Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

CTC Drafts, Transformer Verifies: Fast and Exact Khmer Text Recognition across Print, Scene and Handwriting

Marry Kong
Techo Startup Center
Rina Buoy
rina.buoy@tsc.gov.kh

Techo Startup Center
Sovisal Chenda
Techo Startup Center
Kimchheang Chhea
Techo Startup Center
Nguonly Taing
Techo Startup Center
Masakazu Iwamura
Osaka Metropolitan University
Koichi Kise
Osaka Metropolitan University

Research Article

Keywords: Khmer text recognition, textline recognition, autoregressive decoding, non-autoregressive decoding, speculative decoding

Posted Date: November 26th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11737743/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: Competing interest reported. Koichi Kise is an Editor-in-Chief of the International Journal on Document Analysis and Recognition (IJDAR).

CTC Drafts, Transformer Verifies: Fast and Exact Khmer Text Recognition across Print, Scene and Handwriting

Marry Kong[1†], Rina Buoy[1,2*†], Sovisal Chenda[1], Kimchheang Chhea[1], Nguonly Taing[1], Masakazu Iwamura[2], Koichi Kise[2]

[1]Techo Startup Center, Ministry of Economy and Finance, Phnom Penh, Cambodia.
[2]Osaka Metropolitan University, Osaka, Japan.

*Corresponding author(s). E-mail(s): rina.buoy@techostartup.center;
†These authors contributed equally to this work.

Abstract

Text recognition models generally follow one of two generation schemes: non-autoregressive (NAR) or autoregressive (AR). NAR methods (e.g., those with a connectionist temporal classification (CTC) decoder) produce all tokens in a single pass but tend to be less accurate. AR methods (e.g., those with a Transformer decoder) generate one token at a time and tend to be more accurate. Although both decoders can coexist within a single model, the choice between them must be fixed before inference, which forces a trade-off between latency and accuracy. Speculative decoding avoids this trade-off. Specifically, a weaker draft model proposes several tokens at once, and a stronger target model verifies them in one parallel pass, keeping the longest prefix it agrees with and correcting the first mismatch. The output is therefore identical to the target model's, but fewer sequential passes are needed. For Khmer text recognition, we propose a novel lossless speculative decoding scheme that pairs an NAR decoder as the draft model with an AR decoder as the target model, along with an improved architecture based on a hybrid vision Transformer (ViT) backbone. The proposed model using the proposed decoding strategy preserves the accuracy of AR decoding exactly while substantially reducing the number of decoding steps compared to the standard AR decoding. The models are trained in two phases: large-scale pretraining on document text, followed by fine-tuning to adapt to handwritten, scene, and document text images. Experiments on handwritten, scene, and printed textline datasets show that the proposed model with the proposed speculative decoding achieves the best overall performance against the re-implemented baselines and existing Khmer text recognizers. Moreover, the proposed decoding scheme matches the AR output exactly while lowering latency by **6.09**× (without KV caches) and the number of sequential decoder passes by **12.3**×. The project repository, including codes, models, and datasets, is available on GitHub.*

Keywords: Khmer text recognition, textline recognition, autoregressive decoding, non-autoregressive decoding, speculative decoding

1 Introduction

Although vision-language models (VLMs) perform remarkably well on many document analysis and recognition tasks for Latin-script languages, they frequently hallucinate on low-resource languages such as Khmer [1]. More broadly, text

recognition remains difficult for complex modalities such as scene and handwritten text, both in accuracy and in practical deployment under resource constraints. These challenges are compounded for Khmer, a low-resource language with a highly intricate script and writing system, where scene and handwritten text recognition remains unsolved despite recent advances in methods and datasets [2, 3].

Like approaches for Latin-based scripts, Khmer text recognition methods can be broadly categorized as either non-autoregressive (NAR) or autoregressive (AR), according to how characters or tokens are generated [4]. An NAR decoder, such as a connectionist temporal classification (CTC) decoder [5], decodes tokens in parallel and therefore does not capture token dependencies. As a result, an NAR decoder is not language-aware (i.e., it lacks an internal language model) [6]. In contrast, an AR decoder, such as a Transformer decoder, decodes one token at a time, conditioning on the previously decoded tokens. Consequently, an AR decoder learns an internal language model, which helps resolve ambiguous cases [7].

Although both NAR and AR decoders can be combined in a unified UKTR architecture [2], only one decoding mode can be chosen at inference time. It is not possible to route individual input images to different decoders to balance latency and accuracy. Instead, the choice is a binary trade-off between latency and accuracy. Thus, even though the two decoders are jointly optimized during training, their complementary information is not exploited at inference to balance latency and accuracy. This raises a question: can latency be reduced without any loss of accuracy by decreasing the number of decoding steps?

Inspired by speculative decoding [8], which accelerates token generation during large language model (LLM) inference via a draft-and-verify technique, this paper proposes a novel lossless speculative decoding scheme that leverages joint NAR and AR decoders to dynamically balance latency and accuracy. Specifically, the proposed speculative decoding strategy matches the accuracy of greedy AR decoding while needing only a small fraction of its decoding steps. In addition, we propose an enhanced architecture built on a hybrid vision Transformer (ViT) [9] backbone for Khmer text recognition. We experiment with three ViT variants of increasing complexity: Tiny (ViT-T), Small (ViT-S), and Base (ViT-B). The models are trained in two phases. Specifically, the models undergo a pretraining phase on large-scale document text images, followed by a fine-tuning phase to adapt to real scene and handwritten text images. We evaluate the models on real benchmark datasets covering different text modalities, including printed, scene, and handwritten images. The experimental results show that our proposed models with the lossless speculative decoding strategy achieve the best overall performance by outperforming or performing on par with the re-implemented baselines and existing Khmer text recognizers. In addition, the proposed decoding scheme matches the AR output exactly while lowering latency by 6.09× (without KV caches) and the number of sequential decoder passes by 12.3×.

Our key contributions can be summarized as follows:

- We propose a novel lossless speculative decoding scheme that leverages joint NAR and AR decoders to dynamically balance latency and accuracy. Specifically, it matches the accuracy of greedy AR decoding while needing only a small fraction of its decoding steps.
- We propose an enhanced architecture built on a hybrid vision Transformer (ViT) backbone for accurate Khmer text recognition.
- Our proposed models with the proposed speculative decoding strategy achieve the lowest overall character error rate (CER) among the compared systems and a substantially lower latency than the AR decoding on real evaluation datasets.

2 Literature Review

2.1 Non-autoregressive (NAR) Recognizers

Existing methods [10–14] for Latin scene text recognition employ a CTC decoder [5], which maps a one-dimensional (1D) feature sequence along the width direction to output tokens in a single pass or step. This is enabled by a blank token (ϵ) that is collapsed at inference time. Early CTC-based methods paired a convolutional feature extractor with a recurrent [10, 12–14] or

self-attention [11] network to model feature dependencies, whereas ViT-based methods [15–17] can predict text tokens directly from visual tokens alone. What all of these NAR methods share is that they emit every token at once, with tokens conditionally independent given the input image. As a result, they lack an internal language model, which is crucial when the image is ambiguous or partially occluded and visual cues alone cannot determine the correct characters.

2.2 Autoregressive (AR) Recognizers

Attention-based encoder-decoder methods [18–24] relax the conditional independence assumption by conditioning each token prediction on the tokens generated before it. This conditioning lets the model learn an internal language model, which helps with complex and ambiguous text images. With the rise of Transformers [25], the attention-based decoder was later superseded by a Transformer decoder [7, 26–30], which uses cross-attention to attend to visual features and self-attention to model language context. However, because tokens are produced one at a time, an AR decoder is slower than an NAR decoder built on the same encoder. Cross-attention can also drift on long textline images [22].

2.3 Joint NAR and AR Recognizers

Methods such as [7, 29, 31] combine both decoders in a single model, allowing NAR or AR decoding on demand. Since AR decoding sees only left-to-right context and NAR decoding sees none, additional iterative refinement steps can bring in context and improve language understanding. Even so, users must still pick one decoder at inference time, trading accuracy for speed or vice versa.

2.4 Accelerated Decoding

Speculative decoding [8] speeds up AR generation without altering its output. Specifically, a much smaller model proposes draft tokens, and the target larger model verifies them all in one parallel pass. Subsequent variants attach draft prediction heads to the target model (Medusa [32]) or train the drafter with a CTC objective (CTC-drafter [33]). To the best of our knowledge, however, no prior text recognition work uses the CTC decoder of a jointly trained recognizer to draft for its own attention decoder while exactly preserving the greedy AR output. Our scheme differs from existing approaches in two respects. Firstly, the acceptance is exact, and secondly, upon rejection, we re-draft from the CTC posteriors, constrained to the accepted prefix.

2.5 OCR-Free Models

Although OCR-free models [34–36], such as vision-language models (VLMs), recognize Latin-script text remarkably well, they still hallucinate on handwritten and scene text in low-resource languages, whose scripts are often underrepresented in training data [1]. Moreover, general-purpose VLMs [34, 35] contain billions of parameters, which makes them too costly for resource-constrained deployment. Specialized recognizers therefore remain the practical choice for low-resource scripts such as Khmer.

2.6 Khmer Script and Khmer Text Recognition

Khmer is a low-resource language written in a structurally complex script [3]. Its alphabet is among the largest in the world, comprising 33 consonants, 23 dependent vowels, 14 independent vowels, and 14 diacritics, and words are not separated by spaces. Unlike Latin letters, Khmer consonants can take on different roles within a cluster, which consists of a base consonant combined with up to two subscript consonants, a vowel, and a diacritic. Certain vowels also form ligatures with the base consonant, and the visual order of glyphs often differs from the logical character order. Further details are provided in Appendix 8.1.

Early Khmer optical character recognition (OCR) research methods [37–41] focused on isolated printed and handwritten characters, as well as characters and short words from Khmer palm leaf manuscripts. Subsequent end-to-end textline recognizers, built on attention-based encoder-decoder and Transformer architectures [3, 42], were able to read printed Khmer textlines. To handle long lines without resizing, a chunking and

merging strategy was introduced that avoids the quadratic cost of self-attention [3, 11]. A comparative study of attention-based and CTC-based decoders [41] on a shared visual feature extractor found that the attention-based decoder was more accurate for Khmer text recognition [12].

UKTR [2] recognizes printed, scene, and handwritten Khmer text with a single model. Its modality-aware adaptive feature selection (MAFS) module routes features through a mixture of source-specific adapters, which keeps underrepresented modalities from being overwhelmed during training on imbalanced mixed data. Although UKTR trains a CTC decoder and a Transformer decoder jointly, inference requires selecting one of them via configuration, regardless of the input image. It therefore faces the same speed-accuracy trade-off as the hybrid Latin-script models described above.

This paper extends the Khmer recognizers above in three ways. Firstly, we replace the convolutional Transformer encoder with a hybrid ViT backbone. Secondly, we share MAFS-style mixture-of-adapters routing across both CTC and Transformer decoders. Thirdly, and most significantly, we treat the CTC (NAR) decoder and the Transformer (AR) decoder as collaborators rather than alternatives. Specifically, the CTC decoder drafts, and the Transformer decoder verifies, producing exactly the output of greedy AR decoding with far fewer sequential decoder passes or steps.

3 Proposed Method

Figure 1 provides an overview of the proposed method. Since Khmer lacks explicit word boundaries, a textline image can be arbitrarily long. To avoid the quadratic cost of self-attention on long inputs, we divide each line into fixed-size, overlapping chunks and encode each chunk independently with a shared ViT with register tokens (ViT-S). A seam-aware overlap-add then stitches the chunk-level features into a single line-level feature map, and a mixture of lightweight source adapters (MoA) adapts it to the input text modality (i.e., print, scene, and handwriting). The adapted features feed two jointly trained decoders, a CTC decoder and a Transformer decoder. At inference time, either decoder can be used alone, or both can be combined in our CTC-drafted speculative decoding, which reproduces the output of greedy AR decoding with far fewer sequential decoder passes or steps.

3.1 Visual Encoder

First, we split each textline image into fixed-size, overlapping image chunks (see Appendix 8.5 for details). Next, we encode each image chunk with a ViT-S backbone [9] that incorporates a convolutional stem [43], LayerScale [44], and register tokens [45]. The backbone is trained from scratch.

Each image chunk first passes through two 3×3 convolutions ($3 \to 48 \to 96$ channels, strides 2 and 1), each followed by batch normalization and ReLU, which yields a 16×64 feature map. A 2×2 patch embedding with stride 2 then converts this map into $8 \times 32 = 256$ tokens of dimension $d = 384$, and a learned absolute position embedding is added. Overall, the encoder downsamples by a factor of 4 in both directions, retaining the granular spatial details required for densely stacked Khmer characters.

Four learnable register tokens $\mathbf{r}_1, \ldots, \mathbf{r}_4$, which receive no position embedding, are prepended to the patch tokens and give the model a place to store global information. Together, the 260 tokens pass through 12 pre-norm Transformer blocks, each consisting of 6-head self-attention, an MLP with expansion ratio 4 and GELU, and LayerScale initialized at 10^{-5}. After the final LayerNorm, the register tokens are dropped, and the patch tokens are reshaped into a chunk feature map $\mathbf{f}_n \in \mathbb{R}^{8\times32\times384}$. We obtain a line-level feature map $\mathbf{F}$ by stitching the chunk-level features with a seam-aware overlap-add method (see Appendix 8.6 for details).

3.2 Mixture of Adapters (MoA)

The training data mix printed document, scene, and handwritten textline images in considerably different proportions. Instead of training a separate model for each source, we insert $K = 5$ bottleneck adapters combined with a soft, line-level router (Fig. 1, bottom). Each adapter is a residual bottleneck

$$A_k(\mathbf{X}) = \mathrm{LN}\big(\mathbf{X} + \mathbf{W}_k^{\uparrow}\,\mathrm{GELU}(\mathbf{W}_k^{\downarrow}\mathbf{X})\big), \qquad (1)$$

with $\mathbf{W}_k^{\downarrow} \in \mathbb{R}^{128\times384}$ and $\mathbf{W}_k^{\uparrow} \in \mathbb{R}^{384\times128}$. The router reads a mean of the input sequence $\mathbf{X}$ that

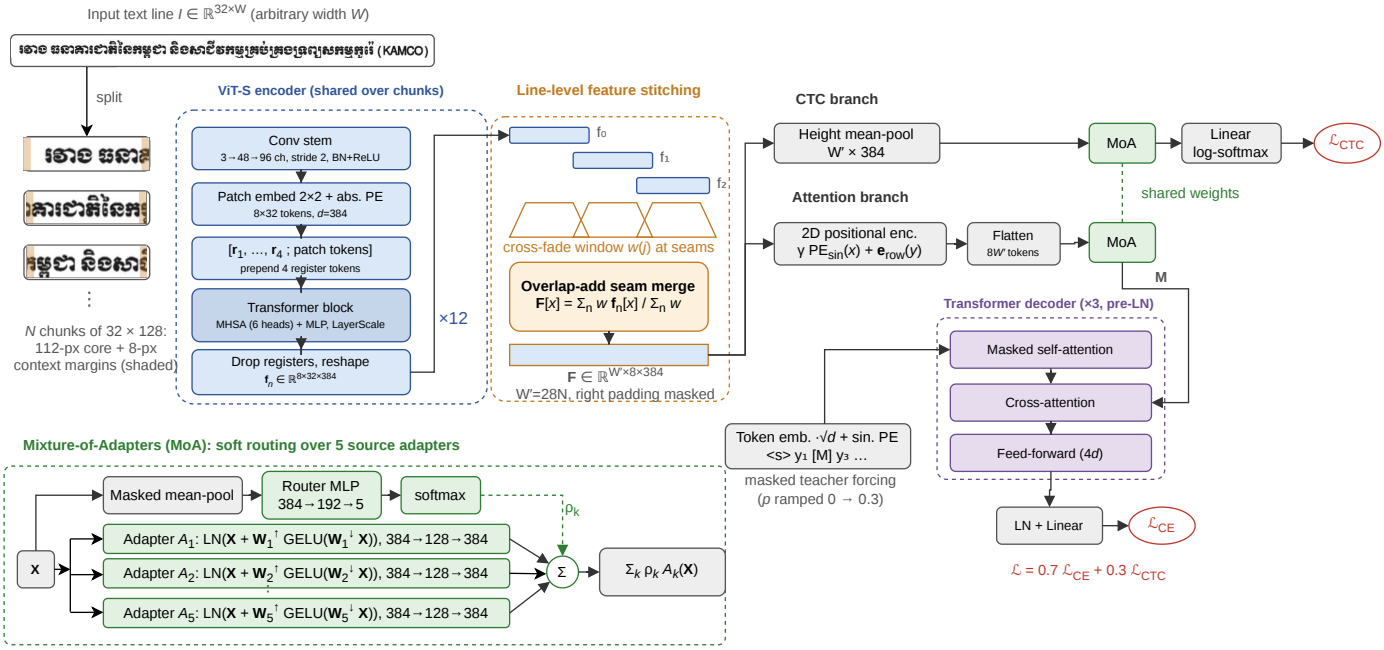

Fig. 1: Overview of the proposed architecture. A textline is divided into overlapping 32×128 chunks, each encoded by a shared ViT-S backbone. Overlap-add with a cross-fade window stitches the chunk features into a line-level feature map $\mathbf{F}$, which is routed through a mixture of source adapters (MoA, bottom) into a CTC branch and an attention branch. The attention branch supplies the memory for a 3-layer Transformer decoder. Both branches are trained with $\mathcal{L} = 0.7\,\mathcal{L}_{\text{CE}} + 0.3\,\mathcal{L}_{\text{CTC}}$.

is masked by $\mathbf{v}$, so padding does not dilute the summary, and outputs mixture weights through a two-layer MLP ($384 \to 192 \to 5$, ReLU):

$$\boldsymbol{\rho} = \text{softmax}\Big(\text{MLP}\Big(\frac{\sum_i v_i \mathbf{x}_i}{\sum_i v_i}\Big)\Big), \qquad \text{MoA}(\mathbf{X}) = \textstyle\sum_{k=1}^{K} \rho_k\, A_k(\mathbf{X}). \tag{2}$$

No source labels are used. Instead, the router is trained end-to-end through the joint recognition losses alone, and a single MoA module with shared weights is applied separately to the inputs of each decoder, as detailed in the following sections. Following [2], $K = 5$ is used to capture not only the principal text modalities (print, scene, and handwriting) but also to give a modeling flexibility in handling mixed-modality cases.

3.3 CTC As An NAR Decoder

The CTC decoder maps a 1D feature sequence along the width direction to the target tokens by marginalizing over all possible alignments that include a blank token (ϵ). For this purpose, the encoded 2D feature map is averaged over its height and adapted, producing one frame per feature column. A linear layer and a log-softmax produce

$$\mathbf{P} = \log \text{softmax}\big(\text{MoA}(\bar{\mathbf{F}})\,\mathbf{W}_{\text{ctc}}\big) \in \mathbb{R}^{T\times|\mathcal{V}|}, \tag{3}$$

where $\bar{\mathbf{F}} = \frac{1}{8}\sum_y \mathbf{F}[:, y]$, $T = W'$, and $\mathcal{V}$ is the vocabulary.

3.4 Transformer As An AR Decoder

The decoder takes a flattened visual feature sequence ($\mathbf{M}$) as input, so two-dimensional (2D) position information is added before flattening:

$$\tilde{\mathbf{F}}[x, y] = \mathbf{F}[x, y] + \gamma\,\text{PE}_{\text{sin}}(x) + \mathbf{e}_{\text{row}}(y), \tag{4}$$

where $\text{PE}_{\text{sin}}(x)$ is the sinusoidal encoding of the global column index, which is continuous across chunk boundaries and computed on the fly, so the line length is unbounded. Here, γ is a learnable scale initialized to 1, and $\mathbf{e}_{\text{row}} \in \mathbb{R}^{8\times384}$ is a learned row embedding. The flattened visual feature sequence (the memory) is $\mathbf{M} =$

$\mathrm{MoA}(\mathrm{flatten}(\tilde{\mathbf{F}})) \in \mathbb{R}^{8W' \times 384}$, with its elements ordered column-major.

The AR decoder is a 3-layer pre-LN Transformer decoder [25, 46] with $d = 384$, 6 attention heads, and a feed-forward width of 1536 with ReLU activation. In each layer, masked self-attention, cross-attention to $\mathbf{M}$, and the feed-forward block are each preceded by LayerNorm and wrapped in a residual connection. A final LayerNorm and a linear layer project the output onto the vocabulary ($|\mathcal{V}|$). Token embeddings are scaled by $\sqrt{d}$ and drawn from $\mathcal{N}(0, d^{-1})$, which gives them unit variance so that they are not dominated by the sinusoidal position encoding.

The ground-truth texts are tokenized with a Khmer-specific subword tokenization scheme based on Khmer character clusters (KCC) concept (see Appendix 8.2 for details), with a vocabulary of $|\mathcal{V}| = 11{,}899$ entries, including the special tokens `<s>` (start of sentence), `</s>` (end of sentence), `<pad>` (padded token), `<mask>` (masked token), and `<sp>` (for the word space). The padding index doubles as the CTC blank ϵ.

3.5 Training Objective

The joint decoders share the same visual encoder and are optimized jointly minimizing a weighted loss:

$$\mathcal{L} = (1 - \lambda)\,\mathcal{L}_{\mathrm{CE}} + \lambda\,\mathcal{L}_{\mathrm{CTC}}, \qquad \lambda = 0.3, \tag{5}$$

where $\mathcal{L}_{\mathrm{CE}}$ is the cross-entropy of the Transformer decoder and $\mathcal{L}_{\mathrm{CTC}}$ is the CTC decoder's loss over $\mathbf{P}$, computed on ground-truth texts. The CTC decoder's loss encourages monotonic alignment between encoder frames and output tokens, which helps the decoder's cross-attention. We set $\lambda = 0.3$ to balance the convergence rates of the two decoders, because the CTC decoder tends to converge faster.

Standard teacher-forced cross-entropy training lets the decoder rely heavily on the preceding ground-truth tokens, which can pull it away from the relevant visual features. This behavior is commonly referred to as attention drift. To counter this, we replace each decoder input token with `<mask>` with probability $p_s = 0.3 \cdot \min(1, s/15{,}000)$ at training step s. This linear ramp preserves full teacher forcing at the start, when the decoder is still learning alignment, and gradually forces it to predict from relevant visual features at roughly 30% of the positions.

To expose the model to long textline images, we concatenate each sample in the second half of a mini-batch, with probability 0.5, horizontally with a randomly chosen earlier sample from the same batch, placing it on either the left or the right at random. The ground-truth texts are concatenated accordingly.

3.6 Proposed CTC-Drafted Speculative Decoding

Let $\mathbf{P}$ be the CTC log-probabilities and $\mathbf{M}$ the flattened visual feature sequence, both derived from the visual encoder described earlier. We consider three decoding schemes (Figs. 2–4).

For the CTC decoder, the greedy path, which takes the highest-probability token at each frame, $\pi_t = \arg\max_c P_t(c)$, is collapsed by merging repeated tokens and removing blanks (ϵ) to give $\hat{\mathbf{y}} = \mathcal{B}(\pi)$ (Fig. 2). Because decoding is a single parallel pass over conditionally independent frames, the output has no language awareness unless an external language model is added.

For the Transformer decoder, we start from $\mathbf{g} = (\texttt{<s>})$ and repeatedly run the decoder, appending the greedy token $y_i = \arg\max_c p_\theta(c \mid \mathbf{y}_{<i}, \mathbf{M})$ at each step until `</s>` is generated (Fig. 3). Decoding a textline image of L tokens thus takes $L + 1$ sequential passes, so the AR decoding is the most accurate option but also the slowest [12].

Speculative decoding [8] pairs a much smaller draft model, which proposes several tokens, with a larger target model, which verifies all draft tokens in one forward decoding pass. In our proposed approach, the drafter requires no extra model. Specifically, the CTC decoder is trained jointly on the same encoder, so its log-probabilities $\mathbf{P}$ are available at no additional cost. The procedure (Fig. 4, Algorithm 1) maintains a confirmed prefix $\mathbf{g}$ and repeats four steps:

1. **Draft.** Obtain a draft continuation $\mathbf{d} = (d_1, \ldots, d_k, \texttt{</s>})$ from CTC. In the first iteration $\mathbf{g}$ is empty and this is plain CTC greedy decoding.
2. **Verify.** Run the decoder once on $[\texttt{<s>}; \mathbf{g}; \mathbf{d}]$. The output token at each position equals the greedy AR prediction for that position given

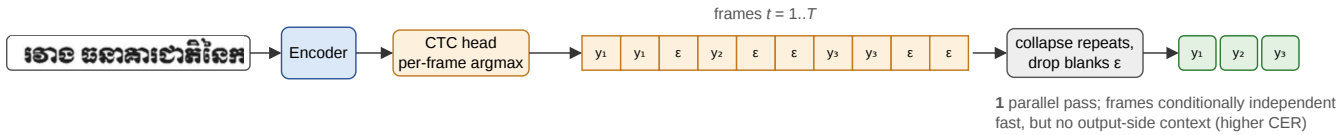

Fig. 2: CTC greedy decoding: one parallel pass, argmax per frame, then collapse of repeats and blanks.

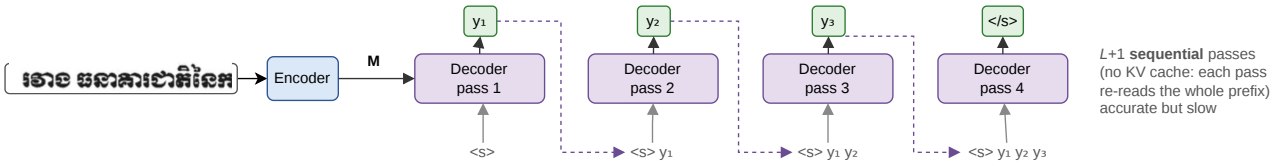

Fig. 3: AR greedy decoding: $L{+}1$ sequential decoder passes, each conditioned on all previously generated tokens.

the previous tokens, so this single pass yields the greedy choices $p_1, \ldots, p_{k+2}$ for all draft slots plus one extra output token.

3. **Accept.** Let n be the length of the longest prefix of the draft that the decoder agrees with ($p_i = d_i$ for $i \le n$). Append the n accepted tokens and the decoder's own token at the first mismatch, $\mathbf{g} \leftarrow \mathbf{g} \oplus d_{1:n} \oplus p_{n+1}$. Every pass therefore advances by at least one token, and by $n+1$ tokens in general.
4. **Re-draft.** If `</s>` has not been accepted, compute a new draft with prefix-constrained CTC decoding (see Appendix 8.4 for details) and return to step 2.

Because every accepted token is exactly the token greedy AR would have produced at that position, the output is identical to the greedy AR decoding, while the number of decoder passes drops from $L+1$ to the number of draft-verify rounds.

Algorithm 1 CTC-drafted speculative decoding

Require: CTC log-probs $\mathbf{P} \in \mathbb{R}^{T \times |\mathcal{V}|}$, memory $\mathbf{M}$, max length $L_{\max}$

$\mathbf{g} \leftarrow ()$; $\mathbf{d} \leftarrow \textsc{CtcPrefixDraft}(\mathbf{P}, \mathbf{g})$
while $|\mathbf{g}| < L_{\max}$ **do**
  $\mathbf{p} \leftarrow \arg\max \textsc{Decoder}([\texttt{<s>}; \mathbf{g}; \mathbf{d}], \mathbf{M})_{|\mathbf{g}|+1:}$ ▷ one pass
  $n \leftarrow \max\{i : p_{1:i} = d_{1:i}\}$
  $\mathbf{g} \leftarrow \mathbf{g} \oplus d_{1:n} \oplus p_{n+1}$ ▷ truncated after `</s>`
  if `</s>` $\in \mathbf{g}$ **then break**
  end if
  $\mathbf{d} \leftarrow \textsc{CtcPrefixDraft}(\mathbf{P}, \mathbf{g})$ ▷ Viterbi align $\mathbf{g}$, argmax tail
end while
return $\mathbf{g}$ without `</s>`

4 Data & Experiments

4.1 Model Complexity

Table 1 summarizes the parameter counts and per-chunk FLOPs of the main modules in the default configuration, which uses the ViT-S (Small) encoder. The model has 43.0M trainable parameters, 62% of which are in the encoder, and the adapters and router add only 0.57M (1.3%). Roughly 13.7M parameters sit in the three unshared $|\mathcal{V}|$-sized matrices, which are the CTC classifier, the decoder embedding, and the decoder output projection. Besides the default ViT-S encoder, we evaluate two alternatives, ViT-T (Tiny, $d = 192$) and ViT-B (Base, $d = 512$), which share the same layout. Table 2 compares the three variants. In all three variants, the backbone accounts for over 90% of the FLOPs, and scaling from ViT-S to ViT-B raises the per-chunk cost by 1.7× (from 6.70 to 11.52 GFLOPs).

4.2 Datasets : Training & Evaluation

We train and evaluate our models on the following Khmer text recognition datasets:

- **Buoy et al. [3]'s Synthetic Document and Scene Textline Data (SynDoc)**: The authors synthesized 2.8 million textline images (1.5 million document and 1.3 million scene images) using 11 commonly used Khmer fonts.

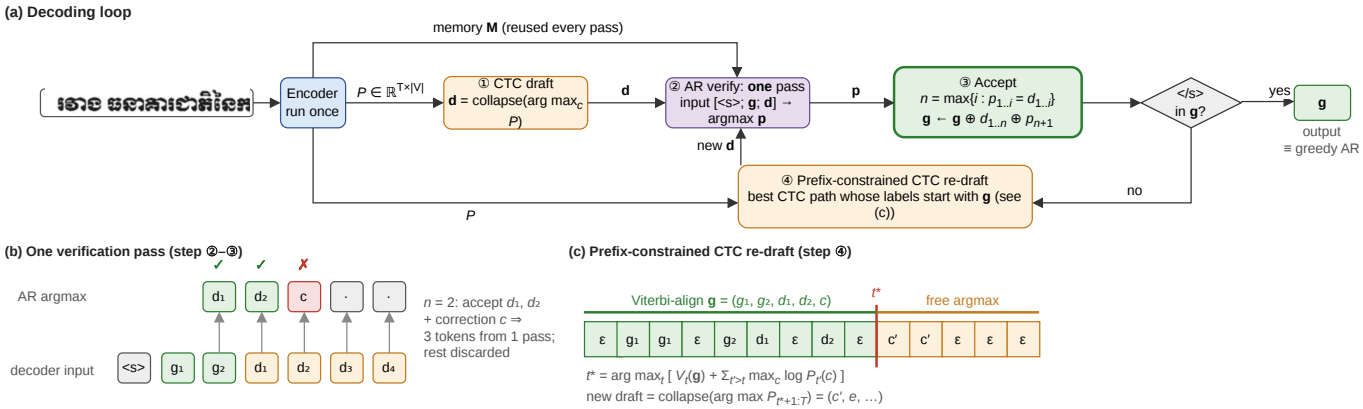

Fig. 4: Proposed CTC-drafted speculative decoding. (a) The encoder runs once. The CTC decoder generates draft tokens, and the Transformer (AR) decoder verifies. The longest agreeing prefix tokens plus one corrected token are accepted, and the CTC decoder re-drafts from the confirmed prefix tokens until `</s>`. (b) A verification pass in which d_1, d_2 are accepted and d_3 is corrected to c, giving three tokens from one AR decoding step. (c) Prefix-constrained re-drafting: the confirmed prefix tokens are Viterbi-aligned to frames $1..t^*$, and the per-frame argmax of the remaining frames forms the new draft tokens.

Table 1: Parameters and FLOPs of the main modules for one 32×128-pixel chunk.

Module	Params (M)	GFLOPs
ViT-S backbone	21.59	6.238
Stitching, 2D PE and MoA	0.58	0.126
CTC head	4.58	0.128
Encoder + CTC head	**26.75**	**6.492**
Token emd. + output proj.	9.15	0.005
Decoder layers (3×)	7.10	0.205
Decoder (one pass)	**16.25**	**0.209**
Total	**43.00**	**6.702**

- **KHOB** [47]: KHOB is a manually curated dataset [48] of 1,318 textline image samples drawn from scanned, low-resolution Khmer PDF documents.
- **SynthText**[1]: This dataset contains 10,000 synthetically generated Khmer identity card images, yielding approximately 70,000 cropped textline images.
- **Long et al. [49]'s HierText Data**: The dataset is split into training (8,281 images), validation (1,724 images), and test (1,634 images) sets, with annotations for Latin printed and handwritten textlines and words. Once the images are cropped into textlines and vertical text is removed from the training and validation sets, approximately 518,726 images remain.
- **Nom et al. [50]'s KhmerST**: This dataset consists of real Khmer scene text images, mostly logos and billboards, captured indoors and outdoors in natural settings. It features both short words and long textlines. The original 1,544 manually annotated images yielded 3,237 cropped text images, and the processed KhmerST version [48] retains 3,022 of them. The dataset is particularly challenging because of its complex imaging conditions and diverse artistic fonts.
- **Nom et al. [51]'s WildKhmerST**: This dataset is an extension of KhmerST, containing 29,601 annotated textlines from 10,000 images captured at diverse public locations in Cambodia, such as streets, signboards, supermarkets, and commercial areas. It includes many difficult samples, among them artistic, blurred, low-light, curved, and occluded text, as well as text on complex backgrounds.
- **Soy**[2]**'s Khmer Handwritten Data (KH)**: This dataset contains approximately 4.2K handwritten and printed textline images in

[1] Huggingface: seanghay/SynthKhmer-10k

[2] Huggingface: SoyVitou/khmer-handwritten-dataset-4.2k

Table 2: Parameters and FLOPs of the main modules for ViT-T, ViT-S, and ViT-B encoders, measured as in Table 1 (one 32×128-pixel chunk).

	Tiny		Small (ours)		Base	
Width d / heads / stem channels	192 / 3 / 64		384 / 6 / 96		512 / 8 / 128	
Module	Params (M)	GFLOPs	Params (M)	GFLOPs	Params (M)	GFLOPs
ViT backbone	5.46	1.730	21.59	6.238	38.31	10.807
Stitching, PE and MoA	0.27	0.063	0.58	0.126	0.80	0.169
CTC head	2.30	0.064	4.58	0.128	6.10	0.171
Encoder + CTC head	**8.03**	**1.858**	**26.75**	**6.492**	**45.22**	**11.146**
Token emb. + output proj.	4.58	0.002	9.15	0.005	12.20	0.006
Decoder layers (3×)	1.78	0.051	7.10	0.205	12.61	0.364
Decoder (one pass)	**6.36**	**0.054**	**16.25**	**0.209**	**24.81**	**0.370**
Total	**14.39**	**1.911**	**43.00**	**6.702**	**70.03**	**11.516**

Khmer and Latin scripts, divided into 3,991 training images and 211 evaluation images.

- **Kong et al. [2]'s General Khmer Scene Text Data (GKST)**: This dataset contains 4,221 manually annotated Khmer scene text images. Like KhmerST and WildKhmerST, its images were taken with smartphones in general indoor and outdoor conditions and exhibit natural distortions and perspective transformations. The data are split into 4,009 training images and 212 evaluation images.
- **Kong et al. [2]'s General Khmer Handwritten Text Data (KHT)**: This dataset contains 14,168 manually annotated images of Khmer handwriting from diverse sources, such as birth certificates, exam papers, and personal notes. It is divided into 13,457 training images and 711 evaluation images.
- **Mixed Khmer-English Textline Data (MixKhEn)**[3]: This recently released Huggingface dataset consists of 1,491,770 printed textline images, divided into 1,342,589 training images and 149,181 validation images. Its textlines are in Khmer, Latin, or mixed Khmer-English and are rendered on varied backgrounds with degradations such as blur, low contrast, and noise. We use only the training set.

We organize the datasets described above into three groups according to their primary modalities:

- **Document (D)**: With the exception of the Latin-script HierText dataset, the datasets in this group are mostly synthetic and large-scale: SynDoc, SynthText, HierText, and the training set of MixKhEn. The goal of this group is to learn visual representations of Khmer and Latin scripts, including textlines that mix the two scripts, from large-scale printed text.
- **Scene & Handwritten (S&H)**: This group consists of real-world data, primarily scene text and handwriting: WildKhmerST and the training sets of KH, GKST, and KHT.
- **Evaluation**: To remain consistent with prior work, the evaluation group consists of KhmerST (S), KHOB (D), and the evaluation sets of KH (D&H), GKST (S), and KHT (H).

4.3 Training Configuration and Setup

Training consists of two phases (Table 3). Phase 1 learns Khmer and Latin visual representations from large-scale printed data, starting from random initialization. Phase 2 then adapts the model to real scene and handwritten text, while continuing to use printed data. Both phases use the same joint objective from Section 3.5 ($0.7\,\mathcal{L}_{\mathrm{CE}} + 0.3\,\mathcal{L}_{\mathrm{CTC}}$) with masked teacher forcing, the same tokenizer ($|\mathcal{V}| = 11{,}899$), and the same chunking method.

[3] Huggingface: LazyGreed/mixedKhENTextLine

4.3.1 Phase 1: Pre-training on Printed Data

The training and validation data are drawn from the Document (D) group, with all textlines pooled and shuffled. The last 10,000 samples are held out for validation and the rest used for training. We train for 5 epochs using mini-batches of 32 textlines and the Adam optimizer, with a peak learning rate of 10^{-4}, a 2,000-step linear warm-up, and cosine decay to 10^{-5}. Gradient norms are clipped at 5. Augmentations are applied on the fly: random perspective warp, horizontal motion blur, dilation or erosion, ink blobs, elastic noise distortion, paper and ink textures blended into the foreground and background, Gaussian blur, and intensity inversion. Textline concatenation (Section 3.5) is applied to all data sources.

4.3.2 Phase 2: Fine-tuning on Scene and Handwritten Data

Phase 2 is initialized from the best Phase 1 checkpoint. Because fine-tuning only on the small real datasets risks forgetting the printed domain, each training step combines two mini-batches of 16 textlines, one from the Scene & Handwritten (S&H) group and one from the D group, both processed exactly as in Phase 1, and the two batch losses are summed before each update. Each S&H textline receives one randomly chosen augmentation which can be additive Gaussian noise, an 8×8 box blur, a 4×4 dilation, or a horizontal stretch by a factor drawn from $[0.5, 1.5]$. Latin characters in the S&H ground-truth texts are lowercased. The learning rate is held at 10^{-5} for the first 2,000 steps and then follows a triangular cyclic schedule between 10^{-6} and 10^{-5} with a half-cycle of 10,000 steps. Gradients are clipped to a norm of 5, and the masked teacher forcing ramp restarts from 0. One epoch is one pass over the S&H training set, and training runs for up to 50 epochs.

4.4 Baseline Models

Because few Khmer text recognition models are available for benchmarking, and to isolate the effect of the decoder, we build five baselines that are identical to the proposed model except for the decoder and its training loss. All baseline models follow the same training recipe as that of our models. Table 4 summarizes the differences.

- **CTC Decoder Only:** In this baseline model, only the CTC decoder of the proposed model is used. A linear classifier predicts a token distribution for each feature column, producing $W' = 28N$ frames per textline. We train with the CTC loss and decode with best-path decoding, which takes the most likely token at each frame, merges repeats, and removes blanks. The CTC decoder needs only one parallel pass but cannot model dependencies between output tokens.
- **Attention-Based Decoder:** This baseline adopts the attention-based prediction module of ASTER [18], following the standardized implementation of Baek et al. [12]. At each step t, the decoder attends to the visual features and computes attention weights from its previous hidden state, forms a context vector $\mathbf{c}_t$, updates its state $\mathbf{c}_t]$, and predicts y_t .
- **PARSeq Decoder:** PARSeq [7] employs a single two-stream decoder layer, in which learned position queries attend both to a content stream of previous tokens and to the visual features, and it is trained with permutation language modeling. The pre-LN decoder has 6 attention heads, a feed-forward size of 1,536, and position queries for up to 200 tokens plus `</s>`. Each batch uses $K = 6$ factorization orders: left-to-right, right-to-left, and two random orders along with their mirrors.
- **ViTSTR:** ViTSTR [15] predicts the first N output tokens of a ViT in parallel, one character per token, which requires the encoder to see the entire word. Because our backbone processes only one chunk at a time, we recover line-level context by adding 200 learnable slot tokens, which are prepended to the height-pooled feature sequence and passed through 3 pre-LN Transformer layers (6 attention heads, feed-forward size 1,536). Every slot is classified independently. The targets are the transcription followed by `</s>`, padded to 200 slots, and the loss is the cross-entropy over the non-padding slots. At inference, a single parallel pass is made, and the output is truncated at the first `</s>`.
- **ViTSTR-Transducer.** ViTSTR-Transducer [16] extends ViTSTR with an autoregressive language model that does not use cross-attention. Its visual features $\mathbf{F}_u$ are the $T = 200$ slot outputs of the ViTSTR textline encoder described above. A 3-layer

Table 3: Training phases. Both phases use $\mathcal{L} = 0.7\,\mathcal{L}_{\text{CE}} + 0.3\,\mathcal{L}_{\text{CTC}}$ and masked teacher forcing (p ramped $0 \to 0.3$ over 15k steps).

	Phase 1	Phase 2
Initialization	random	best phase-1 checkpoint
Training data	D	S&H + D
Batch	32 (D)	16 (S&H) + 16 (D)
Epochs	5 over D	up to 50 over S&H
Optimizer	Adam	Adam
Learning rate	warm-up 2k, cosine $10^{-4} \to 10^{-5}$	2k const., cyclic 10^{-6}–10^{-5} (half-cycle 10k)
Gradient clip	5	5
Augmentation	document degradations	noise / blur / dilation / stretch (S&H)

Table 4: Re-implemented Baseline Models. "1D": the height-pooled $\mathbf{F}$. "2D": 2D features, $\mathbf{F}$. AR: autoregressive, greedy; TF: teacher forcing. Params: including the backbone.

Model	Head	Head input	Training loss	Decoding	Params (M)
CTC	linear classifier	1D	CTC	NAR	26.75
Attention	additive attn. + LSTM	1D	CE, masked TF	AR	36.14
PARSeq	1 two-stream layer	2D	perm. LM, K=6	AR	33.77
ViTSTR	3 layers, 200 slots	1D	CE	NAR	32.15
ViTSTR-Trans.	slots + 3-layer ARLM	1D	CE	AR	42.19
Ours	3-layer decoder + CTC	1D + 2D	$0.7\,\text{CE} + 0.3\,\text{CTC}$	Speculative	43.00

causal Transformer (8 attention heads, feed-forward size 1,536) encodes the previous tokens into $\mathbf{G}_u$ without access to the visual features, and the two representations are combined by a learnable gate:

$$\boldsymbol{\alpha}_u = \sigma\big(\mathbf{W}_g(\mathbf{F}_u \odot \mathbf{G}_u)\big), \tag{6}$$

$$\hat{y}_u = \text{softmax}\big(\mathbf{W}_c\big[(1 - \boldsymbol{\alpha}_u) \odot \mathbf{F}_u + \boldsymbol{\alpha}_u \odot \mathbf{G}_u\big]\big), \tag{7}$$

Thus, output u depends only on slot u and the previous tokens. The model is trained with cross-entropy and plain teacher forcing, and decoding is greedy and autoregressive for at most T steps.

4.5 Evaluation

Before computing the CER, we apply the same normalization to both the ground-truth text and the predicted text. First, Latin and non-Khmer characters are converted to lowercase, since the labels use letter case inconsistently. Second, zero-width spaces (U+200B) are removed. Khmer writers use them as invisible word boundaries, so they appear in some ground-truth texts, and they have no visible glyph in the image. Finally, each Khmer character cluster is rewritten in a canonical character order [3].

5 Results and Discussion

5.1 Effect of Backbone Size: ViT-T, ViT-S, and ViT-B

We train the proposed model with three backbone sizes that differ only in the model dimension, d, of the ViT encoder. These are ViT-T ($d = 192$), ViT-S ($d = 384$), and ViT-B ($d = 512$). All the three variants have 12 layers, and the decoder width matches d. Table 5 reports character error rate (CER) and latency for the three decoding modes (AR, NAR, and Speculative or SPEC).

Under the AR decoding, overall CER decreases as the backbone grows. Specifically, the CER falls from 2.57% (ViT-T) to 2.03% (ViT-S) and 1.72% (ViT-B), relative reductions of 21% and 15%. Moving from ViT-T to ViT-S mainly helps the most varied datasets, namely scene text (the KhmerST dataset, −44%) and handwriting (the

Table 5: Effect of backbone size (ViT-T, ViT-S, and ViT-B) measured on an Apple M2 Max (MPS). Best CER per column in bold, second best underlined. SPEC: Speculative.

Backbone	Decoding	CER (%) KHOB 1,318	KhmerST 3,021	KH 211	GKST 212	KHT 711	Overall 5,473	Time (ms / line)	Passes / line
ViT-T	CTC	2.24	3.65	6.39	4.04	11.44	3.15	17.50	–
	AR	1.99	2.89	3.78	3.56	7.86	2.57	285.75	17.48
	SPEC	1.99	2.89	3.78	3.56	7.86	2.57	39.02 (7.32×)	1.53 (11.4×)
ViT-S	CTC	2.34	2.22	3.89	3.56	6.58	2.54	24.21	–
	AR	1.97	<u>1.62</u>	<u>2.89</u>	2.96	<u>4.79</u>	<u>2.03</u>	301.08	17.34
	SPEC	1.97	<u>1.62</u>	<u>2.89</u>	2.96	<u>4.79</u>	<u>2.03</u>	49.43 (6.09×)	1.41 (12.3×)
ViT-B	CTC	<u>1.96</u>	1.64	3.11	<u>2.43</u>	5.80	2.06	28.66	–
	AR	**1.72**	**1.30**	**2.50**	**2.18**	**3.96**	**1.72**	299.59	17.33
	SPEC	**1.72**	**1.30**	**2.50**	**2.18**	**3.96**	**1.72**	52.30 (5.73×)	1.34 (12.9×)

KHT dataset, −39%), while the KHOB dataset barely changes. Moving from ViT-S to ViT-B improves every evaluation set by 13-26%. The AR decoding beats the CTC decoding on every dataset at every size and model complexity, lowering overall CER by 17–20%, and because the speculative decoding reproduces the AR output exactly, it preserves all of these gains.

Latency depends on the number of sequential decoding passes. At batch size 1, a single decoder pass takes roughly 15–18 ms at any ViT backbone. Thus, the latency for the AR decoding is near 300 ms per textline for all three backbones. The proposed decoding scheme lowers from near 300 to 39–52 ms (per line), a 5.7–7.3× speed-up and near 17.5 to 1.53–1.34 passes (per line), a 11.4–12.9 speed-up. The gain diminishes with model size because the encoder, which still runs in full, takes a larger share of the time (45% for ViT-T, 55% for ViT-B).

With the proposed decoding scheme, the ViT-B model (1.72% CER, 52.3, ms) runs 5.5× faster than the ViT-T model with the AR decoding while making a third fewer errors. Going from ViT-S to ViT-B adds only 6% latency for a 15% lower CER.

In summary, ViT-B is the better choice when memory allows. Under tighter compute budgets, ViT-S strikes a better balance, requiring only 61% of the parameters and 58% of the per-chunk FLOPs of ViT-B. All subsequent analyses use the ViT-S variant (Small, $d = 384$) as the proposed model.

5.2 Comparison with Existing Models and Baselines

Table 6 compares the proposed model using the proposed speculative (SPEC) decoding scheme with the existing Khmer recognizers and with the baseline models described in Section 4.4. These baseline models use the same ViT-S backbone and training recipe as those of our model and differ only in the decoder.

With the AR decoding or the SPEC decoding, the proposed model achieves an overall CER of 2.03%, essentially matching the strongest existing system, Anaksor (SVTRv2-B [6] with a CTC decoder, 2.07%). Anaksor is better on the KHOB dataset (1.48% vs. 1.97%) and the KH dataset (2.39% vs. 2.89%), but its CER on the KHT dataset is 71% higher (8.19% vs. 4.79%). Note that its training data include the KhmerST dataset and the handwritten dataset that may overlap with our KhmerST and KH test textlines. The next strongest system, Netra OCR (8.30%), has 4.1× our overall CER and 9.5× our latency (471.7 vs. 49.4 ms). Tesseract and Surya largely fail on handwriting (81.6% and 107.8% CER on KHT; insertions can push CER above 100%). Compared with the results from UKTR [2], our CER is 11-30% lower on all five evaluation datasets.

Under the AR decoding, our model reduces CER by 19% compared with the CTC baseline model, 20% compared with the attention-based baseline model, 26% compared with ViTSTR-Transducer, and 35% compared with PARSeq. Our model's own CTC decoder (2.54%) performs

Table 6: Comparison with existing Khmer text recognizers and with the baselines of Section 4.4. Best CER per column in bold, second best underlined. SPEC: Speculative.

Model	Decoding	Params (M)	KHOB 1,318	KhmerST 3,021	KH 211	GKST 212	KHT 711	Overall 5,473	ms / line
Existing Khmer OCR systems									
Tesseract 5 [52]	CTC	–	9.05	39.18	67.82	39.44	81.56	22.32	122.86
Surya [53]	AR	–	16.07	41.12	74.87	49.62	107.81	28.81	631.64
Kiri OCR [54]	AR	–	39.54	38.10	16.12	39.34	55.81	39.60	341.99
TrorYong OCR [55]	AR	5.5	42.19	28.76	55.53	29.32	70.69	39.48	71.76
Netra OCR [56]	AR	–	4.79	10.73	27.79	12.23	35.37	8.30	471.70
Anaksor OCR [57]	CTC	25.1	**1.48**	2.34	**2.39**	3.15	8.19	2.07	164.74
Reported in prior work (from [2]; not re-run)									
Buoy et al. [3]	AR	–	3.03	–	–	–	–	–	–
Buoy et al. [4]	NAR	–	2.33	–	–	–	–	–	–
Buoy et al. [48]	AR	–	2.13	7.01	–	–	–	–	–
Nom et al. [50]	AR	–	–	17.00	–	–	–	–	–
Soy et al. [58]	AR	–	–	–	17.00	–	–	–	–
UKTR [2]	CTC	–	2.46	3.02	5.89	4.41	9.52	–	–
	AR	–	2.37	2.19	4.11	3.34	6.10	–	–
Baselines (trained in our pipeline)									
CTC	CTC	26.75	2.15	2.45	3.61	3.15	7.29	2.51	18.39
Attention	AR	36.14	1.99	2.85	3.00	4.35	7.33	2.54	38.51
	NAR		35.40	55.32	89.77	35.44	88.57	44.19	23.66
PARSeq	AR	33.77	2.19	3.83	3.78	5.64	10.90	3.14	54.80
	SPEC		2.19	3.83	3.78	5.64	10.90	3.14	36.05
ViTSTR	NAR	32.15	14.68	6.05	6.73	5.39	11.55	11.76	**18.34**
	NAR		39.95	33.20	32.30	32.31	39.95	37.76	21.02
ViTSTR-Transducer	AR	42.19	2.88	1.83	4.11	3.03	6.45	2.75	72.56
	SPEC		2.88	1.83	4.11	3.03	6.45	2.75	49.03
	CTC		2.34	2.22	3.89	3.56	6.58	2.54	24.21
Ours	AR	43.00	1.97	**1.62**	2.89	**2.96**	**4.79**	**2.03**	301.08
	SPEC		1.97	**1.62**	2.89	**2.96**	**4.79**	**2.03**	49.43

competitively with the standalone CTC baseline model (2.51%), which indicates that joint training does not improve the CTC decoder itself and that the CER reduction gain comes from the Transformer (AR) decoder. ViTSTR is the weakest baseline (11.76%), failing mainly on the KHOB dataset, whose textlines are the longest.

The speculative decoding is effective only when the draft tokens are accurate. The NAR-drafted tokens of PARSeq and ViTSTR-Transducer have CERs of 44.2% and 37.8%, so the speculative decoding accelerates their AR decoding by just 1.52× and 1.48×, respectively, whereas the CTC-drafted tokens of our proposed model (2.54% CER) achieve a 6.09× speed-up. Because our Transformer (AR) decoder is evaluated without a KV cache, it re-reads the whole prefix tokens at each step (about 16 ms per pass). The attention-based decoder baseline model carries its state forward and therefore needs only 38.5 ms per line, less than our proposed decoding scheme (49.4 ms), although its CER is 25% higher.

In summary, the proposed model with speculative decoding delivers state-of-the-art results on Khmer printed, scene, and handwritten text recognition, outperforming or matching existing Khmer recognizers, prior work, and our re-implemented baselines. It reproduces the greedy AR output exactly while using only a fraction of the sequential decoding steps.

5.3 Convolutional Transformer Encoder Versus ViT-S Encoder

Table 7 reports recognition performance and latency obtained by replacing the ViT-S backbone with the convolutional Transformer backbone (i.e., the ResNet-45 + Transformer) of UKTR [2] while keeping other parts unchanged.

Compared with the ResNet backbone, the ViT-S encoder reduces the overall CER by the AR decoding from 2.22% to 2.03% (−9%) while using 21% fewer parameters (43.0M vs. 54.2M). It should be highlighted that our proposed model using the ViT-S encoder is trained from scratch, whereas the ResNet encoder is initialized from ABINet. The ViT-S encoder wins on the KHOB, KhmerST, KH, and KHT datasets, while the ResNet wins on the GKST dataset (2.46% vs. 2.96%, 212 textlines) and under the CTC decoding (2.32% vs. 2.54% overall CER). On the KHOB dataset, the AR CER of the ResNet model, is worse compared with the CTC CER. This is because of attention drift problem in the KHOB dataset comprises low-resolution long textlines. This problem is not seen in our ViT-S model. Finally, our re-implementation of UKTR surpasses the published results on every evaluation dataset with both decoders. This can be attributed to the better training data in this study.

Both models need approximately 1.4 decoder passes per textline, and a single decoder pass takes about 15–16 ms with either backbone, so the latency difference comes from the encoder. The ResNet is 5.1 ms faster under the CTC decoding (19.1 vs. 24.2 ms) and 8.0 ms faster under the proposed speculative decoding (41.4 vs. 49.4 ms).

In summary, the ViT-S encoder outperforms the ResNet encoder, and speculative decoding delivers CTC-to-AR accuracy gains on the evaluation datasets at a fraction of AR's decoding cost.

5.4 Soft versus Hard Adapter Routing

The router in Section 3.2 is soft. Specifically, each textline passes through all $K = 5$ adapters, and their outputs are averaged based on a routing distribution. A hard router would, otherwise, send each line to a single adapter. This encourages the adapters to specialize and requires only one adapter at inference. During training, a Gumbel-softmax sample ($\tau = 1$) selects the adapter, so the forward pass uses one adapter while gradients still flow. Table 8 compares the two variants.

Hard routing helps the Transformer (AR) decoder but not the CTC decoder. With the CTC decoding, it is worse on the KhmerST (2.22% → 2.40%), KH (3.89% → 4.11%) and especially KHT (6.58% → 7.68%), and better only on KHOB and GKST datasets. As a result, the AR decoder corrects more of the CTC errors. Specifically, the AR decoding improves the CTC decoding approximately 23% with hard routing against 20% with soft routing. One explanation is that the CTC decoder classifies each frame independently, so it relies more on the adapted features themselves. A mixture of five adapters may give it smoother and more robust features than a single specialized adapter. The AR decoder attends over the whole textline and conditions on its own output, so it can compensate for a less suitable adapter.

Even though hard routing weakens the CTC decoder, the proposed speculative decoding is unaffected. Specifically, the draft acceptance rate (71.4% vs. 71.3%) and the number of verification decoder passes per textline (1.41 vs. 1.42) are nearly identical, and the proposed speculative decoding reproduces the AR decoding on every textline with both routers. Thus, the additional CTC errors do not reduce the usefulness of the draft tokens to the verifier. At inference, hard routing runs one adapter rather than five, saving about 0.2 GFLOPs per chunk (less than 2% of the encoder cost), so the latency difference is negligible.

In summary, hard routing is a viable alternative to soft routing. Specifically, it produces a slightly better Transformer decoder and a marginally cheaper encoder at the cost of a slightly worse CTC decoder, with the same proposed speculative decoding efficiency.

5.5 Effect of the Number of Adapters

The mixture of adapters (Section 3.2) has one free hyperparameter that is the number of adapters K. We trained the ViT-S model with $K = 3$, 5, and 8 while keeping other parts unchanged. Each adapter, together with its router output, adds only 0.1 M parameters. Thus, the three models

Table 7: Effect of the backbone. Best CER per column in bold, second best underlined. SPEC: Speculative.

Backbone	Dec.	Params (M)	CER (%) KHOB 1,318	KhmerST 3,021	KH 211	GKST 212	KHT 711	Overall 5,473	Time (ms)	Passes / line
ResNet-45 + Transformer Encoders (UKTR [2], published)	CTC	–	2.46	3.02	5.89	4.41	9.52	–	27†	–
	AR		2.37	2.19	4.11	3.34	6.10	–	777†	–
ResNet-45 + Transformer Encoders (UKTR [2], re-impl.)	CTC	54.17	**1.97**	2.24	4.17	3.22	6.95	2.32	19.14	–
	AR		2.12	1.83	3.67	**2.46**	5.72	2.22	287.66	17.42
	SPEC		2.12	1.83	3.67	**2.46**	5.72	2.22	41.40 (6.95×)	1.38
ViT-S(Ours)	CTC	43.00	2.34	2.22	3.89	3.56	6.58	2.54	24.21	–
	AR		**1.97**	**1.62**	**2.89**	2.96	**4.79**	**2.03**	301.08	17.34
	SPEC		**1.97**	**1.62**	**2.89**	2.96	**4.79**	**2.03**	49.43 (6.09×)	1.41

Table 8: Soft versus hard adapter routing. Best CER per column in bold, second best underlined. SPEC: Speculative.

Routing	Decoding	CER (%) KHOB 1,318	KhmerST 3,021	KH 211	GKST 212	KHT 711	Overall 5,473	Passes / line
Soft	CTC	2.34	2.22	3.89	3.56	6.58	2.54	–
	AR	1.97	**1.62**	2.89	2.96	**4.79**	2.03	17.34
	SPEC	1.97	**1.62**	2.89	2.96	**4.79**	2.03	**1.41**
Hard	CTC	2.26	2.40	4.11	2.81	7.68	2.58	–
	AR	**1.93**	1.68	**2.61**	**1.73**	4.87	**1.99**	17.35
	SPEC	**1.93**	1.68	**2.61**	**1.73**	4.87	**1.99**	1.42

have 42.80 M, 43.00 M, and 43.30 M parameters, respectively. Table 9 reports model recognition performance and latency.

Increasing the number of adapters lowers the overall CER. Specifically, under the AR decoding, it decreases from 2.21% ($K = 3$) to 2.03% ($K = 5$) and 1.93% ($K = 8$). This corresponds to relative reductions of 8% and 5%, respectively. The CTC decoding is less sensitive (2.56%, 2.54%, and 2.44%). $K = 3$ is clearly insufficient for the Transformer (AR) decoder. Specifically, moving from $K = 3$ to $K = 5$ reduces the AR CER by 20% on the KhmerST dataset (2.03% → 1.62%) and by 15% on the KHT dataset (5.65% → 4.79%). Additionally, moving from $K = 5$ to $K = 8$ improves every evaluation dataset except the KhmerST dataset, where the CER increases slightly from 1.62% to 1.70%.

Extra adapters benefit the Transformer decoder more than the CTC decoder. The AR decoding corrects 14% of the CTC errors with $K = 3$, 20% with $K = 5$, and 21% with $K = 8$. The CTC drafts improve slightly as well. Specifically, the number of verification passes per textline decreases from 1.45 to 1.41 and 1.40. For all three cases, the proposed speculative decoding reproduces the greedy AR output on 100% of the textlines.

In summary, a larger number of adapters K markedly improves recognition performance, especially under the AR decoding, although the gains become marginal beyond $K = 5$. Speculative decoding remains robust across all values of K, matching the greedy AR output exactly with only a small fraction of the AR decoding steps.

6 Limitations and Future Work

We note the following limitations of our study:

- The proposed speculative decoding scheme presumes that AR decoding is more accurate but slower than NAR decoding, so it reduces latency without sacrificing AR accuracy. In some cases,

Table 9: Effect of the number of adapters K in the mixture of adapters (ViT-S, soft routing). Best CER per column in bold, second best underlined. SPEC: Speculative.

			CER (%)						Passes
K	Params (M)	Decoding	KHOB 1,318	KhmerST 3,021	KH 211	GKST 212	KHT 711	Overall 5,473	/ line
		CTC	2.29	2.26	4.89	3.09	7.38	2.56	–
3	42.80	AR	2.01	2.03	<u>2.89</u>	<u>2.71</u>	5.65	2.21	17.38
		SPEC	2.01	2.03	<u>2.89</u>	<u>2.71</u>	5.65	2.21	1.45
		CTC	2.34	2.22	3.89	3.56	6.58	2.54	–
5	43.00	AR	<u>1.97</u>	**1.62**	<u>2.89</u>	2.96	<u>4.79</u>	<u>2.03</u>	17.34
		SPEC	<u>1.97</u>	**1.62**	<u>2.89</u>	2.96	<u>4.79</u>	<u>2.03</u>	<u>1.41</u>
		CTC	2.19	2.12	3.61	2.84	7.49	2.44	–
8	43.30	AR	**1.84**	<u>1.70</u>	**2.67**	**2.14**	**4.42**	**1.93**	17.36
		SPEC	**1.84**	<u>1.70</u>	**2.67**	**2.14**	**4.42**	**1.93**	**1.40**

however, such as when attention drifts, AR decoding performs worse than CTC decoding. Future work could address this by automatically choosing the more suitable decoder for each input image.

- This study does not include any fine-tuned VLM-based baselines. Future work should fine-tune VLMs on Khmer text and compare them against the proposed model.
- This study is limited to printed, scene, and handwritten textline images. Future work should evaluate whether the proposed model and decoding scheme generalize to other modalities, such as historical documents like palm leaf manuscripts.

7 Conclusion

We introduce a novel lossless speculative decoding scheme that combines NAR and AR decoders for Khmer textline recognition, together with an improved architecture built on a hybrid vision Transformer (ViT) backbone. The scheme preserves the accuracy of AR decoding while substantially reducing the number of decoding steps. Experiments on printed, scene, and handwritten datasets show that the proposed model achieves the best overall performance against re-implemented baselines and existing Khmer text recognizers. In addition, the proposed decoding scheme matches the AR output exactly while lowering latency by 6.09× (without KV caches) and the number of sequential decoder passes by 12.3×. The code, models, and datasets are available on GitHub.

Acknowledgment. This study was supported by JSPS Kakenhi Grant Number 26K14965 and JST CRONOS Grant Number JPMJCS24K4.

References

[1] Thuon, N. & Theang, P. *Do mllms really understand low-resource khmer documents? a pilot study on khmer document vqa. International Conference on Pattern Recognition*, 247–262 (Springer, 2026).

[2] Kong, M. *et al. Towards universal khmer text recognition. International Conference on Document Analysis and Recognition*, 698–715 (Springer, 2026).

[3] Buoy, R., Iwamura, M., Srun, S. & Kise, K. Toward a low-resource non-latin-complete baseline: An exploration of khmer optical character recognition. *IEEE Access* **11**, 128044–128060 (2023).

[4] Buoy, R., Iwamura, M., Srun, S. & Kise, K. *Language-aware non-autoregressive khmer textline recognition. International Conference on Pattern Recognition and Artificial Intelligence*, 339–353 (Springer, 2024).

[5] Graves, A., Fernández, S., Gomez, F. & Schmidhuber, J. *Connectionist temporal classification: labelling unsegmented sequence*

data with recurrent neural networks. Proceedings of the 23rd international conference on Machine learning, 369–376 (2006).

[6] Du, Y., Chen, Z., Xie, H., Jia, C. & Jiang, Y.-G. *Svtrv2: Ctc beats encoder-decoder models in scene text recognition. 2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, 20147–20156 (IEEE, 2025).

[7] Bautista, D. & Atienza, R. *Scene text recognition with permuted autoregressive sequence models. Proceedings of European conference on computer vision*, 178–196 (Springer, 2022).

[8] Leviathan, Y., Kalman, M. & Matias, Y. *Fast inference from transformers via speculative decoding. International conference on machine learning*, 19274–19286 (PMLR, 2023).

[9] Dosovitskiy, A. *et al.* An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[10] Shi, B., Bai, X. & Yao, C. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **39**, 2298–2304 (2017).

[11] Hernandez Diaz, D., Qin, S., Ingle, R., Fujii, Y. & Bissacco, A. Rethinking text line recognition models. *arXiv e-prints* arXiv–2104 (2021).

[12] Baek, J. *et al. What is wrong with scene text recognition model comparisons? dataset and model analysis. Proceedings of the IEEE/CVF international conference on computer vision*, 4715–4723 (2019).

[13] Wang, J. & Hu, X. Gated recurrent convolution neural network for ocr. *Advances in Neural Information Processing Systems* **30** (2017).

[14] Liu, W., Chen, C., Wong, K.-Y. K., Su, Z. & Han, J. *Star-net: a spatial attention residue network for scene text recognition. BMVC*, Vol. 2, 7 (2016).

[15] Atienza, R. *Vision transformer for fast and efficient scene text recognition. Proceedings of International Conference on Document Analysis and Recognition*, 319–334 (Springer, 2021).

[16] Buoy, R., Iwamura, M., Srun, S. & Kise, K. Vitstr-transducer: Cross-attention-free vision transformer transducer for scene text recognition. *Journal of Imaging* **9**, 276 (2023).

[17] Luan, X., Zhang, J., Xu, M., Silamu, W. & Li, Y. Lightweight scene text recognition based on transformer. *Sensors* **23**, 4490 (2023).

[18] Shi, B. *et al.* Aster: An attentional scene text recognizer with flexible rectification. *IEEE transactions on pattern analysis and machine intelligence* **41**, 2035–2048 (2018).

[19] Shi, B., Wang, X., Lyu, P., Yao, C. & Bai, X. *Robust scene text recognition with automatic rectification. Proceedings of the IEEE conference on computer vision and pattern recognition*, 4168–4176 (2016).

[20] Lee, C.-Y. & Osindero, S. *Recursive recurrent nets with attention modeling for ocr in the wild. Proceedings of the IEEE conference on computer vision and pattern recognition*, 2231–2239 (2016).

[21] Lin, Q., Luo, C., Jin, L. & Lai, S. Stan: A sequential transformation attention-based network for scene text recognition. *Pattern Recognition* **111**, 107692 (2021).

[22] Cheng, Z. *et al. Focusing attention: Towards accurate text recognition in natural images. 2017 IEEE international conference on computer vision (ICCV)*, 5086–5094 (IEEE, 2017).

[23] Alshawi, A. A. A., Tanha, J. & Balafar, M. A. An attention-based convolutional recurrent neural networks for scene text recognition. *IEEE Access* **12**, 8123–8134 (2024).

[24] Li, H., Wang, P., Shen, C. & Zhang, G. *Show, attend and read: A simple and strong baseline for irregular text recognition. Proceedings of the AAAI conference on artificial intelligence*, Vol. 33, 8610–8617 (2019).

[25] Vaswani, A. *et al. Attention is all you need. Proceedings of Advances in Neural Information Processing Systems*, Vol. 30 (Curran Associates, Inc., 2017). URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[26] Selvam, P. *et al.* A transformer-based framework for scene text recognition. *IEEE Access* **10**, 100895–100910 (2022).

[27] Lee, J. *et al. On recognizing texts of arbitrary shapes with 2d self-attention. 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2326–2335 (IEEE, 2020).

[28] Buoy, R., Iwamura, M., Srun, S. & Kise, K. Parstr: partially autoregressive scene text recognition: R. buoy et al. *International Journal on Document Analysis and Recognition (IJDAR)* **27**, 303–316 (2024).

[29] Fang, S., Xie, H., Wang, Y., Mao, Z. & Zhang, Y. *Read like humans: Autonomous, bidirectional and iterative language modeling for scene text recognition. 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 7094–7103 (IEEE, 2021).

[30] Jiang, Q., Wang, J., Peng, D., Liu, C. & Jin, L. *Revisiting scene text recognition: A data perspective. 2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, 20486–20497 (IEEE, 2023).

[31] Qiao, Z. *et al.* Pimnet: A parallel, iterative and mimicking network for scene text recognition. *Proceedings of the 29th ACM International Conference on Multimedia* (2021).

[32] Cai, T. *et al.* Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774* (2024).

[33] Wen, Z., Gui, S. & Feng, Y. Speculative decoding with ctc-based draft model for llm inference acceleration. *Advances in Neural Information Processing Systems* **37**, 92082–92100 (2024).

[34] Team, G. *et al.* Gemma 4 technical report. *arXiv preprint arXiv:2607.02770* (2026).

[35] Bai, S. *et al.* Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631* (2025).

[36] Wei, H., Sun, Y. & Li, Y. Deepseek-ocr: Contexts optical compression. *arXiv preprint arXiv:2510.18234* (2025).

[37] Sok, P. & Taing, N. *Support vector machine (svm) based classifier for khmer printed character-set recognition. Proceedings of Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2014 Asia-Pacific*, 1–9 (IEEE, 2014).

[38] Valy, D., Verleysen, M. & Chhun, S. *Text recognition on khmer historical documents using glyph class map generation with encoder-decoder model. Proceedings of the 8th International Conference on Pattern Recognition Applications and Methods* (2019).

[39] Valy, D., Verleysen, M. & Chhun, S. *Data augmentation and text recognition on khmer historical manuscripts. Proceedings of 17th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, 73–78 (IEEE, 2020).

[40] Annanurov, B. & Noor, N. A compact deep learning model for khmer handwritten text recognition. *IAES International Journal of Artificial Intelligence* **10**, 8938–8938 (2021).

[41] Keo, S., Coustaty, M., Bakkali, S. & Rossinyol, M. *State-of-the-art khmer text recognition using deep learning models. ASEAN Conference on Emerging Technologies 2024* (2024).

[42] Buoy, R., Taing, N., Chenda, S. & Kor, S. Khmer printed character recognition using attention-based seq2seq network. *Ho Chi Minh City Open University Journal Of Science - Engineering And Technology* **12**, 3–16 (2022).

[43] Xiao, T. *et al.* Early convolutions help transformers see better. *Advances in neural information processing systems* **34**, 30392–30400 (2021).

[44] Touvron, H., Cord, M. & Jégou, H. *Deit iii: Revenge of the vit. European conference on computer vision*, 516–533 (Springer, 2022).

[45] Darcet, T., Oquab, M., Mairal, J. & Bojanowski, P. *Vision transformers need registers. International conference on learning representations*, Vol. 2024, 2632–2652 (2024).

[46] Xiong, R. *et al. On layer normalization in the transformer architecture. International conference on machine learning*, 10524–10533 (PMLR, 2020).

[47] EKYCSolutions. Ekycsolutions/khmer-ocr-benchmark-dataset: A standardized benchmark dataset for khmer optical character recognition (ocr) engine. URL https://github.com/EKYCSolutions/khmer-ocr-benchmark-dataset.

[48] Buoy, R. *et al.* Addressing the attention drift problem for khmer long textline recognition. *International Journal on Document Analysis and Recognition (IJDAR)* **29**, 241–266 (2026).

[49] Long, S. *et al. Towards end-to-end unified scene text detection and layout analysis. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2022).

[50] Nom, V., Bakkali, S., Luqman, M. M., Coustaty, M. & Ogier, J.-M. Khmerst: A low-resource khmer scene text detection and recognition benchmark (2024). URL https://arxiv.org/abs/2410.18277.

[51] Nom, V. *et al. Wildkhmerst: a comprehensive dataset and benchmark for khmer scene text detection and recognition in the wild. International Conference on Document Analysis and Recognition*, 351–368 (Springer, 2025).

[52] Tesseract-Ocr. Tesseract-ocr/tesseract: Tesseract open source ocr engine (main repository). URL https://github.com/tesseract-ocr/tesseract.

[53] Paruchuri, V. & Team, D. Surya: A lightweight document ocr and analysis toolkit. https://github.com/datalab-to/surya (2025). GitHub repository.

[54] mrrtmob. Kiri OCR: A lightweight ocr library for english and khmer documents. GitHub repository, https://github.com/mrrtmob/kiri-ocr (2025). Accessed: 2026-11-14.

[55] KHUN, K. Troryongocr: A Novel Encoder-Decoder Model for Scene Text Recognition. URL https://kimang18.github.io/krorngai-blog/TrorYongOCR/.

[56] author names from Paper/*.pdf, T. A squeeze-and-excitation transformer network for khmer optical character recognition (2026). Preprint. Available at https://github.com/netra-ai-lab/Netra-OCR.

[57] Din, S. anaksor-ocr: Khmer and latin ocr detection and recognition weights. https://huggingface.co/Sokheng/anaksor-ocr (2025). Revision `rec-v4-det-dbpp-v1`. Accessed: 2026-11-14.

[58] Vitou, S., Y, K., Lany, M., Botum, C. & Kor, S. Khmer handwritten OCR using pretrained TrOCR architecture. Poster, Royal University of Phnom Penh (2024). URL https://www.researchgate.net/publication/385417010_Khmer_Handwritten_OCR_using_Pretrained_TrOCR_Architecture. ResearchGate.

[59] Huffman, F. E. *Cambodian system of writing and beginning reader* (SEAP Publications, 1987).

8 Appendix

8.1 Khmer Script

Khmer is the official language of Cambodia. Khmer script descends from the Brahmi family [59]. Khmer script is an abugida writing system written from left to right. Each consonant carries an inherent vowel that is invisible while other vowels appear as glyphs attached to the consonant. Words are not separated by spaces. In Unicode, Khmer occupies the block U+1780-U+17FF, while the lunar-date symbols are placed in the separate Khmer Symbols block, U+19E0-U+19FF. Table 10 lists the Khmer alphabet inventory grouped by character class.

8.1.1 Consonants

Modern Khmer has 33 consonants, which fall into two series (registers) based on their inherent vowel. The 15 first-series consonants carry *-a* (the *a*-series), and the 18 second-series consonants carry *-o* (the *o*-series). The pronunciation of every dependent vowel depends on the consonant series, and thus, each vowel can be pronounced in two ways.

8.1.2 Subscript Consonants

Consonant clusters are formed by stacking: the invisible sign *coeng* (U+17D2), placed before a consonant. This transforms the consonant into its subscript form, which is attached to the base consonant. Almost all consonants have a subscript form. Most subscripts are rendered below the base consonant but in some cases, it is rendered to its left or right. Subscripts are often rendered much smaller than the base consonant and can differ markedly in shape. Some consonants have the same subscript forms when rendered.

8.1.3 Vowels

Khmer has 16 dependent vowel signs (U+17B6-U+17C5). Pairing these with the signs *nikahit* (U+17C6) and *reahmuk* (U+17C7) yields 7 composite vowels. Thus, the total number of dependent vowels is 23. In addition, Khmer has 14 independent vowels (U+17A5-U+17B3), each of which serves as the base consonant.

8.1.4 Diacritics and Signs

Khmer has 14 diacritics. The two *shifters*, *muusikatoan* (U+17C9) and *triisap* (U+17CA) are used to change a consonant's series. *Bantoc* (U+17CB) shortens the vowel, *robat* (U+17CC) marks an /r/ in Sanskrit loanwords, and *toandakhiat* (U+17CD) marks a silent final letter.

8.1.5 Digits, Punctuation, and Symbols

Khmer has its own decimal digits (U+17E0-U+17E9) as well as a set of divine numerals (U+17F0-U+17F9). Among the 9 punctuation marks and symbols (U+17D4-U+17DC) are *khan* and *bariyoosan*, which terminate a sentence and a section, respectively. *lek too* is used to repeat the preceding word.

8.2 Khmer Character Clusters

The basic orthographic unit of Khmer is the *Khmer character cluster* (KCC), which comprises a base character and all of the dependent marks attached to it. Splitting a KCC would break its rendering and its reading. Following the Unicode encoding model, a cluster has the simplified structure

$$\mathrm{KCC} \;=\; B\,\big(\textsc{coeng}\;C\big)^{\{0,1,2\}}\;M^{?}\;V^{?}\;S^{*}, \qquad (8)$$

where B is a base consonant or independent vowel, $\textsc{coeng}\;C$ is a subscript consonant, M is a shifter, V is a dependent vowel, and S is a sign such as *nikahit*, *reahmuk*, *bantoc* or *robat*.

A cluster is laid out in two dimensions around the base (Fig. 5a). A zone can hold more than one mark. *Toandakhiat* (U+17CD), which marks a silent final consonant, is usually written on a bare consonant. When it follows a vowel, the vowel is almost always *i* (U+17B7), and *toandakhiat* is stacked on top of it (Fig. 5d).

In printed text, a cluster contains at most two subscripts, so the most complex clusters combine a base, two subscripts, a dependent vowel, and a sign (Fig. 5b,c). Such a cluster spans seven code points and can occupy all four zones around the base simultaneously (Fig. 5b).

Table 10: Inventory of the Khmer script in the Unicode Khmer block. Dependent marks are shown on a dotted circle (U+25CC) in the position where they attach to the base. Gray characters are obsolete or deprecated and rare in modern text.

Class (code points)	#	**Inventory**
1st-series consonants U+1780–17A2	15	ក ខ ច ឆ ដ ឋ ណ ត ថ ប ផ ស ហ ឡ អ
2nd-series consonants U+1780–17A2	18	គ ឃ ង ជ ឈ ញ ឌ ឍ ទ ធ ន ព ភ ម យ រ ល វ obsolete: ឝ ឞ
Subscript consonants *coeng* U+17D2 + C	32	្ក ្ខ ្គ ្ឃ ្ង ្ច ្ឆ ្ជ ្ឈ ្ញ ្ដ ្ឋ ្ឌ ្ឍ ្ណ ្ត ្ថ ្ទ ្ធ ្ន ្ប ្ផ ្ព ្ភ ្ម ្យ ្រ ្ល ្វ ្ឝ ្ឞ ្ស ្ហ ្ឡ ្អ
Dependent vowels U+17B6–17C5	16	ា ិ ី ឹ ឺ ុ ូ ួ ើ ឿ ៀ េ ែ ៃ ោ ៅ
composite vowels U+17C6, U+17C7	7	ុំ ំ ាំ ះ ុះ េះ ោះ (16 + 7 = 23 traditional dependent vowels)
Independent vowels U+17A5–17B3	14	ឥ ឦ ឧ ឨ ឩ ឪ ឫ ឬ ឭ ឮ ឯ ឰ ឱ ឲ ឳ deprecated: ឣ ឤ ឵
Diacritics and signs U+17C6–17D2, U+17DD	14	ំ ះ ៈ ៉ ៊ ់ ៌ ៍ ៎ ៏ ័ ៑ ្ ៓ nikahit reahmuk yuukalea. muusikat. triisap bantoc robat toandakh. kakabat ahsda samyok viriam coeng atthacan
Digits U+17E0–17E9	10	០ ១ ២ ៣ ៤ ៥ ៦ ៧ ៨ ៩ divination digits U+17F0–17F9: ᧰ ᧱ ᧲ ᧳ ᧴ ᧵ ᧶ ᧷ ᧸ ᧹
Punctuation, symbols U+17D4–17DC	9	។ ៕ ៖ ៗ ៘ ៙ ៚ ៛ ៜ khan bariyoosan camnuc pii k. lek too beyyal phnaek muan koomuut riel avakrahas.

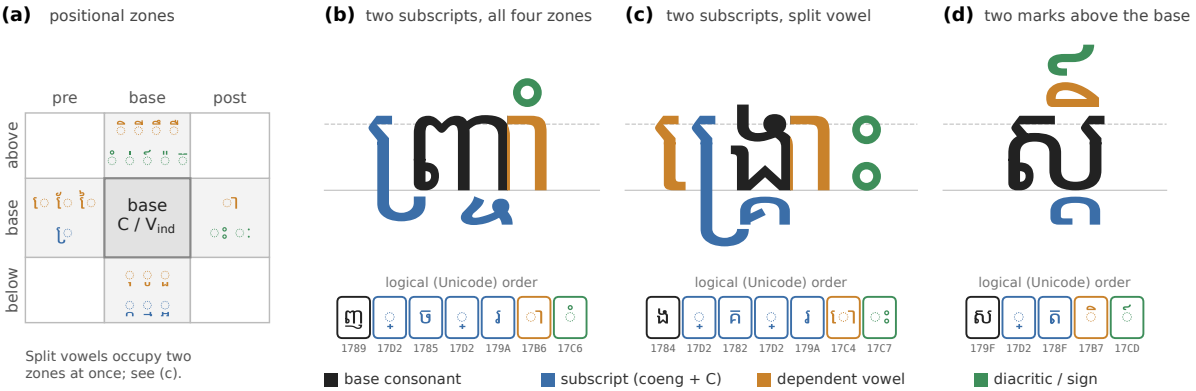

Fig. 5: Layout of Khmer character clusters. (a) Positional zones around the base. Dependent vowels, subscripts and signs can be placed before, after, above or below the base. (b)–(d) Three clusters with each component colored by its role, and their logical (Unicode) order below them.

8.3 Character-Level vs. Cluster-Level Tokenization

Text can be tokenized into Unicode code points (character level) or into Khmer character clusters

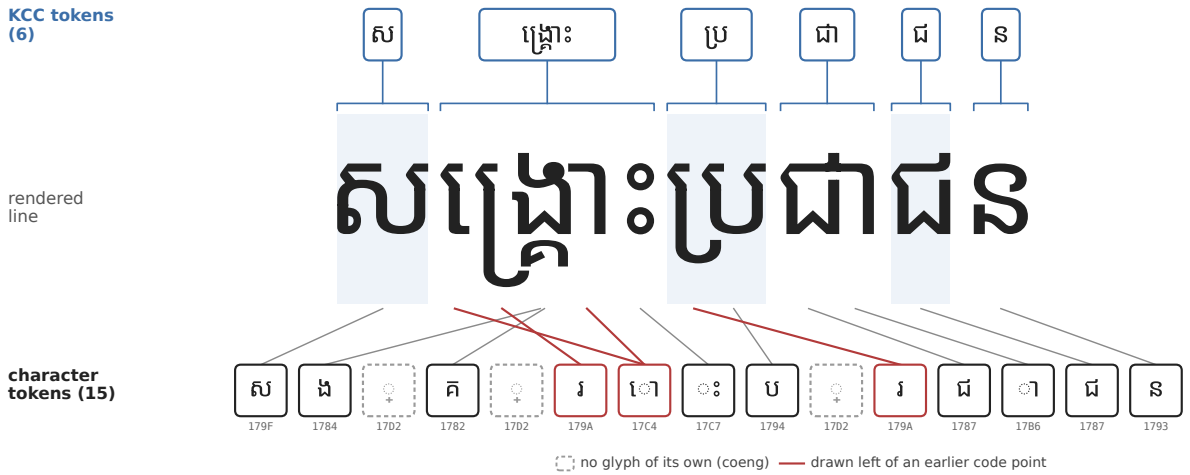

Fig. 6: Character-level and KCC tokenization of the line *sangkruoh pracheachon* 'rescue the people'.

(KCC level) using the above KCC rule defined in Section 8.2. Fig. 6 shows both tokenizations of the same line.

8.4 Prefix-Constrained CTC Re-Drafting

After a correction, $\mathbf{g}$ no longer agrees with the unconstrained CTC path. Simply continuing the old draft tokens would misalign every following token and would not take advantage of the speculative decoding. We instead find the best CTC path whose collapsed labels *start with* $\mathbf{g} = (g_1, \ldots, g_{|\mathbf{g}|})$ (Fig. 4c). Let $\mathbf{z} = (\epsilon, g_1, \epsilon, g_2, \ldots, g_{|\mathbf{g}|}, \epsilon)$ be the blank-interleaved label sequence. A Viterbi pass with the standard CTC transitions (stay, advance by one, or skip a blank between two different labels),

$$\alpha_t(s) = P_t(z_s) + \max\big\{\alpha_{t-1}(s),\ \alpha_{t-1}(s-1),\ \alpha_{t-1}(s-2) + \log \delta_s\big\}, \tag{9}$$

where $\delta_s = 1$ if $z_s \neq \epsilon$ and $z_s \neq z_{s-2}$ and 0 otherwise. This gives $V_t(\mathbf{g}) = \max\{\alpha_t(S), \alpha_t(S-1)\}$, the best score of emitting exactly $\mathbf{g}$ in frames $1..t$, where $S = |\mathbf{z}|$. Since the frames after the split point are unconstrained, their best path is the per-frame argmax, and the optimal split frame is

$$t^* = \arg\max_t \Big[V_t(\mathbf{g}) + \sum_{t'>t} \max_c P_{t'}(c)\Big], \tag{10}$$

computed for all t in one $O(T \cdot |\mathbf{g}|)$ pass using suffix sums. The new draft is the collapsed argmax path after t^*, $\mathbf{d} = \mathcal{B}(\pi_{t^*+1:T}) \oplus$ `</s>`, where a repeat of $g_{|\mathbf{g}|}$ is merged only if the aligned path ends on that label rather than on a blank. Tokens `<s>` and `</s>` emitted by CTC are never placed in a draft. If $\mathbf{g}$ cannot be aligned within T frames, the draft is only (`</s>`) and the next pass reduces to one ordinary AR step.

8.5 Line Normalization and Overlapping Chunking

Each grayscale textline image is resized to a common height $H = 32$ pixels with its aspect ratio preserved, giving $I \in \mathbb{R}^{H \times W}$ for an arbitrary width W. To avoid the quadratic cost of self-attention complexity for a long textline image, instead of encoding the whole textline image at once, we split it into $N = \lceil W/w \rceil$ chunks with a core width of $w = 112$ pixels. Each chunk is padded by $m = 8$ pixels on both sides, so the input textline image is always $32 \times (w + 2m) = 32 \times 128$ pixels:

$$C_n = \tilde{I}\big[:,\ nw\ :\ nw{+}w{+}2m\big], \qquad n = 0, \ldots, N{-}1, \tag{11}$$

where $\tilde{I}$ is I right-padded to Nw columns and padded by m columns on the left and right. A line shorter than w yields a single chunk. All chunks of a mini-batch are stacked into a tensor of shape $(\sum_b N_b) \times 1 \times 32 \times 128$, so lines of different

widths are batched without resizing, and cumulative chunk counts record which line each chunk belongs to.

8.6 Seam-Aware Overlap-Add Stitching

Cropping the margins would leave a discontinuity at each seam. Instead, we cross-fade the two chunks across the overlap with a fixed window over the $w_c = 32$ chunk columns,

$$\omega(j) = \min\Big(1,\ \frac{\min\big(j + \tfrac{1}{2},\ w_c - j - \tfrac{1}{2}\big)}{2m'}\Big), \quad (12)$$

for $j = 0, \ldots, w_c - 1$, which ramps linearly over the four outermost columns of each side $(0.125, 0.375, 0.625, 0.875)$ and is one elsewhere. The ramps of two neighbours are complementary, so the weights sum to one at every seam, and the columns with the least context receive the least weight. Placing chunk n at global column offset $28n$, the line-level feature map is

$$\mathbf{F}[x] = \frac{\sum_n \omega(j_n)\, \mathbf{f}_n[j_n]}{\sum_n \omega(j_n)}, \qquad j_n = x - 28n + m', \quad (13)$$

for $x = 0, \ldots, W' - 1$, where the sums run over the chunks that cover column x, and the two outermost margins are cropped. This yields $\mathbf{F} \in \mathbb{R}^{W' \times 8 \times 384}$ with $W' = 28N$. Columns beyond the true line width, $x \geq \lceil W/4 \rceil$, are marked as padding by a validity mask $\mathbf{v} \in \{0, 1\}^{W'}$.