19.94±6.38	6.45±2.40	3.43±1.44

Table 1: Full-session frontend benchmark on emg2qwerty. Each test session is decoded as a whole, matching the standard protocol used by prior frontend systems. Lower CER is better. Transformer baselines are shown with published greedy metrics only. Our variants are cumulative. Where available, $\pm$ denotes the standard deviation across the eight held-out users. The first three variants are single-seed runs, while the final row is averaged over three random seeds: each held-out user’s CER is first averaged across the three seeds, and the mean and standard deviation are then computed over the eight seed-averaged user scores.

We report two protocol-specific evaluations. Table 1 follows the standard full-session frontend benchmark used by prior emg2qwerty systems: each test session is decoded as a whole using backspace-aware CTC beam search with an external character LM. This protocol evaluates first-pass frontend transcription quality and enables comparison with published frontend systems.

Table 2 uses a separate windowed hypothesis-export protocol for LLM correction. The frontend decodes each 4-second window locally, exports the top-20 CTC hypotheses, and the LLM corrects chunks of three non-overlapping consecutive windows using the top-10 candidates from each window. Corrected chunk outputs are concatenated back into complete sessions before scoring.

Because these protocols use different decoding contexts, the two tables should not be read as a single leaderboard. LLM correction is compared only with the windowed rank-1 frontend output produced under the same hypothesis-export protocol. We report Character Error Rate (CER) as the primary metric, together with Word Error Rate (WER), which treats typed Enter events as word boundaries, and substitution, deletion, and insertion rates for error analysis.

4.2 Main Frontend Transcription Results

Table 1 compares frontend transcription results under the full-session benchmark protocol. We include the original emg2qwerty baseline, SplashNet (Hadidi et al., 2025), and raw-EMG Transformer scaling results from Mehlman et al. (2025). Transformer rows are shown only for greedy decod-

ing because beam-search results are not reported.

Our frontend variants are cumulative. The multi-scale CNN mainly improves generic transcription, while split-and-share encoding and keyboard-state auxiliary supervision bring the largest gains across both generic and personalized settings. The full frontend achieves the best reported full-session CER among the compared systems. Under greedy decoding, it also outperforms the 109M-parameter Large Transformer while using far fewer parameters.

These results establish the frontend as a strong first-pass decoder and hypothesis generator. We next evaluate whether its exported window-level CTC beam hypotheses can improve LLM-based correction under the windowed hypothesis-export protocol.

4.3 Beam-Aware LLM Correction Results

Table 2 evaluates LLM correction under the windowed three-window protocol. All rows use session-level scoring: outputs from corrected chunks are concatenated back into complete sessions before CER/WER is computed. The windowed rank-1 frontend baseline is different from the full-session frontend result because the decoder has shorter local context and hypotheses must be exported per window. We therefore compare correction gains only against the corresponding windowed rank-1 baseline.

The untuned Qwen3.5–9B model fails to improve over the windowed rank-1 baseline even after output filtering, showing that direct prompting is unreliable for noisy sEMG transcripts. After

task-specific fine-tuning, Qwen3.5–9B improves both CER and WER in the generic and personalized settings, with larger relative gains in WER. This trend is also consistent at the per-user level, see Appendix Figure 4. This suggests that the fine-tuned LLM more often converts noisy transcripts into correct word-level outputs. A two-sided exact paired sign-flip permutation test on user-level CER confirms the gain for all eight held-out users in both settings ($p = 0.0078$ each; the values are identical because all eight paired differences have the same direction).

As a smaller encoder–decoder baseline, we fully fine-tune a Flan-T5-base corrector (Chung et al., 2024) on the same beam inputs, context, training data, filter, and protocol. It improves WER only modestly, does not consistently improve CER, and passes the output filter less often in the generic and personalized settings (32.3% and 79.0%, respectively) than Qwen3.5–9B (99.3% and 99.9%). The gains therefore do not follow from adding another supervised N-best corrector; task-specific adaptation and model capability both matter.

We also test a matched setting where the LLM corrects full-session decoding output directly, using timestamp-segmented full-session transcripts as the rank-1 input and window-level top-10 hypotheses as local evidence (Appendix A.5). The fine-tuned LLM reduces CER from 20.07% to 18.61% (generic) and from 2.85% to 2.40% (personalized), so correction improves the strongest frontend output rather than only compensating for windowed decoding.

We also report top-10 candidate-selection oracles, which independently select the best candidate for each window from its top-10 beam hypotheses according to CER or WER. The selected window outputs are then concatenated for session-level scoring. These oracles estimate how much recoverable information remains in the beam hypotheses beyond the rank-1 transcript. The oracle results show that useful alternatives are often present below rank 1, leaving headroom for better correction models. Because the oracle is restricted to selecting one candidate per window, it should not be interpreted as an upper bound for generative LLM correction; the LLM can combine and locally revise candidates rather than only choose one beam hypothesis. User-level standard deviations are reported in Appendix Table 6.

Figure 2 shows that substitutions dominate both CER and WER, especially in the generic setting.

Method	Generic (%)		Personalized (%)	
	CER ↓	WER ↓	CER ↓	WER ↓
Windowed rank-1	22.56	58.86	3.42	17.51
Untuned Qwen3.5–9B	22.66	60.15	3.72	18.35
Fine-tuned Flan-T5-base	22.54	57.35	3.60	15.32
Fine-tuned Qwen3.5–9B	19.65	44.09	2.01	9.11
Top-10 Candidate Oracle	16.16	47.68	1.18	7.11

Table 2: Beam-aware LLM correction under the windowed protocol with session-level scoring. All rows use the same windowed hypothesis-export setting; the windowed rank-1 is the within-protocol baseline for measuring LLM correction gains. The Flan-T5-base corrector uses the same beam inputs, context, training data, output filter, and protocol as fine-tuned Qwen3.5–9B. The top-10 candidate oracle selects the best candidate per window from its top-10 hypotheses, separately for CER and WER.

The untuned LLM barely changes the error profile relative to the baseline, while the fine-tuned LLM mainly reduces substitution errors, leading to larger relative improvements in WER than CER. This suggests that many corrected character-level errors affect whole-word recognition.

Appendix Table 13 provides representative correction examples. Successful cases show that the LLM usually makes local edits rather than rewriting the whole chunk: it preserves reliable parts of the rank-1 baseline, while using candidate alternatives to recover missing characters, word fragments, or spacing patterns. Adjacent-window context is also useful when the correct word form is ambiguous within a single window. Failure cases typically occur when the beam evidence is weak or ambiguous, in which case the LLM may prefer a more fluent phrase that is plausible linguistically but inconsistent with the exact typed reference. These examples support the use of beam hypotheses as correction evidence, while also showing the need for stronger beam-support constraints. An edit-level analysis of all LLM insertions and substitutions (Appendix A.11) confirms that beam-supported edits are far more reliable than unsupported ones, with punctuation changes the main source of unsupported harmful edits in the generic setting.

The beam-aware output filter has little effect on the fine-tuned Qwen3.5–9B corrector, whose outputs pass the filter in 99.3% of generic examples and 99.9% of personalized examples. As shown in Appendix Table 10, the filter mainly acts as a safeguard for weaker or untuned correctors, substantially reducing severe generation failures.

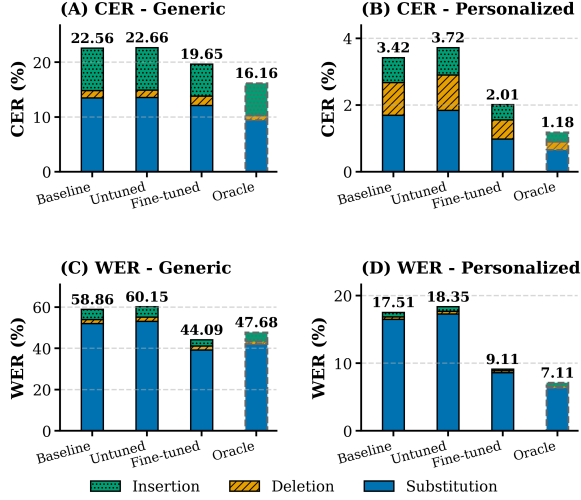


Figure 2: Error breakdown under the windowed three-window protocol with session-level scoring. Untuned and Fine-tuned denote the Qwen3.5-9B corrector before and after task-specific fine-tuning, matching Table 2. Oracle selects the best candidate per window from its top-10 beam hypotheses according to CER or WER. Bars are decomposed into substitutions, deletions, and insertions.

4.4 Does the LLM Memorize Training Phrases?

Since some words and phrases in the test set also appear in the training set, a possible concern is that the fine-tuned LLM corrector may benefit from memorizing training phrases rather than learning correction strategy. To examine this concern, we compute the maximum character 10-gram overlap between test windows and any training reference. The overlap score measures the fraction of test character 10-grams that also appear in the most similar training reference. The test inputs are then split by the median overlap score.

Table 3 shows that the corrector improves both low and high overlap groups. The average overlap scores are 0.067 and 0.221 for the low and high overlap groups, respectively, indicating that even the high overlap group is not dominated by duplicate training phrases. The larger gain in the high overlap group suggests that lexical familiarity can help correction, while the improvement on low overlap inputs indicates that the overall gains are not explained solely by memorized or duplicate training phrases.

Table 4 reports exact token error rates on rare and task-unseen reference tokens, grouped by token frequency in the corrector’s fine-tuning references (task-unseen tokens never appear there,

Subset	Mean overlap	Rank-1 CER	LLM CER
Low overlap	0.067	23.22	21.25
High overlap	0.221	22.21	18.37

Table 3: Generic CER by training-reference overlap. Each test input is grouped by its maximum character 10-gram overlap with any training reference.

Token subset	n	Generic (%)		Personalized (%)	
		Rank-1	LLM	Rank-1	LLM
Task-unseen	594	90.9	86.7	52.0	51.0
Rare (1–5)	330	80.6	61.5	31.5	18.2

Table 4: Exact token error rate (percentage of reference tokens not recovered exactly) on rare and task-unseen tokens, grouped by token frequency in the LLM fine-tuning references. Task-unseen tokens never occur in those references; rare tokens occur 1–5 times. Lower is better.

though they may occur in base-model pretraining data). The corrector reduces errors on both subsets, with larger reductions on rare tokens, indicating that lexical familiarity helps but the gains are not limited to forms seen during fine-tuning. It also exactly recovers naturally occurring typos such as “offesnsive”, “protocol”, “avaialble”, and “tehhe”. Still, over-correction remains: among task-unseen tokens already correct in the rank-1 output, the LLM changed 7/54 (generic) and 30/285 (personalized) toward more common spellings (e.g., changing “itdential” to “identical” or “mistaks” to “mistakes”).

4.5 Ablation of Beam-Aware LLM Correction

We further ablate the main factors in the LLM correction stage using the same exported window-level hypotheses as Table 2. Figure 3 varies chunk size, beam size, and LLM scale, with full numbers reported in Appendix Table 9.

The ablations show three main trends. First, local context is important: increasing the chunk size from one to three windows substantially reduces CER, while longer chunks bring small or inconsistent gains. Second, multiple beam candidates are useful: performance improves from top-1 to top-5 and top-10, then largely saturates at top-20. This indicates that the correction gains come partly from exposing decoder uncertainty beyond the best-path transcript, and that most useful alternatives are already covered by the top-10 candidates. Third, larger LLM correctors improve CER, especially in

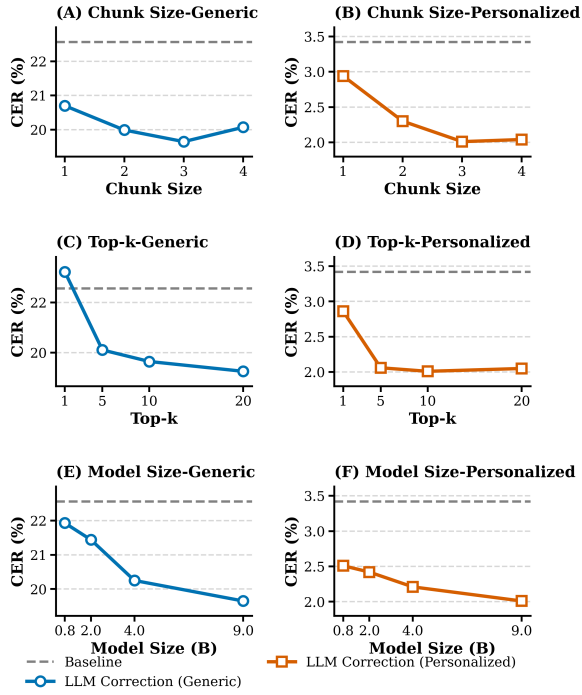


Figure 3: Ablation trends for beam-aware LLM correction. Dashed lines indicate the rank-1 frontend baseline under the same windowed protocol. Lower is better.

the generic setting, although gains diminish with scale. Together with the untuned result in Table 2, this suggests that both model capacity and task-specific fine-tuning are needed for reliable beam-aware correction.

5 Discussion and Conclusion

Our results suggest that language-level correction for sEMG typing should be constrained by hypotheses generated from the same sEMG input. Unlike ASR post-correction, sEMG typing correction must preserve the exact typed sequence, including keyboard events and non-lexical fragments; unrestricted language-model rewriting can therefore be harmful. This makes direct rank-1 correction unreliable, especially when the first-pass transcript can be far from the reference.

The correction stage depends directly on the hypotheses produced by the frontend. In our system, the CNN-Conformer serves both as a first-pass decoder and as the source of candidate alternatives for the LLM. A stronger frontend gives the LLM a cleaner rank-1 baseline, while its beam-ordered candidate lists can still preserve locally useful alternatives, such as missing characters, word fragments, or spacing patterns. The LLM then uses these alternatives together with the rank-1 baseline

and local context to make beam-supported corrections.

The ablation results support this interface design. Local context helps resolve spelling and word-boundary errors, while lower-ranked beam candidates provide information that is unavailable from the rank-1 transcript alone. At the same time, the gains saturate with a compact beam, suggesting that useful alternatives are usually concentrated near the top of the candidate list. The training-overlap analysis further indicates that the fine-tuned corrector is not simply memorizing training phrases, since it also improves inputs with limited phrase-level similarity.

The corrector is also practical at inference time: measured with batch size 1 on a single RTX 5090, it adds about 0.36 seconds per three-window chunk on top of the shared frontend decoding cost, with an expected delay of about 3.84 seconds after a segment ends, reducible to about 2.84 seconds with a two-second-stride sliding window (Appendix A.2).

Overall, our results show that sEMG typing benefits from connecting a strong signal frontend with a beam-aware LLM corrector through an explicit CTC hypothesis interface. Future work should improve both hypothesis generation and correction constraints so that LLMs can better exploit useful beam alternatives while avoiding unsupported rewrites.

Limitations

Several limitations remain. First, the proposed correction method is bounded by the quality of the CTC hypothesis set. When the correct characters or word fragments are absent from the beam, the LLM has limited reliable evidence for recovery. The oracle results indicate that useful information can remain in the beam, but better correction requires more effective use of beam alternatives rather than freer language-model rewriting.

Second, sEMG typing transcription requires exact recovery of the typed sequence. Although fine-tuning, prompt constraints, and output filtering reduce severe generation failures, the LLM can still over-correct toward more common words or phrases when the beam evidence is ambiguous. This is problematic for typed text, where spelling variants, punctuation, non-lexical fragments, and even user errors may be part of the intended output. Moreover, the benchmark mainly contains lowercase dictionary words and Wikipedia sentences

with basic punctuation, and does not include a dedicated controlled test set for names, abbreviations, URLs, passwords, or code-like strings; our rare-token analysis (Table 4) covers only naturally occurring cases, and we do not claim reliable exact recovery for these categories.

Third, our LLM correction stage operates on fixed three-window chunks. This keeps the prompt compact and supports local correction, but it may miss longer-range context and remain sensitive to window-boundary errors. More flexible streaming correction, memory across chunks, or confidence-aware chunking may improve robustness.

Finally, our experiments are limited to the emg2qwerty benchmark with English QWERTY typing on a physical keyboard, using the official split with eight held-out users; leave-one-user-out evaluation across more folds would further quantify the effect of training-set size and cross-user variance. The current corrector also uses a relatively large LLM and is not optimized for fully on-device deployment. Future work should evaluate smaller correctors or distillation, as well as broader settings including other languages, keyboard layouts, users, and keyboard-free typing scenarios.

Ethical Considerations

The original emg2qwerty data collection was approved by an Institutional Review Board and obtained informed consent from participants; the released data were de-identified, as described by Sivakumar et al. (2024). Our experiments use only this public benchmark with de-identified data. sEMG typing systems can potentially reveal private text input. Deployment should require explicit consent, local processing when possible, secure storage of EMG and keystroke data, and clear user control over when transcription is active. In particular, deployments should guard against failure modes in which the system transcribes unintended muscle activity, remains active without clear user awareness, or retains raw EMG or keystroke data in ways that could expose private text. Our experiments use a public benchmark with de-identified data and evaluate prompted text transcription rather than covert inference.

Code Availability

The source code for this work is publicly available at <https://github.com/Hunyxx/beam-aware-semg-typing>.

Acknowledgments

This work was supported by the Japan Agency for Medical Research and Development (AMED) under Grant Number JP24wm0625403 and by JST BOOST, Japan, under Grant Number JPMJBS2430.

References

- Christoph Amma, Thomas Krings, Jonas Böer, and Tanja Schultz. 2015. [Advancing muscle-computer interfaces with high-density electromyography](#). In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI '15*, page 929–938, New York, NY, USA. Association for Computing Machinery.
- Manfredo Atzori, Arjan Gijsberts, Claudio Castellini, Barbara Caputo, Anne-Gabrielle Mittaz Hager, Simone Elsig, Giorgio Giatsidis, Franco Bassetto, and Henning Müller. 2014. [Electromyography data for non-invasive naturally-controlled robotic hand prostheses](#). *Scientific Data*, 1:140053.
- Meghna Roy Chowdhury, Shreyas Sen, and Yi Ding. 2026. [From muscle to text with myotext: semg to text via finger classification and transformer-based decoding](#). *Preprint*, arXiv:2601.03098.
- Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, and 16 others. 2024. [Scaling instruction-finetuned language models](#). *Journal of Machine Learning Research*, 25(70):1–53.
- Rui Dong, David Smith, Shiran Dudy, and Steven Bedrick. 2019. [Noisy neural language modeling for typing prediction in BCI communication](#). In *Proceedings of the Eighth Workshop on Speech and Language Processing for Assistive Technologies*, pages 44–51, Minneapolis, Minnesota. Association for Computational Linguistics.
- Yu Du, Wenguang Jin, Wentao Wei, Yu Hu, and Weidong Geng. 2017. [Surface emg-based inter-session gesture recognition enhanced by deep domain adaptation](#). *Sensors*, 17(3).
- Ethan Eddy, Evan Campbell, Scott Bateman, and Erik Scheme. 2024. [Big data in myoelectric control: large multi-user models enable robust zero-shot emg-based discrete gesture recognition](#). *Frontiers in Bioengineering and Biotechnology*, Volume 12 - 2024.
- Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. 2006. [Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks](#). In *Proceedings of*

- the 23rd International Conference on Machine Learning, ICML '06, page 369–376, New York, NY, USA. Association for Computing Machinery.
- Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, and Ruoming Pang. 2020. [Conformer: Convolution-augmented transformer for speech recognition](#). *Preprint*, arXiv:2005.08100.
- Nima Hadidi, Jason Chan, Ebrahim Feghhi, and Jonathan Kao. 2025. [Splashnet: Split-and-share encoders for accurate and efficient typing with surface electromyography](#). In *Advances in Neural Information Processing Systems*, volume 38, pages 166117–166141. Curran Associates, Inc.
- Awni Hannun, Ann Lee, Qiantong Xu, and Ronan Collobert. 2019. [Sequence-to-sequence speech recognition with time-depth separable convolutions](#). *Preprint*, arXiv:1904.02619.
- Kenneth Heafield. 2011. [KenLM: Faster and smaller language model queries](#). In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 187–197, Edinburgh, Scotland. Association for Computational Linguistics.
- Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. [LoRA: Low-rank adaptation of large language models](#). In *International Conference on Learning Representations*.
- Yuchen Hu, Chen Chen, Chengwei Qin, Qiushi Zhu, Eng Siong Chng, and Ruizhe Li. 2024. [Listen again and choose the right answer: A new paradigm for automatic speech recognition with large language models](#). In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 666–679, Bangkok, Thailand. Association for Computational Linguistics.
- Piotr Kaczmarek, Tomasz Mańkowski, and Jakub Tomczyński. 2019. [putemg—a surface electromyography hand gesture recognition dataset](#). *Sensors*, 19(16).
- Yichong Leng, Xu Tan, Linchen Zhu, Jin Xu, Renqian Luo, Linquan Liu, Tao Qin, Xiangyang Li, Edward Lin, and Tie-Yan Liu. 2021. [Fastcorrect: Fast error correction with edit alignment for automatic speech recognition](#). In *Advances in Neural Information Processing Systems*, volume 34, pages 21708–21719. Curran Associates, Inc.
- Rao Ma, Mark J. F. Gales, Kate M. Knill, and Mengjie Qian. 2023. [N-best t5: Robust asr error correction using multiple input hypotheses and constrained decoding space](#). In *Proc. Interspeech 2023*.
- Rao Ma, Mengjie Qian, Mark Gales, and Kate Knill. 2025. [Asr error correction using large language models](#). *IEEE Transactions on Audio, Speech and Language Processing*, 33:1389–1401.
- Nick Mehlman, Jean-Christophe Gagnon-Audet, Michael Shvartsman, Kelvin Niu, Alexander H Miller, and Shagun Sodhani. 2025. [Scaling and distilling transformer models for sEMG](#). *Transactions on Machine Learning Research*.
- Qwen Team. 2026. [Qwen3.5: Towards native multi-modal agents](#).
- Sasha Salter, Richard Warren, Collin Schlager, Adrian Spurr, Shangchen Han, Rohin Bhasin, Yujun Cai, Peter Walkington, Anuoluwapo Bolarinwa, Robert Wang, Nathan Danielson, Josh Merel, Eftychios Pnevmatikakis, and Jesse Marshall. 2024. [emg2pose: A large and diverse benchmark for surface electromyographic hand pose estimation](#). In *Advances in Neural Information Processing Systems*, volume 37, pages 55703–55728. Curran Associates, Inc.
- Viswanath Sivakumar, Jeffrey Seely, Alan Du, Sean R Bittner, Adam Berenzweig, Anuoluwapo Bolarinwa, Alexandre Gramfort, and Michael I Mandel. 2024. [emg2qwerty: A large dataset with baselines for touch typing using surface electromyography](#). In *Advances in Neural Information Processing Systems*, volume 37, pages 91373–91389. Curran Associates, Inc.
- Chao-Han Huck Yang, Yile Gu, Yi-Chieh Liu, Shalini Ghosh, Ivan Bulyko, and Andreas Stolcke. 2023. [Generative speech recognition error correction with large language models and task-activating prompting](#). In *2023 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 1–8.
- Jehan Yang, Maxwell Soh, Viviana Lieu, Douglas J Weber, and Zackory Erickson. 2024. [Emgbench: Benchmarking out-of-distribution generalization and adaptation for electromyography](#). In *Advances in Neural Information Processing Systems*, volume 37, pages 50313–50342. Curran Associates, Inc.

A Appendix

A.1 Training and Runtime Details

We provide additional implementation details for both stages of the proposed system. The CNN–Conformer frontend was trained for 200 epochs with AdamW, using an initial learning rate of 3×10^{-4} , weight decay 0.1, and batch size 320. We used a linear-warmup cosine-annealing schedule with 5 warmup epochs, warmup start learning rate 10^{-8} , and minimum learning rate 10^{-6} . The frontend was trained on raw sEMG windows of 4 seconds, corresponding to 8000 samples at 2 kHz, with 900 ms past context and 100 ms future context.

For the beam-aware LLM corrector, we fine-tuned Qwen3.5–9B with LoRA for 5 epochs using AdamW with learning rate 10^{-4} , weight decay 0.1, and batch size 8, together with a cosine learning-rate schedule. The LoRA configuration used rank

16, scaling factor $\alpha = 32$, and dropout 0.05. Gradient checkpointing was enabled to reduce GPU memory usage. Following the main setting in the paper, each LLM input contains the top-10 CTC beam candidates from three non-overlapping consecutive windows.

All training was conducted on two NVIDIA H100 GPUs. Training the generic CNN-Conformer frontend took approximately 15 hours. For personalized frontend adaptation, fine-tuning took about 2 hours for all 8 test users, and beam-search evaluation required an additional 30 minutes. LoRA fine-tuning for the beam-aware LLM corrector took approximately 3 hours, and evaluation took about 2 minutes.

A.2 Inference Latency

We measure inference latency with batch size 1 on a single NVIDIA RTX 5090 GPU (Table 5). The frontend uses the same non-overlapping 4-second windows and beam decoder as the emg2qwerty baseline, so window accumulation and frontend decoding are shared costs rather than latency introduced by the LLM. The LLM correction stage adds about 0.36 seconds per three-window chunk once the frontend hypotheses are available. During continuous typing, preceding windows have already been processed, so the three-window input provides historical context instead of introducing a new 12-second computation delay; the delay after a segment ends is approximately 1.84–5.84 seconds, with an expected delay of about 3.84 seconds. Most of the latency comes from the fixed 4-second window and CTC beam search. A 4-second sliding window with a 2-second stride would reduce the estimated average delay to about 2.84 seconds; shorter windows could reduce it further, although their accuracy–latency trade-off requires separate evaluation.

A.3 Comparison of Full-Session and Windowed Protocols

Table 7 summarizes the difference between the two evaluation protocols used in Table 1 and Table 2. The full-session frontend benchmark decodes each test session as a whole and is used to compare the frontend with prior emg2qwerty systems. The windowed hypothesis-export protocol decodes each 4-second window locally so that CTC beam hypotheses can be exported for LLM correction. Its rank-1 outputs are concatenated back into complete sessions before computing CER, matching

(a) Shared frontend latency per 4-second window

Component	Median latency (ms)
Preprocessing and normalization	5
CNN-Conformer forward	3
CTC beam search	1475
Total frontend	1483

(b) Additional LLM latency per three-window chunk

Component	Median latency (ms)
Prompt construction	1
Qwen3.5-9B generation	357
Output filtering	< 1
Total additional	358

Table 5: Median inference latency measured with batch size 1 on a single RTX 5090. (a) Per-window frontend cost, which is shared with the emg2qwerty baseline decoding setup. (b) Additional cost of the LLM correction stage per three-window chunk.

the evaluation setting used for Table 2. Therefore, LLM correction results should be compared with the windowed rank-1 baseline under the same protocol, while full-session results should be used for frontend benchmarking.

A.4 Backspace Handling

Backspace is modeled as an explicit CTC label in the frontend. During beam search, the decoder preserves backspace tokens in the raw keystroke sequence while rolling back the corresponding character-LM state, following the emg2qwerty baseline decoder. The full-session frontend results in Table 1 therefore evaluate the original keystroke sequence, including backspaces.

For LLM correction, evaluation uses displayed text. Before the LLM stage, each backspace is applied by deleting the preceding character in both the beam hypotheses and the references, and the backspace tokens are then removed. The LLM is thus trained and evaluated only on the text that remains after these deletions.

A.5 Matched Correction of Full-Session Frontend Output

Table 8 evaluates whether the LLM corrector improves the strongest frontend output directly. Each test session is first decoded as a whole, following the full-session protocol of Table 1. The resulting rank-1 transcript is divided into 4-second segments using character timestamps, and this full-session text serves as the rank-1 input in the LLM prompt, while the remaining top-10 candidates

Method	Generic (%)		Personalized (%)	
	CER ↓	WER ↓	CER ↓	WER ↓
Windowed rank-1	22.56 ± 6.24	58.86 ± 9.43	3.42 ± 1.32	17.51 ± 4.62
Untuned Qwen3.5-9B	22.66 ± 6.19	60.15 ± 9.58	3.72 ± 1.22	18.35 ± 4.18
Fine-tuned Flan-T5-base	22.54 ± 6.36	57.35 ± 10.40	3.60 ± 1.09	15.32 ± 3.84
Fine-tuned Qwen3.5-9B	19.65 ± 7.07	44.09 ± 11.78	2.01 ± 0.86	9.11 ± 2.91
Top-10 Candidate Oracle	16.16 ± 5.66	47.68 ± 10.85	1.18 ± 0.71	7.11 ± 3.25

Table 6: Expanded beam-aware LLM correction results corresponding to Table 2. All rows use the windowed hypothesis-export protocol with session-level scoring. Values are mean ± standard deviation across the eight held-out users.

Protocol	Generic CER (%) ↓	Personalized CER (%) ↓
Full-session	19.94 ± 6.38	3.43 ± 1.44
Windowed export	22.56 ± 6.24	3.42 ± 1.32

Table 7: Comparison of the frontend rank-1 outputs under the two evaluation protocols used in the paper. Full-session corresponds to the frontend benchmark in Table 1. Windowed export corresponds to the hypothesis-export setting used as the within-protocol rank-1 baseline in Table 2. Values are mean ± standard deviation across the eight held-out users.

Method	Generic (%)		Personalized (%)	
	CER ↓	WER ↓	CER ↓	WER ↓
Full-session frontend	20.07	52.47	2.85	12.69
+ Fine-tuned LLM	18.61	47.69	2.40	9.08

Table 8: Matched correction of the full-session frontend output. Both rows use displayed-text evaluation, with backspaces applied to both predictions and references before scoring. Unlike Table 1, which reports raw-keystroke CER averaged over three random seeds, this experiment uses a single frontend checkpoint; the full-session baseline here should only be read as the matched baseline for LLM correction and should not be directly compared with Table 1.

come from independent window-level beam decoding and serve as local correction evidence. The fine-tuned LLM improves over this matched full-session baseline in both settings, confirming that the correction gains are not an artifact of windowed hypothesis export.

A.6 Expanded LLM Correction Results

Table 6 provides the expanded version of the beam-aware LLM correction results in Table 2, including user-level standard deviations. All rows follow the same windowed hypothesis-export protocol: CTC hypotheses are exported for each window, the LLM corrects three-window chunks, and the corrected

chunk outputs are concatenated back into complete sessions before computing CER and WER. The standard deviations summarize variation across the eight held-out users.

A.7 Ablation Study of Beam-Aware LLM Correction

Table 9 provides additional ablation results for the beam-aware LLM correction stage. We vary three factors: the number of adjacent windows included in each correction chunk, the number of CTC beam candidates provided to the LLM, and the size of the LLM corrector. Unless otherwise specified, the default setting uses Qwen3.5-9B with three-window chunks and top-10 beam candidates.

These ablations support the main correction results reported in Section 4.3. Using multiple beam candidates clearly improves over top-1 correction, showing that beam-ordered CTC candidate lists contain useful alternatives for the LLM. Increasing the chunk size from one to three windows improves correction by providing local context, while larger chunks bring limited additional gains. The untuned LLM performs substantially worse than the frontend output, confirming that task-specific fine-tuning is necessary for reliable correction.

A.8 Output Filter Analysis

Table 10 reports the effect of the beam-aware output filter. For each model, we compare the direct LLM output with the final output after filtering. The filter checks for invalid generations such as prompt leakage, abnormal length changes, and unsupported rewrites, and falls back to the rank-1 frontend output when such failures are detected.

More concretely, the implemented filter first rejects prompt-format leakage and assistant-role artifacts, then applies character-length and word-count guards relative to the rank-1 baseline. For lexical support, it builds a support vocabulary from

Setting	Chunk	Top- k	Generic (%)		Personalized (%)	
			CER ↓	WER ↓	CER ↓	WER ↓
<i>Effect of chunk size, with top-k = 10 and Qwen3.5-9B</i>						
Chunk 1	1	10	20.70	51.57	2.94	14.08
Chunk 2	2	10	19.99	46.77	2.30	10.53
Full setting (default)	3	10	19.65	44.09	2.01	9.11
Chunk 4	4	10	20.07	43.31	2.04	8.72
<i>Effect of beam candidates, with chunk size = 3 and Qwen3.5-9B</i>						
Top-1	3	1	23.22	48.97	2.86	11.26
Top-5	3	5	20.11	44.92	2.06	9.12
Full setting (default)	3	10	19.65	44.09	2.01	9.11
Top-20	3	20	19.26	43.35	2.05	9.28
<i>Effect of LLM size, with chunk size = 3 and top-k = 10</i>						
Qwen3.5-0.8B	3	10	21.93	47.51	2.51	10.29
Qwen3.5-2B	3	10	21.44	46.39	2.42	10.03
Qwen3.5-4B	3	10	20.25	45.43	2.21	9.71
Qwen3.5-9B (default)	3	10	19.65	44.09	2.01	9.11
Untuned Qwen3.5-9B	3	10	22.66	60.15	3.72	18.35

Table 9: Ablation of the beam-aware LLM correction stage. We vary chunk size, the number of beam candidates, and LLM size while keeping the remaining settings fixed. The full setting uses Qwen3.5-9B with three-window chunks and top-10 beam candidates. Shaded rows indicate the default configuration. Bold numbers indicate the best result within each ablation block. Lower is better.

Model	Output	Generic			Personalized		
		CER (%) ↓	WER (%) ↓	Pass Rate	CER (%) ↓	WER (%) ↓	Pass Rate
Fine-tuned Qwen3.5-9B	LLM raw	19.72	44.16	99.3%	2.02	9.05	99.9%
	Filtered	19.65	44.09		2.01	9.11	
Fine-tuned Qwen3.5-0.8B	LLM raw	22.27	47.67	98.0%	2.52	10.29	99.9%
	Filtered	21.93	47.51		2.51	10.29	
Untuned Qwen3.5-9B	LLM raw	41.80	84.30	50.5%	28.12	47.66	52.8%
	Filtered	22.66	60.15		3.72	18.35	

Table 10: Ablation of the beam-aware output filter on session-level LLM correction outputs. “LLM raw” denotes the direct LLM output before post-filtering, and “Filtered” denotes the final output after applying the beam-aware output filter. Filter pass rate is reported as a percentage over total examples for each setting. Baseline rows are omitted because this comparison focuses only on the effect of filtering the same LLM outputs. Lower is better.

the rank-1 baseline and all top- k candidates in the chunk, computes per-word Levenshtein similarity against that vocabulary, and accepts the LLM output if at least one of three conditions holds: character-level similarity to rank-1 is at least 0.72, the fraction of beam-supported output words is at least 0.55, or the mean word-support score is at least 0.72. A word is considered supported if it exactly matches some beam word when its length is at most 2, reaches similarity ≥ 0.67 when its length is 3-4, or reaches similarity ≥ 0.66 when its length is at least 5. These thresholds were selected empirically through preliminary experiments, with the goal of rejecting clearly invalid generations while retaining valid corrections, and were then fixed for all reported test experiments.

For the fine-tuned Qwen3.5-9B corrector, the

filter has only a small effect because most outputs already satisfy the format and beam-support constraints. For the untuned LLM, filtering substantially reduces CER and WER, showing that the filter mainly acts as a safeguard against severe generation failures.

A.9 Per-user Chunk-level Error Distributions

Figure 4 reports per-user chunk-level error distributions under the LLM correction protocol. Each box is computed from the three-window test chunks of one held-out user, with CER and WER measured separately for the rank-1 frontend output and the LLM-corrected output. LLM correction lowers the mean CER and WER for most held-out users in both generic and personalized settings, showing that the overall gains are not concentrated

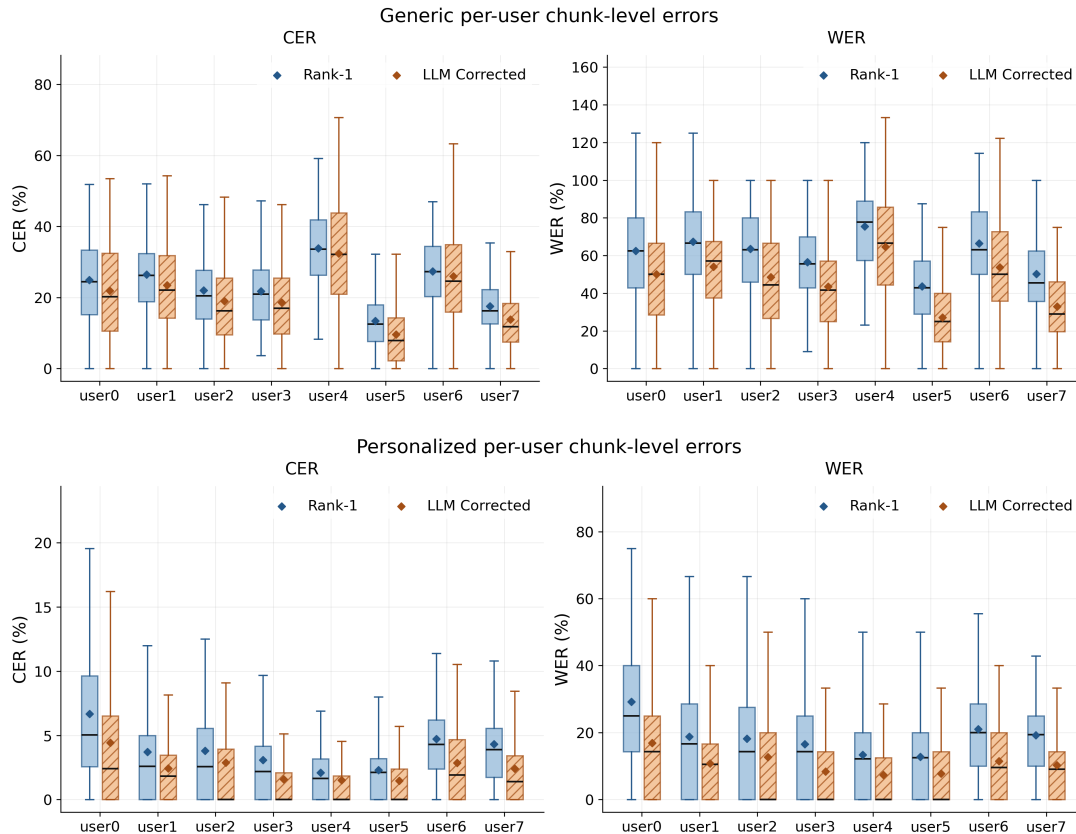


Figure 4: Per-user chunk-level CER and WER distributions in the generic (top) and personalized (bottom) settings. Boxes show distributions across chunks, and diamonds indicate user-level means.

in a single subject. The improvements are more consistent after personalization, while the generic setting shows larger user-dependent variance and occasional high-error chunks. Overall, the per-user results indicate broad benefits from beam-aware correction, with remaining errors mainly concentrated in difficult generic cases.

A.10 Chunk-level Correction Behavior

Table 11 further analyzes how LLM correction changes individual chunks under the windowed correction protocol. For each chunk, we compare the rank-1 frontend output with the LLM-corrected output and categorize whether CER or WER is reduced, unchanged, or increased.

As shown in the upper panel of Table 11, the LLM improves more chunks than it worsens in both generic and personalized settings. In the generic setting, the corrector changes the noisier rank-1 transcripts more aggressively, leading to a higher CER worsened rate. However, the WER worsened rate remains much lower, indicating that many character-level degradations do not introduce additional word-level errors. After personalization, cor-

rection becomes more conservative: many chunks remain unchanged, and the worsened rates are substantially lower.

The lower panel of Table 11 shows that the net error reduction is positive for both CER and WER in both settings, confirming that the improvements outweigh the degradations on average. Unsupported word outputs are much rarer after personalization, and the low filter fallback rates indicate that the reported gains are not mainly caused by rule-based fallback.

A.11 Edit-level Error Analysis

We examine every insertion and substitution introduced by the fine-tuned LLM corrector under the main top-10 setting. An edit is *beam-supported* when the identical edit appears in at least one top- k hypothesis in the chunk, and *helpful* when it reduces the character distance to the reference. Table 12 reports the resulting statistics.

Beam-supported edits are much more reliable than unsupported edits, showing the benefit of exposing multiple hypotheses to the corrector. Some unsupported edits are also helpful, suggesting that

(a) Outcome rates				
Outcome	Generic		Personalized	
	CER	WER	CER	WER
Improved by LLM	61.6%	63.5%	49.0%	46.5%
Unchanged	14.2%	30.3%	42.0%	47.9%
Worsened by LLM	24.2%	6.2%	9.0%	5.6%

(b) Magnitude and safety diagnostics				
Metric	Generic		Personalized	
	CER	WER	CER	WER
<i>Error change (pp)</i>				
Gain on improved	7.0	23.7	3.5	19.1
Loss on worsened	5.7	17.9	3.7	18.3
Net reduction	2.9	13.9	1.4	7.8
<i>Safety diagnostics</i>				
Unsupported words	10.3%		0.6%	
Filter fallback	0.5%		0.1%	

Table 11: Chunk-level correction behavior under the windowed LLM correction protocol. In panel (a), values are percentages over evaluated chunks, and improved, unchanged, and worsened are computed separately using CER and WER. In panel (b), error changes are reported in percentage points (pp). Net reduction is the average per-chunk error reduction from rank-1 to LLM-corrected outputs.

the LLM can combine partial evidence across candidates instead of only copying a single candidate. The personalized corrector is more conservative and corrupts fewer already-correct chunks. Punctuation changes remain the main source of unsupported harmful edits in the generic setting; angle-bracket token changes are rare and disappear after personalization.

A.12 Qualitative Examples

Table 13 shows representative examples of beam-aware LLM correction. We include both successful cases and failure cases to illustrate how the corrector uses the top-10 CTC hypotheses. The examples are selected from the session-level correction outputs.

The first two cases show typical successful patterns. In the beam-supported correction example, the LLM recovers words that are corrupted in the rank-1 output but appear in lower-ranked beam candidates. In the cross-window composition example, the correction requires combining partial evidence across adjacent windows. The last two cases show common failure modes, including language-prior over-correction and over-editing caused by misleading lower-ranked beam candidates.

Metric	Generic	Personalized
Helpful beam-supported edits	65.3%	75.5%
Helpful unsupported edits	34.7%	46.7%
Correct rank-1 chunks corrupted	29.4%	13.0%
WER improved, CER worsened	6.8%	0.4%
Unsup. harmful punct. changes	41.3%	11.9%

Table 12: Edit-level analysis of LLM-introduced insertions and substitutions under the main top-10 setting. An edit is beam-supported if the identical edit appears in at least one top- k hypothesis of the chunk, and helpful if it reduces the character distance to the reference. Generic/personalized counts: 2592/3968 and 778/1030 beam-supported edits, 1550/4468 and 122/261 unsupported edits, 5/17 and 43/332 already-correct rank-1 chunks corrupted, 92/1357 and 5/1357 chunks with improved WER but worsened CER, and 275/666 and 13/109 unsupported harmful punctuation changes. Three angle-bracket token changes occurred in the generic setting, all unsupported and incorrect; none occurred after personalization.

A.13 Prompt Template

We use the following prompt template for beam-aware LLM correction in the main top-10 setting. The actual prompt follows the Qwen chat format; for readability, we omit model-specific chat tokens here. The placeholders in `<INPUT>` and `<rank1_concat>` are filled with the top-10 candidates and the concatenated rank-1 output from the corresponding three-window chunk. Within each window, candidates are listed in beam order at inference time, but rank labels and beam scores are not shown. The rank-1 hypotheses are additionally concatenated and provided as a baseline reference to guide the LLM toward conservative edits. In the ablation experiments, the number of provided candidates is varied as top- k .

The prompt was designed around the exact-recovery requirements of the task: the rank-1 baseline serves as a conservative starting point, edits are allowed only when supported by other beam candidates, transcription formatting must be preserved, and the model is instructed to retain the rank-1 text when evidence is ambiguous. Its wording was iteratively refined during development to reduce over-correction and formatting leakage, and was then fixed for all reported test experiments.

Type	Example and analysis
Beam-supported correction	Reference: spiritual balloon <ENTER> electoral passing sensors tutorial Rank-1: spiritual tallon <ENTER> electoral passint sensors tutorial LLM: spiritual balloon <ENTER> electoral passing sensors tutorial Relevant beam evidence: W1 rank-2 spiritual balloon W3 rank-3 ng sensors tutorial Diagnosis: The LLM output exactly matches the reference. The correction of the first phrase is directly supported by a lower-ranked candidate, and the correction of the word ending in the later phrase is also supported by lower-ranked beam evidence. This illustrates how exposing top-10 hypotheses can recover information missing from the rank-1 frontend output.
Cross-window composition	Reference: rs pendant curious potato <ENTER> reed childhood teaches a Rank-1: rs pendant curious porato <ENTER> reed cchildhood teaches a LLM: rs pendant curious potato <ENTER> reed childhood teaches a Relevant beam evidence: W1 rank-1 rs pendant curious p W2 rank-5 otato <ENTER> reed c W3 rank-2 hildhood teaches a Diagnosis: The correction requires combining partial evidence across adjacent windows. The target word is recoverable by composing the end of the first window with the beginning of the second window, and the duplicated boundary character in the rank-1 output is removed using evidence from the following window. This illustrates why chunk-level correction is useful for window-boundary errors.
Language-prior over-correction	Reference: opers churches <ENTER> characterstic vertical Rank-1: opers churchex <ENTER> charactersto vertical LLM: opers churches <ENTER> characteristic vertical Relevant beam evidence: W1 rank-1 opers churche W2 rank-5 s <ENTER> character W3 rank-2/8 tic vertical Diagnosis: The correction to the plural word is supported by evidence across adjacent windows. However, the LLM also rewrites the misspelled typed sequence to a more common spelling by inserting an additional character. This improves linguistic fluency but hurts exact keystroke transcription, showing a limitation of language-prior correction when the reference contains typed spelling variants or errors.
Over-editing with misleading beam evidence	Reference: uments chan <ENTER> abroad franchise charms inve Rank-1: uments chan <ENTER> abrhad franchise charms inve LLM: uments dish <ENTER> abroad franchise charms inve Relevant beam evidence: W1 rank-1 uments chan <ENTER> abr W1 rank-8 uments dish <ENTER> abr W2 rank-1 had franchise W3 rank-1 charms inve Diagnosis: The rank-1 output preserves the reference span in the first window but contains a boundary error around the following word. The LLM repairs the boundary phrase into a more fluent sequence, but it also replaces the correct span with a lower-ranked and incorrect candidate from the beam. This shows that exposing top-10 hypotheses can help correction, but misleading lower-ranked candidates may also encourage over-editing.

Table 13: Qualitative examples of beam-aware LLM correction under the main top-10 setting. <ENTER> denotes the typed Enter key in the target sequence, not a formatting line break. We analyze whether corrected spans are supported by the chunk-level hypothesis set rather than assigning each output token to a unique beam candidate.

Prompt template for beam-aware LLM correction

System message:

You are a beam-aware transcription corrector. Each window shows multiple candidate transcriptions from a beam decoder. Combine evidence across candidates within and across windows, and output the corrected text for all windows concatenated, with no separators or labels. Preserve punctuation and angle-bracketed tokens. If a window is unrecoverable, copy the primary rank-1 baseline verbatim.

User message:

TASK: Combine beam-decoded candidates into the corrected transcription.

HARD RULES

- 1) Output the corrected text for all windows concatenated, with no separators between windows.
- 2) Use PRIMARY as your starting point; modify only where other candidates clearly suggest a different form.
- 3) Preserve punctuation, spacing, and angle-bracketed tokens, such as <NO_CONTEXT>, exactly as they appear in candidates.
- 4) Do not add or delete characters beyond what the candidates suggest.
- 5) When unsure or garbled, copy PRIMARY verbatim.
- 6) Output the corrected text only. Do not echo window headers, dashes, labels, or list formatting.

Examples (simplified, single window):

Input candidates:

```
- helo worl
- hello world
- hello worl
PRIMARY: helo worl
Output: hello world
```

Input candidates:

```
- roveel auth
- roveel oath
- roveel auth
PRIMARY: roveel auth
Output: roveel auth
```

Input format:

```
<INPUT>
=== Window 1 ===
- <candidate_1>
- <candidate_2>
...
- <candidate_10>
=== Window 2 ===
- <candidate_1>
- <candidate_2>
...
- <candidate_10>
=== Window 3 ===
- <candidate_1>
- <candidate_2>
...
- <candidate_10>

PRIMARY:
<rank1_concat>
```

Table 14: Prompt template used for beam-aware LLM correction. The actual prompt is wrapped in the Qwen chat format. The in-context examples are simplified single-window cases, while the main evaluated setting uses three-window chunks with top-10 candidates per window.