# DEPTH-CONDITIONED LOOPED TRANSFORMERS FOR ADAPTIVE ASR INFERENCE

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

End-to-end automatic speech recognition (ASR) systems use fixed-depth encoders, so each accuracy–compute budget typically requires its own trained model. We ask whether a single compact encoder can instead make its inference budget a deployment-time choice. A natural approach is to reuse a shared Transformer block recurrently, but we find that naive looping does not fully exploit the additional recurrent compute. We introduce LARM, a depth-conditioned looped Transformer that combines sparse connectionist temporal classification (CTC) checkpoints, supervision-clock embeddings, FiLM depth conditioning, and soft-posterior feedback. These components structure the loop into recognition checkpoints separated by latent refinement phases and let shared weights specialize across recurrent steps. A single trained model can thus stop at any checkpoint within its trained loop budget $K$. On LibriSpeech, LARM improves WER as more loops are executed within this budget, and in a common pipeline it achieves lower WER than closely related self-conditioning and recurrent CTC methods. The design also transfers to a Conformer backbone. Compared with deeper unshared encoders in the same pipeline, LARM reaches equal or better WER with far fewer parameters and less peak memory, making it well suited to memory-constrained deployment.

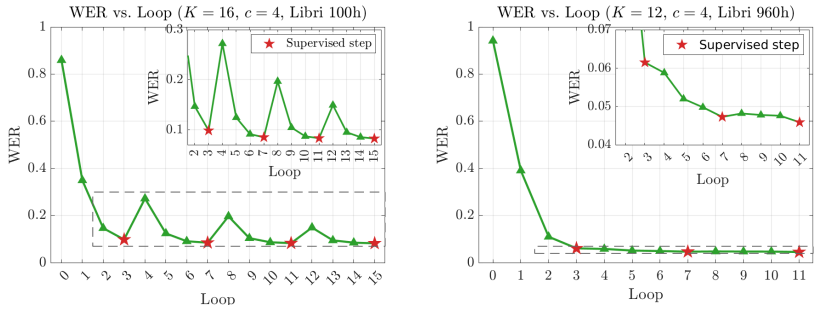
## 1 INTRODUCTION

End-to-end automatic speech recognition (ASR) models typically use a fixed-depth acoustic encoder followed by a connectionist temporal classification (CTC) (Graves et al., 2006), transducer, or attention-based prediction head. While this design has been effective, the computational depth of the encoder is fixed by the architecture, so every input and every deployment target receives the same computation. In practice, ASR systems run under very different constraints, from memory-limited on-device settings to server-side recognition where accuracy matters most. Serving these regimes with fixed-depth encoders typically requires training, validating, and storing a separate model for each budget. Approaches that vary encoder depth at inference, such as structured layer dropout (Fan et al., 2019) or dynamic encoders (Shi et al., 2021), can serve several depths from one model, but must store the full unshared stack, so the memory footprint is set by the largest budget. This motivates encoders whose inference computation can be modulated within a single compact model, without storing parameters for the largest budget.

In language modeling, varying inference computation without changing the learned parameters has recently gained significant traction. Autoregressive generation naturally supports longer inference-time computation that can substitute for parameter scaling (Graves, 2016; Wang et al., 2025). Looped or recursive Transformers, in which a shared block of weights is applied repeatedly, offer a natural way to decouple computational depth from parameter count (Dehghani et al., 2018; Giannou et al., 2023; Jolicoeur-Martineau, 2025). However, adapting this paradigm to continuous, non-autoregressive sequence-to-sequence tasks like ASR is non-trivial. While iterative refinement in ASR has been explored, prior works primarily focus on masking or correcting discrete output hypotheses (Higuchi et al., 2020; Chi et al., 2021) rather than progressively deepening the core acoustic-linguistic representation. Motivated by recent looped architectures in language modeling, we applied recurrence within the acoustic encoder, but found that simply reapplying the same block did not fully exploit the additional loops, motivating conditioning mechanisms that let successive iterations specialize.



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.




(a) WER across recurrent loops (b) Smooth refinement regime

Figure 1: **Loop-structured inference in LARM.** WER trajectories on LibriSpeech 100h ($K = 16$, $c = 4$) and 960h ($K = 12$, $c = 4$). Red stars denote supervised loops. Intermediate loops can be non-monotonic (a) or smooth (b), but in both cases each supervised checkpoint improves on the previous one, supporting sparse supervision as an anchor for latent acoustic refinement.

To this end, we propose the **Loop Audio Recurrent Model (LARM)**, a shared-parameter architecture that conditions repeated computation to support stage-specific processing. LARM applies a shared Transformer block recurrently to a latent acoustic sequence, effectively treating computational depth as a dynamic axis. To avoid redundant computation across recurrent steps, LARM introduces three key structural mechanisms (see Figure 2). First, we employ a sparse supervision schedule and a supervision clock embedding. By applying CTC loss only at periodic loop intervals, we structure the iterative process into a sequence of supervised recognition checkpoints separated by intermediate refinement steps that are not directly supervised by CTC (see Figure 1). Second, we utilize FiLM-based depth conditioning (Perez et al., 2018), which explicitly modulates the hidden state based on the normalized loop iteration, allowing the shared weights to perform specialized roles as computation progresses. Finally, we introduce delayed prediction feedback, an ASR-specific recurrent mechanism where soft CTC posteriors from the previous iteration are shifted by one frame and reinjected into the network. This explicitly propagates left-to-right token-level continuity through the depth of the model, guiding the trajectory of acoustic refinement.

Because every supervised checkpoint is trained as a recognition output, a single trained LARM model can stop at any checkpoint within its trained loop budget $K$. We refer to this as *budget-adaptive inference*. The compute budget is selected at deployment rather than per input, and within the trained range rather than beyond it. Our main contributions can be summarized as follows:

- **Budget-Adaptive Inference for ASR:** We introduce LARM, a depth-conditioned looped Transformer for ASR that decouples parameter count from inference depth. A single trained model can operate at several compute budgets with a nearly constant memory footprint.
- **Stage-Specialized Recurrent Training:** We propose a novel loop-structuring mechanism combining sparse intermediate CTC supervision, periodic clock embeddings, and FiLM depth conditioning, successfully specializing a shared encoder across recurrent iterations.
- **Prediction Feedback:** We introduce an ASR-specific feedback mechanism that reinjects soft-token posteriors from the previous loop into the recurrent state, improving iterative acoustic recognition.
- **Empirical validation on LibriSpeech (Panayotov et al., 2015):** LARM improves across supervised checkpoints within its trained budget, matches or outperforms deeper unshared encoders with a fraction of their parameters, outperforms closely related self-conditioning and recurrent CTC methods in a common pipeline, and transfers to a Conformer backbone.

## 2 Related Work

We position LARM at the intersection of adaptive inference-time computation, looped Transformer architectures, and iterative refinement methods for ASR.

**Adaptive Inference-Time Computation.** Dynamically allocating additional computation at inference has long been studied, with Adaptive Computation Time providing an early mechanism for input-dependent recurrent computation (Graves, 2016). More recently, test-time compute scaling has become central in large language models, where search, verification, self-correction, and iterative refinement can substantially improve reasoning performance without changing model parameters (Snell et al., 2024; Jaech et al., 2024; Muennighoff et al., 2025; Madaan et al., 2023). A related direction

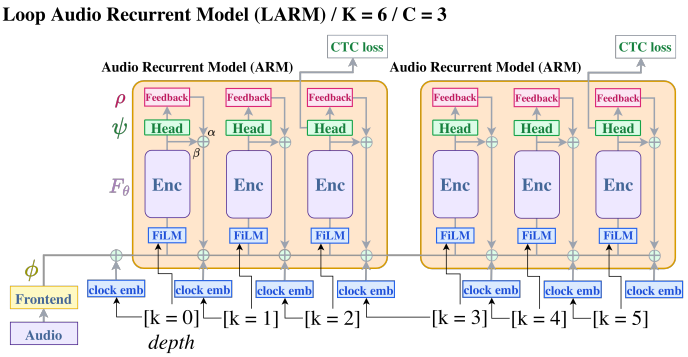



Figure 2: **Overview of the LARM architecture.** The acoustic frontend maps the input audio to an initial latent representation, which a shared encoder $F_\theta$ processes recurrently. The frontend output is also passed to every loop. Each loop uses FiLM depth conditioning on the normalized loop index $k$, supervision-clock embeddings on $k \bmod c$, and delayed prediction feedback from a shared CTC head $\psi$, weighted by the learnable scalars $\alpha$ and $\beta$. CTC loss is applied only at sparse recognition checkpoints, and the intervening loops refine the latent representation. See Section 3 for details.

is latent reasoning, where models perform additional internal computation before emitting outputs, for example through pause tokens, internal rationales, or filler-token computation (Goyal et al., 2023; Zelikman et al., 2024; Pfau et al., 2024; Wang et al., 2025). These methods primarily exploit the autoregressive structure of language generation and aim to improve performance by spending more inference computation, whereas our goal is to let a continuous ASR encoder operate at several compute budgets within its trained range.

**Looped Architectures.** Architectures that decouple computational depth from parameter count provide a natural mechanism for this goal. Cross-layer parameter sharing was used in ALBERT to reduce memory and regularize deep models (Lan et al., 2019), while the Universal Transformer applied shared Transformer blocks recurrently across depth (Dehghani et al., 2018). Related formulations include Deep Equilibrium Models, which repeatedly apply a shared transformation to solve for a fixed point (Bai et al., 2019). Recent work further shows that recurrent or looped Transformers can act as powerful iterative computers, with both theoretical and empirical evidence for improved algorithmic and reasoning behavior (Pérez et al., 2019; Giannou et al., 2023; Schwarzschild et al., 2021; Wang et al., 2025; Jolicoeur-Martineau, 2025). Recursive Transformers have also been applied to speech recognition (Zhao et al., 2020; Jang et al., 2026) and made more parameter-efficient (Bae et al., 2025), but without supervised exits or depth-specialized loops. LARM adapts this looped Transformer view to acoustic modeling, using recurrent depth as a selectable inference budget rather than only as a parameter-sharing device.

**Iterative Refinement and Feedback in ASR.** In ASR, iterative refinement has mainly been used to narrow the gap between non-autoregressive (NAR) and autoregressive decoding. Methods such as Mask CTC (Higuchi et al., 2020), Imputer (Chan et al., 2020), and Align-Refine (Chi et al., 2021) repeatedly mask, correct, or realign discrete textual hypotheses after an initial CTC-style prediction. While effective, these approaches primarily refine outputs rather than deepen the continuous acoustic representation. Closer to LARM are methods that use intermediate acoustic predictions inside the encoder. Auxiliary intermediate CTC losses improve the optimization of deep ASR encoders (Sanabria & Metze, 2018), while Self-Conditioned CTC (Nozaki & Komatsu, 2021) and Intermediate CTC (Lee & Watanabe, 2021) feed internal CTC predictions forward to condition later layers and relax CTC's conditional-independence assumption. Self-Conditioned Folded Encoders (Komatsu, 2022) similarly explore reusing acoustic blocks across depth. However, these methods typically rely on fixed-depth computation, distinct layer weights, dense intermediate supervision, or limited control over the recurrent transition.

**Positioning of Our Work.** LARM brings the adaptive-inference perspective of looped architectures to continuous acoustic modeling. Unlike Mask CTC, Imputer, and Align-Refine, which iteratively revise discrete output hypotheses, LARM performs refinement inside the encoder by repeatedly transforming the continuous acoustic-linguistic representation. Unlike self-conditioned CTC models and folded encoders, which generally operate with fixed inference depth, dense intermediate prediction losses, or limited control over how computation changes across depth, LARM exposes recurrent encoder depth as an explicit inference-time compute axis. To this end, we introduce several mechanisms (Section 3)

that turn repeated encoder computation from simple weight reuse into stage-specialized acoustic refinement, enabling parameter-efficient budget-adaptive inference for ASR.

# 3 METHOD

In this section, we first introduce the standard looped Transformer formulation, then present the Loop Audio Recurrent Model (LARM). We describe its recurrent formulation, delayed prediction feedback, sparse supervision, supervision clock embedding, and FiLM-based depth conditioning, which together turn selected loops of a shared-parameter encoder into usable recognition outputs.

## 3.1 STANDARD LOOPED TRANSFORMER FORMULATION

In ASR, the model maps an acoustic input sequence $\boldsymbol{x} = \{\boldsymbol{x}_i\}_{i=1}^{n}$ to an output token sequence $\boldsymbol{y} = \{\boldsymbol{y}_j\}_{j=1}^{m}$. Standard encoder-based architectures increase representational depth by stacking distinct layers, so depth and parameter count typically grow together. A looped Transformer instead applies a shared block $F_\theta$ repeatedly to a latent representation, decoupling computational depth from parameter count. Given an input sequence $\boldsymbol{x}$, an initial latent representation is first obtained using a feature encoder $\phi$, after which the same block $F_\theta$ is applied for $K$ loops:

$$\mathbf{h}^{(0)} = \phi(\boldsymbol{x}), \qquad \tilde{\boldsymbol{y}}^{(K)} = \psi\left(\underbrace{F_\theta \circ F_\theta \circ \cdots \circ F_\theta}_{K \text{ times}}\left(\mathbf{h}^{(0)}\right)\right), \tag{1}$$

where $\psi$ denotes the CTC head.

## 3.2 LOOP AUDIO RECURRENT MODEL ARCHITECTURE

Building on this formulation, we describe how LARM structures recurrent shared-parameter computation for acoustic recognition. Figure 2 provides an overview of the model.

### 3.2.1 PRELIMINARIES

LARM distinguishes the output of the shared encoder at loop $k$, denoted $\mathbf{z}^{(k)}$, from the recurrent state passed to the next loop, denoted $\mathbf{h}^{(k)}$. The recurrent state $\mathbf{h}^{(k)}$ is formed by combining the encoder output with prediction feedback, the frontend skip connection, and loop-conditioning signals.

**Acoustic Frontend.** LARM maps log-Mel features $\boldsymbol{x} \in \mathbb{R}^{T\times 80}$ to the initial recurrent state $\mathbf{h}^{(0)} \in \mathbb{R}^{T'\times d}$ using an acoustic frontend $\phi$ with two strided 2D convolutional subsampling layers, a linear projection to dimension $d$, and dropout. The frontend reduces the time resolution by a factor of $4$.

**CTC Prediction Head.** Token predictions are produced by a shared CTC head $\psi$ applied to an encoder output $\mathbf{z}^{(k)} \in \mathbb{R}^{T'\times d}$. It maps the representation to frame-level logits $\boldsymbol{\ell}^{(k)} \in \mathbb{R}^{T'\times|\mathcal{V}|}$, where $\mathcal{V}$ includes the CTC blank symbol:

$$\boldsymbol{\ell}^{(k)} = \psi(\mathbf{z}^{(k)}) = \mathbf{z}^{(k)}\mathbf{W}_\psi^\top + \mathbf{b}_\psi, \qquad \mathbf{p}^{(k)} = \mathrm{softmax}\left(\boldsymbol{\ell}^{(k)}\right). \tag{2}$$

Further implementation details are provided in Appendix A.1.

**Shared Encoder.** The shared encoder $F_\theta$ is composed of $N$ stacked pre-norm Transformer blocks. followed by a feed-forward network with residual connections. The same encoder parameters are reused across all loops. Implementation details are provided in Appendix A.2.

### 3.2.2 SPARSE SUPERVISION AND LOOP STRUCTURE

A looped encoder produces a sequence of encoder outputs $\{\mathbf{z}^{(k)}\}_{k=1}^{K}$. Two natural supervision strategies are to apply the loss either only at the final loop or densely at every loop. Final-loop supervision leaves intermediate loop outputs without direct recognition supervision, while dense supervision provides recognition signals throughout the loop but encourages every loop to behave as a complete recognition stage.

LARM adopts an intermediate regime based on sparse supervision. The goal is to structure the loop as a sequence of recognition checkpoints separated by intervals of intermediate computation. We refer to the supervision period $c$ as the *checkpoint interval*: it determines the spacing between directly supervised recognition stages, leaving $c-1$ intermediate loops for latent refinement between consecutive checkpoints. Selected loops therefore act as recognition stages, while intermediate loops remain available to refine the representation before the next checkpoint. As illustrated in Figure 2, supervised checkpoint loops are separated by intermediate recurrent refinement steps. Given a checkpoint interval $c$ with $c \mid K$ ($c$ divides $K$), only a subset $\mathcal{S}$ of loops is directly optimized $\mathcal{S} = \{c, 2c, \ldots, K\}$. For each supervised loop $k \in \mathcal{S}$, the model produces CTC logits $\boldsymbol{\ell}^{(k)} = \psi(\mathbf{z}^{(k)})$, and the training objective is

$$\mathcal{L} = \frac{1}{|\mathcal{S}|} \sum_{k \in \mathcal{S}} \mathcal{L}_{\mathrm{CTC}} \left( \boldsymbol{\ell}^{(k)}, \boldsymbol{y} \right). \tag{3}$$

#### 3.2.3 Prediction Feedback and State Aggregation

At loop $k$, the shared encoder maps the current recurrent state $\mathbf{h}^{(k-1)}$ to an updated representation

$$\mathbf{z}^{(k)} = F_\theta(\mathbf{h}^{(k-1)}). \tag{4}$$

The shared CTC head can then be applied to $\mathbf{z}^{(k)}$ to obtain the posterior $\mathbf{p}^{(k)}$ as defined above, even when loop $k$ is not directly supervised. These posteriors provide a soft token-level hypothesis that is reused as recurrent feedback. We project them back to the hidden space using a learned feedback projection $\rho$ :

$$\mathbf{r}_t^{(k)} = \rho(\mathbf{p}_t^{(k)}) = \mathbf{p}_t^{(k)} \mathbf{W}_\rho. \tag{5}$$

where $\mathbf{W}_\rho \in \mathbb{R}^{|\mathcal{V}| \times d}$. To inject left-context information, LARM applies a one-frame delay along the time axis:

$$\bar{\mathbf{r}}_t^{(k)} = \mathbf{r}_{t-1}^{(k)}, \tag{6}$$

with zero padding at $t = 0$. The delayed feedback is then combined with the encoder output and the fixed frontend representation to form an aggregated state:

$$\mathbf{a}^{(k)} = \mathbf{z}^{(k)} + \beta\, \mathbf{h}^{(0)} + \alpha\, \bar{\mathbf{r}}^{(k)}, \tag{7}$$

where $\mathbf{h}^{(0)} = \phi(\boldsymbol{x})$ and $\alpha, \beta$ are learnable scalars. This aggregation lets each loop condition on both the original acoustic representation and a delayed soft prediction signal. Across loops, this propagates token-level continuity through recurrent depth and encourages later predictions to follow more consistent local decoding trajectories.

#### 3.2.4 Clock and Depth Conditioning

Sparse supervision gives LARM a periodic loop structure, while repeated shared computation also requires the model to distinguish early from late loops. LARM therefore conditions each recurrent update using two complementary signals, the supervision clock embedding and FiLM depth conditioning.

**Supervision clock embedding.** The supervision clock encodes the position of the current loop within the checkpoint interval $c$. For loop $k = 1, \ldots, K$, LARM selects

$$\mathbf{e}_{\mathrm{clock}}^{(k)} = \mathbf{W}_c[(k-1) \bmod c], \qquad \mathbf{W}_c \in \mathbb{R}^{c \times d}. \tag{8}$$

The embedding is broadcast across time and added to the aggregated state:

$$\hat{\mathbf{a}}_t^{(k)} = \mathbf{a}_t^{(k)} + \mathbf{e}_{\mathrm{clock}}^{(k)}. \tag{9}$$

**FiLM depth conditioning.** In addition, LARM uses FiLM modulation to condition the recurrent update on absolute loop depth. We define $\bar{d}(k) = \frac{k-1}{K-1}$, and compute the next recurrent state as

$$\mathbf{h}^{(k)} = \boldsymbol{\gamma}_{\mathrm{film}}(\bar{d}(k)) \odot \hat{\mathbf{a}}^{(k)} + \boldsymbol{\beta}_{\mathrm{film}}(\bar{d}(k)), \tag{10}$$

where $\boldsymbol{\gamma}_{\mathrm{film}}, \boldsymbol{\beta}_{\mathrm{film}} : \mathbb{R} \to \mathbb{R}^d$ are small MLPs that produce feature-wise scale and bias terms. This modulation lets the shared encoder implement different transformations at different loop depths without untying its parameters across loops. The clock embedding makes the sparse supervision cycle explicit, while FiLM depth conditioning allows the shared encoder to specialize across early and late loops without untying its parameters.

#### 3.2.5 Overall Recurrent Procedure

LARM applies the shared encoder for $K$ loops starting from the frontend representation $\mathbf{h}^{(0)} = \phi(\boldsymbol{x})$. At loop $k$, the shared encoder produces an updated representation $\mathbf{z}^{(k)}$ from the previous recurrent state $\mathbf{h}^{(k-1)}$. The CTC prediction head computes frame-level token posteriors, which are projected back to the hidden space, shifted by one frame to form delayed prediction feedback, and aggregated with both the encoder output and the initial acoustic representation. The aggregated state is then augmented with the supervision clock embedding and modulated by FiLM depth conditioning to produce the next recurrent state $\mathbf{h}^{(k)}$.

During training, CTC supervision is applied only at the loops in $\mathcal{S}$, defining a sequence of supervised recognition checkpoints separated by intermediate refinement loops that are not directly supervised. During inference, the CTC head can be applied at any loop, including loops between supervised checkpoints. Overall, LARM forms a shared-parameter recurrent encoder in which sparse supervision structures the loop, delayed prediction feedback propagates token-level context across recurrent depth, and loop conditioning encourages successive loops to take distinct computational roles.

## 4 Experimental Results

### 4.1 Experimental Setup

**Data.** We evaluate on the LibriSpeech benchmark (Panayotov et al., 2015). Ablation experiments are conducted on the 100 h clean training subset, and scaling experiments use the full 960 h training set. Models are evaluated on `test-clean`, with `test-other` additionally reported for the main LibriSpeech results. Additional preprocessing details are provided in Appendix B.

**Architecture.** LARM uses a convolutional subsampling frontend followed by a shared Transformer encoder. The default configuration uses $N = 4$ shared Transformer blocks, model dimension $d = 384$, and fixed per-head dimension $d_h = 64$, so the number of attention heads scales with width as $H = d/d_h$. The recurrent loop is run for $K = 12$ steps with checkpoint interval $c = 4$, yielding three supervised recognition checkpoints. Prediction feedback uses the previous-frame mode, and the mixing scalars $\alpha$ and $\beta$ are learned. Additional architecture details are provided in Appendix B.1.

**Training and Decoding.** Models are trained with AdamW (Loshchilov & Hutter, 2019), gradient clipping, and a cosine learning-rate schedule with linear warmup. Most models are trained for 50 epochs, with extended-budget runs noted separately. We use light SpecAugment (Park et al., 2019) during training. Additional optimization and augmentation details are provided in Appendix B.2. WER is reported using greedy CTC decoding unless specified otherwise. For the main LibriSpeech benchmark results, we also report decoding with a 4-gram KenLM model (Heafield, 2011) using beam search (beam width 100, LM weight 0.5, word insertion bonus 1.0).

**Baselines.** All baselines use the same character vocabulary, frontend, augmentation, and decoding setup as LARM. In addition to fixed-depth Transformer encoders with 4, 16, and 48 blocks, we reimplement self-conditioned CTC (Nozaki & Komatsu, 2021) and intermediate CTC (Lee & Watanabe, 2021) on a 16-block encoder, the self-conditioned folded encoder (Komatsu, 2022) with $R = 6$ and $R = 12$ recursions of a 4-block stack, and Align-Refine (Chi et al., 2021) at our model width. For each baseline, we evaluated several learning rates and report the best setting. Further details are given in Appendix B.3.

**Significance and cost.** Following common ASR practice, we train one model per configuration. We assess WER differences with matched-pair bootstrap tests over test utterances (10,000 resamples) (Bisani & Ney, 2004); the p-values are listed in Appendix C.3.

### 4.2 LibriSpeech Benchmark Results

We compare the default LARM configuration (Section 4.1; shaded rows in Table 1) with three fixed-depth encoders: the same 4-block stack run once (matched parameters), a 16-block encoder (a conventional deeper encoder), and a 48-block encoder (matched block passes).

**Recurrence versus stacking.** At 100h, looping the 4-block stack reduces greedy WER from 26.78 to 11.34, and LARM outperforms both the 16-block and the 48-block encoders with $3.8\times$ and $11\times$ fewer

Table 1: LibriSpeech WER (%) on `test-clean/other` with greedy CTC decoding and with a 4-gram language model. #Passes is the number of Transformer-block applications per forward pass.

| Data | Model | #Params | #Passes | Greedy CTC | | + 4-gram LM | |
|---|---|---|---|---|---|---|---|
| | | | | `test-clean` | `test-other` | `test-clean` | `test-other` |
| **100h** | Standard encoder, 4 blocks | 7.6M | 4 | 26.78 | 52.82 | 13.44 | 36.60 |
| | Standard encoder, 16 blocks | 28.9M | 16 | 14.43 | 37.23 | 9.97 | 28.68 |
| | Standard encoder, 48 blocks | 85.7M | 48 | 12.24 | 34.06 | 9.03 | 27.07 |
| | Intermediate CTC (2021), 16 blocks | 28.9M | 16 | 12.62 | 34.47 | 9.20 | 27.54 |
| | Self-conditioned CTC (2021), 16 blocks | 28.9M | 16 | 12.32 | 34.10 | 8.98 | 27.28 |
| | Align-Refine (2021), 12 enc. + 6 ref. layers | 36.0M | - | 12.11 | 33.35 | 9.35 | 27.38 |
| | Folded encoder (2022), 4 blocks, $R = 6$ | 7.65M | 24 | 13.78 | 35.97 | 9.46 | 28.18 |
| | Folded encoder (2022), 4 blocks, $R = 12$ | 7.65M | 48 | 12.31 | 33.30 | 8.68 | 26.51 |
| | **LARM** (4 blocks, $K = 12$) | 7.7M | 48 | **11.34** | **31.84** | **8.66** | **26.28** |
| **960h** | Standard encoder, 4 blocks | 7.6M | 4 | 14.07 | 29.82 | 6.77 | 17.39 |
| | Standard encoder, 16 blocks | 28.9M | 16 | 4.79 | 13.26 | 3.51 | 9.87 |
| | Standard encoder, 48 blocks | 85.7M | 48 | **3.87** | **10.58** | **3.20** | **8.56** |
| | **LARM** (4 blocks, $K = 12$) | 7.7M | 48 | 4.59 | 11.75 | 3.51 | 9.38 |
| **960h** | Conformer (2020), 4 blocks | 14.25M | 4 | 5.48 | 15.01 | 3.76 | 10.74 |
| | **LARM-Conformer** (4 blocks, $K = 12$) | 14.2M | 48 | **3.66** | **9.71** | **3.12** | **8.42** |

Table 2: Cost of each supervised exit on LibriSpeech 960h, compared with fixed-depth encoders. Exits of the same LARM model share one checkpoint. WER (%) on `test-clean`.

| Model | Exit | #Params | #Passes | GFLOPs | Memory (MiB) | Latency (ms) | Greedy | + LM |
|---|---|---|---|---|---|---|---|---|
| Standard encoder, 4 blocks | - | 7.6M | 4 | 4.58 | 74.7 | 2.39 | 14.07 | 6.77 |
| Standard encoder, 16 blocks | - | 28.9M | 16 | 16.36 | 158.5 | 8.78 | 4.79 | 3.51 |
| Standard encoder, 48 blocks | - | 85.7M | 48 | 47.78 | 379.0 | 24.22 | 3.87 | 3.20 |
| LARM, $d = 384$ | 4 | 7.7M | 16 | 16.40 | 76.3 | 10.40 | 6.15 | 3.94 |
| | 8 | | 32 | 32.19 | 80.9 | 20.44 | 4.73 | 3.52 |
| | 12 | | 48 | 47.92 | 85.3 | 30.47 | 4.59 | 3.51 |
| LARM, $d = 768$ | 4 | 29.5M | 16 | 60.72 | 174.6 | 11.56 | 4.18 | 3.19 |
| | 8 | | 32 | 120.55 | 183.5 | 24.28 | 3.46 | 2.91 |
| | 12 | | 48 | 180.39 | 192.4 | 35.96 | 3.45 | 2.93 |
| LARM, $d = 1024$ | 4 | 51.9M | 16 | 105.99 | 268.1 | 16.60 | 3.94 | 3.01 |
| | 8 | | 32 | 210.94 | 279.9 | 32.20 | 3.39 | 2.81 |
| | 12 | | 48 | 315.88 | 291.7 | 47.87 | 3.39 | 2.83 |

parameters. Shared recurrent depth is thus a more parameter-efficient way to add depth than stacking unshared layers. At 960h, LARM still outperforms the 16-block encoder under greedy decoding. With LM decoding, it matches it on `test-clean` ($p = 0.97$) and improves on `test-other` ($p = 0.001$). The 48-block encoder is more accurate than this reference LARM at 960h. However, a wider LARM outperforms it. Indeed the $d = 768$ model stopped at exit 8 reaches 3.46 vs. 3.87 (greedy) and 2.91 vs. 3.20 (LM) at 24.3 vs. 24.2 ms, with $3\times$ fewer parameters and half the peak memory (Table 2). This illustrates a key benefit of looping. Because depth is decoupled from parameter count, width, loop budget, and exit can be chosen separately to balance accuracy, latency, and memory for each deployment regime (Section 4.3).

**Comparison with related methods.** Reimplemented in our pipeline at 100h, no prior method matches LARM on either test set under greedy decoding. The closest, the self-conditioned folded encoder, also reuses a 4-block stack. It reaches 13.78 at its published recursion count ($R = 6$) and 12.31 when matched to LARM in parameters and block passes ($R = 12$), against 11.34 for LARM. Under LM decoding, the two are statistically tied. Self-conditioned CTC, intermediate CTC, and Align-Refine (at its published depth) use 3.8–4.7$\times$ more parameters and remain behind LARM. Since the $R = 12$ folded encoder, Align-Refine, and the 48-block encoder all cost about the same as LARM (48.0, 45.5, and 47.8 vs. 47.9 GFLOPs for 10 s of audio), LARM is also the most accurate system at matched FLOPs. Note that the absolute WERs published for these methods are not comparable with ours, as they use subword vocabularies, larger models, and different training recipes.

**Conformer backbone.** The LARM design also transfers to a Conformer backbone (Gulati et al., 2020). We replace the shared Transformer stack by a Conformer stack and keep all loop mechanisms unchanged. At 960h, LARM-Conformer reduces the `test-clean` greedy WER of a parameter-matched 4-block Conformer from 5.48 to 3.66 (95% confidence interval of the difference $[1.62, 2.03]$, $p < 0.0001$). At 100h, it reaches 11.27, against 17.19 for the 4-block Conformer and 15.96 for a 16-block Conformer ($d = 144$, 10M parameters).

Table 3: Scaling LARM with model width on LibriSpeech (WER %). Bold marks the best WER per training-data block.

| Training data | Model | #Params | Greedy CTC | | + 4-gram LM | |
|---|---|---|---|---|---|---|
| | | | `test-clean` | `test-other` | `test-clean` | `test-other` |
| **100h** | Standard encoder, 48 blocks | 85.7M | 12.24 | 34.06 | 9.03 | 27.07 |
| | LARM ($K = 12$, $d = 384$) | 7.7M | 11.34 | 31.84 | 8.66 | 26.28 |
| | LARM ($K = 12$, $d = 768$) | 29.5M | 9.58 | 28.25 | 7.57 | 23.89 |
| | LARM ($K = 12$, $d = 1024$) | 51.9M | **8.95** | **27.93** | **7.22** | **23.79** |
| **960h** | Standard encoder, 48 blocks | 85.7M | 3.87 | 10.58 | 3.20 | 8.56 |
| | LARM ($K = 12$, $d = 384$) | 7.7M | 4.59 | 11.75 | 3.51 | 9.38 |
| | LARM ($K = 12$, $d = 768$) | 29.5M | 3.45 | 9.44 | 2.93 | 7.93 |
| | LARM ($K = 12$, $d = 1024$) | 51.9M | **3.39** | **9.01** | **2.83** | **7.67** |

### 4.3 Budget-Adaptive Inference

We show that a single trained LARM model can operate at several inference budgets, with fixed parameters and only the number of executed loops varying. Four properties support this.

**Accuracy improves with more loops.** In Figure 1, WER decreases from one supervised checkpoint to the next in both settings, although intermediate loops are non-monotonic at 100h and smooth at 960h. Within the trained budget, additional loops are thus converted into better recognition without changing the parameters. Beyond it, loops saturate. Running the 960h $d = 384$, $K = 12$ model for 16 loops leaves WER essentially unchanged (3.51 to 3.48 on `test-clean`), and further gains instead come from increasing the width $d$ (Section 4.4 and Appendix C.2).

**Early exits are nearly as accurate as the final exit.** For the 100h $d = 384$, $K = 16$ model, the third checkpoint is within 0.06 WER points of the final one and the second within 0.70 (Appendix C.1). At 960h, exit 8 of the wider models already matches exit 12 (3.46 vs. 3.45 and 3.39 vs. 3.39, Table 2). LARM can therefore be stopped early, with little loss in accuracy, when lower latency is required. A single model trained with a larger budget thus also covers smaller ones, and its intermediate exits often even outperform models trained directly with that budget (Appendix C.1).

**Exits trade compute at nearly constant memory.** Compute grows linearly with the number of executed loops, whereas peak memory grows by only about 10% from exit 4 to exit 12 (Table 2). At approximately matched FLOPs, the 48-block encoder is more accurate than the final LARM exit (3.87 vs. 4.59), but it needs $11\times$ the parameters and $4.4\times$ the peak memory. LARM also outperforms the 16-block encoder (4.59 vs. 4.79) with about half its peak memory (85.3 vs. 158.5 MiB).

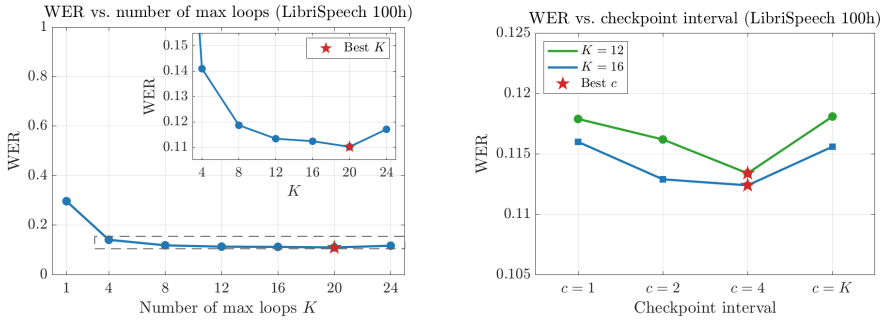
**Under LM decoding, earlier exits can be best.** For the reference $K = 12$ model, the best LM-decoded exit is loop 7 at both 100h and 960h. At 960h, later loops stay statistically tied with it, whereas at 100h it is significantly better than the final loop, even with per-exit LM tuning (Appendix C.1). As loops progress, the encoder's own greedy predictions keep improving leaving less for the external LM to correct. The exit can thus be chosen on development data for each decoding setup.

### 4.4 LARM Scales with Model Size and Data

Table 3 varies the model width $d$ at 100h and 960h, with the 48-block encoder as a reference matched in block passes ($N \times K = 4 \times 12$). Width consistently improves LARM. At 100h, going from $d = 384$ to $d = 1024$ reduces greedy WER from 11.34 to 8.95 on `test-clean` and from 31.84 to 27.93 on `test-other`, with the same trend under LM decoding, and every wider model outperforms the 48-block encoder. More data yields an even larger gain. The $d = 384$ model improves from 11.34 to 4.59 on `test-clean` and from 31.84 to 11.75 on `test-other`, and at 960h the $d = 1024$ model reaches 3.39 and 9.01. Width gains shrink at this scale ($d = 768$ to $d = 1024$ gives 3.45 to 3.39), suggesting saturation with respect to width alone. Longer training, larger loop budgets, and deeper shared blocks bring further gains, down to 3.08/8.29 greedy and 2.73/7.27 LM-decoded WER at 960h (Appendix C.2).

### 4.5 Ablation Study

We ablate the design choices of LARM on LibriSpeech 100h ($d = 384$, greedy WER on `test-clean`). Unless stated otherwise, the reference model uses $K = 12$, $c = 4$, FiLM depth conditioning, previous-frame feedback, and learned $\alpha, \beta$.




(a) Trained loop-budget scaling

(b) Sparse supervision schedule

Figure 3: **Training-time loop-structure ablations** (100h, test-clean). a Effect of the maximum loop budget $K$; each point is a separately trained LARM model ($d = 384$). b Effect of the checkpoint interval $c$: moderately sparse supervision beats both dense and final-only supervision.

Table 4: Ablations on LibriSpeech 100h (greedy CTC WER (%), `test-clean`).

(a) Naive recurrence

| | |
|---|---|
| 4-block encoder | 26.78 |
| Naive looped | 12.70 |
| LARM | **11.34** |

(b) Depth conditioning

| | |
|---|---|
| None | 12.13 |
| Embedding | 11.98 |
| MLP | 11.77 |
| LARM (FiLM) | **11.34** |

(c) Feedback and aggregation

| | |
|---|---|
| Current frame | 11.43 |
| No feedback | 12.31 |
| Fixed $\alpha, \beta$ | 11.91 |
| LARM | **11.34** |

**Naive looping helps, but LARM uses loops better.** Reusing the 4-block stack for $K = 12$ loops without the proposed mechanisms already reduces WER from 26.78 to 12.70 (Table 4a). Among models of about 7.7M parameters with the same 48 block passes, LARM reduces the WER of naive looping by 10.7% relative (11.34), whereas the self-conditioning of the folded encoder improves it by only 3.1% (12.31, Table 1).

**Larger trained budgets improve performance.** Across separately trained models (Figure 3a), WER improves from $K = 1$ to $K = 20$ and saturates at $K = 24$. In most configurations, a larger budget also improves intermediate exits. For example, exit 8 of the $K = 12$ model (11.57) outperforms the final exit of the $K = 8$ model (12.07), suggesting that later checkpoints act as a regularizer on earlier loops (see Appendix C.1).

**Sparse supervision beats dense and final-only supervision.** Varying the checkpoint interval (Figure 3b), $c = 4$ is significantly better than both dense ($c = 1$) and final-only supervision ($p < 0.005$ on both test sets). Forcing every loop to act as a full recognition stage limits latent refinement, while supervising only the last loop loses useful intermediate checkpoints.

**FiLM is the most effective depth conditioning.** Removing depth conditioning raises WER from 11.34 to 12.13 (Table 4b). Additive embeddings (11.98) and MLP conditioning (11.77) recover part of the gap, but FiLM performs best.

**Feedback and learned aggregation matter.** Removing feedback raises WER to 12.31 and fixing $\alpha, \beta$ to 11.91 (Table 4c). Current and previous frame feedback are statistically tied (11.43 vs. 11.34, $p = 0.51$), but this small gap may simply be below what the test set can resolve. We keep previous-frame feedback, as it yielded better results in most of our experiments during model design.

## 5 Conclusion

We introduced LARM, a depth-conditioned looped Transformer that turns recurrent encoder depth into a selectable inference budget for ASR. By combining sparse CTC checkpoints, supervision-clock embeddings, FiLM depth conditioning, and delayed prediction feedback, LARM enables a shared encoder to perform stage-specialized recurrent refinement. Experiments on LibriSpeech show that LARM improves across supervised checkpoints within its trained loop budget and achieves competitive WER against much deeper unshared encoders while using substantially fewer parameters. A single compact checkpoint can therefore serve several compute budgets, making LARM well suited to memory-constrained deployment.

## AI USE STATEMENT

Generative AI tools helped with editing the text for language and clarity and with writing some implementation and analysis scripts. The authors reviewed and verified all such content and code. The scientific findings and conclusions are the authors' own, and the authors are responsible for all text, claims, code, and experiments in this paper.

## REFERENCES

Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed recursive transformers: Effective parameter sharing with layer-wise lora, 2025. URL https://arxiv.org/abs/2410.20672.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Proc. of Advances in Neural Information Processing Systems*, volume 32, 2019.

Maximilian Bisani and Hermann Ney. Bootstrap estimates for confidence intervals in asr performance evaluation. *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1: I–409, 2004. URL https://api.semanticscholar.org/CorpusID:14206767.

William Chan, Chitwan Saharia, Geoffrey Hinton, Mohammad Norouzi, and Navdeep Jaitly. Imputer: Sequence modelling via imputation and dynamic programming. In *Proc. of International Conference on Machine Learning*, pp. 1403–1413. PMLR, 2020.

Ethan A Chi, Julian Salazar, and Katrin Kirchhoff. Align-refine: Non-autoregressive speech recognition via iterative realignment. In *Proc. of North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1920–1927, 2021.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout, 2019. URL https://arxiv.org/abs/1909.11556.

Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *Proc. of International Conference on Machine Learning*, pp. 11398–11442, 2023.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. *arXiv preprint arXiv:2310.02226*, 2023.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning (ICML '06)*, pp. 369–376, New York, NY, USA, 2006. ACM. doi: 10.1145/1143844.1143891. URL https://doi.org.

Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, and Ruoming Pang. Conformer: Convolution-augmented transformer for speech recognition, 2020. URL https://arxiv.org/abs/2005.08100.

Kenneth Heafield. KenLM: Faster and smaller language model queries. In *Proc. of the Sixth Workshop on Statistical Machine Translation*, pp. 187–197, July 2011.

Yosuke Higuchi, Shinji Watanabe, Nanxin Chen, Tetsuji Ogawa, and Tetsunori Kobayashi. Mask CTC: Non-Autoregressive End-to-End ASR with CTC and Mask Predict. In *Proc. of Interspeech 2020*, pp. 3655–3659, 2020. doi: 10.21437/Interspeech.2020-2404.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. OpenAI o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Min Ho Jang, Eun Seo Seo, Jin Young Kim, Hyeongsoo Lim, and Ji Won Yoon. Bicycle: group-wise recursive transformer based on asr mechanism. In *Proceedings of the Fortieth AAAI Conference on Artificial Intelligence and Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence and Sixteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'26/IAAI'26/EAAI'26. AAAI Press, 2026. ISBN 978-1-57735-906-7. doi: 10.1609/aaai.v40i37.40386. URL https://doi.org/10.1609/aaai.v40i37.40386.

Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks. *arXiv preprint arXiv:2510.04871*, 2025.

Tatsuya Komatsu. Non-autoregressive ASR with self-conditioned folded encoders. In *Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 7427–7431, 2022. doi: 10.1109/ICASSP43922.2022.9746770.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

Jaesong Lee and Shinji Watanabe. Intermediate loss regularization for ctc-based speech recognition, 2021. URL https://arxiv.org/abs/2102.03216.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *Proc. of International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-Refine: Iterative refinement with self-feedback. In *Proc. of Advances in neural information processing systems*, volume 36, pp. 46534–46594, 2023.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. s1: Simple test-time scaling. In *Proc. of Conference on Empirical Methods in Natural Language Processing*, pp. 20286–20332, 2025.

Jumon Nozaki and Tatsuya Komatsu. Relaxing the conditional independence assumption of CTC-based ASR by conditioning on intermediate predictions. *arXiv preprint arXiv:2104.02724*, 2021.

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

Daniel S. Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D. Cubuk, and Quoc V. Le. SpecAugment: A simple data augmentation method for automatic speech recognition. In *Proc. of Interspeech 2019*, pp. 2613–2617, 2019. doi: 10.21437/Interspeech.2019-2680.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proc. of the AAAI conference on artificial intelligence*, volume 32, 2018.

Jorge Pérez, Javier Marinković, and Pablo Barceló. On the turing completeness of modern neural network architectures. *arXiv preprint arXiv:1901.03429*, 2019.

Jacob Pfau, William Merrill, and Samuel R Bowman. Let's think dot by dot: Hidden computation in transformer language models. *arXiv preprint arXiv:2404.15758*, 2024.

Ramon Sanabria and Florian Metze. Hierarchical multitask learning with CTC. In *Proc. of IEEE Spoken Language Technology Workshop (SLT)*, pp. 485–490. IEEE, 2018.

Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks. In *Proc. of Advances in Neural Information Processing Systems*, volume 34, pp. 6695–6706, 2021.

Yangyang Shi, Varun Nagaraja, Chunyang Wu, Jay Mahadeokar, Duc Le, Rohit Prabhavalkar, Alex Xiao, Ching-Feng Yeh, Julian Chan, Christian Fuegen, Ozlem Kalinli, and Michael L. Seltzer. Dynamic encoder transducer: A flexible solution for trading off accuracy for latency, 2021. URL https://arxiv.org/abs/2104.02176.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model. *arXiv preprint arXiv:2506.21734*, 2025.

Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D Goodman. Quiet-STaR:: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*, 2024.

Y. Zhao, Chongjia Ni, C. C. Leung, Shafiq R. Joty, Chng Eng Siong, and Bin Ma. Universal speech transformer. In *Interspeech*, 2020. URL https://api.semanticscholar.org/CorpusID:226203241.

# A Additional Architectural Details

## A.1 CTC Prediction Head

The CTC prediction head $\psi$ is implemented as a linear projection from the hidden dimension $d$ to the output vocabulary $\mathcal{V}$:

$$\boldsymbol{\ell}^{(k)} = \psi(\mathbf{z}^{(k)}) = \mathbf{z}^{(k)}\mathbf{W}_{\psi}^{\top} + \mathbf{b}_{\psi}, \qquad \mathbf{W}_{\psi} \in \mathbb{R}^{|\mathcal{V}|\times d}. \tag{11}$$

Here $\boldsymbol{\ell}^{(k)} \in \mathbb{R}^{T'\times|\mathcal{V}|}$, and $\mathcal{V}$ includes the CTC blank symbol. The corresponding token posterior is

$$\mathbf{p}^{(k)} = \mathrm{softmax}\Big(\boldsymbol{\ell}^{(k)}\Big). \tag{12}$$

## A.2 Shared Encoder Details

The shared encoder $F_\theta$ consists of $N$ stacked pre-norm Transformer blocks. Given an input representation $\mathbf{z}$, each block applies multi-head self-attention (MHSA) followed by a feed-forward network (FFN), both with residual connections:

$$\mathbf{z}' = \mathbf{z} + \mathrm{MHSA}(\mathrm{LN}(\mathbf{z})), \qquad \mathbf{z}'' = \mathbf{z}' + \mathrm{FFN}(\mathrm{LN}(\mathbf{z}')). \tag{13}$$

MHSA uses $H$ attention heads with per-head dimension $d_h = d/H$. Queries, keys, and values are produced using a fused projection,

$$[\mathbf{Q}, \mathbf{K}, \mathbf{V}] = \mathbf{z}\mathbf{W}_{QKV}, \qquad \mathbf{W}_{QKV} \in \mathbb{R}^{d\times 3d}, \tag{14}$$

followed by splitting and reshaping across heads. Rotary positional embeddings (RoPE) are applied to $\mathbf{Q}$ and $\mathbf{K}$ before scaled dot-product attention:

$$\mathrm{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\top}}{\sqrt{d_h}}\right)\mathbf{V}. \tag{15}$$

The FFN is a two-layer MLP with GELU activation:

$$\mathrm{FFN}(\mathbf{z}) = \mathbf{W}_2\,\mathrm{GELU}(\mathbf{W}_1\mathbf{z} + \mathbf{b}_1) + \mathbf{b}_2, \tag{16}$$

where $\mathbf{W}_1 \in \mathbb{R}^{4d\times d}$ and $\mathbf{W}_2 \in \mathbb{R}^{d\times 4d}$.

The encoder $F_\theta$, CTC prediction head $\psi$, supervision clock embedding projection $\mathbf{W}_c$, and FiLM conditioning models are shared across all loops.

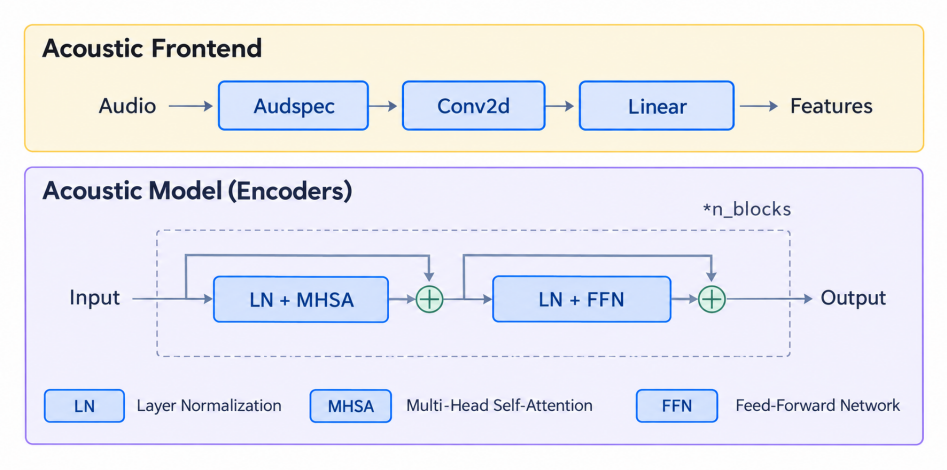



Figure 4: **Acoustic frontend and encoder block.** The frontend converts audio features into latent representations, while the encoder applies stacked pre-norm Transformer blocks with MHSA and FFN modules.

## B Additional Experimental Setup

**Input features.** Audio is resampled to 16 kHz and represented using 80-dimensional log-Mel filterbank features, extracted with a 25 ms window and a 10 ms hop.

**Output vocabulary.** We use a character-level CTC vocabulary with 30 symbols: the 26 lower-case English letters, an apostrophe, a word-boundary symbol, a `<unk>` symbol, and the CTC blank symbol.

### B.1 Experimental Architecture Details

The convolutional frontend uses two strided 2D convolutional layers with 64 channels, kernel size $3 \times 3$, stride 2, padding 1, and SiLU activations. This yields a $4\times$ time downsampling and reduces the frequency axis from 80 to 20 bins. The flattened output ($64 \times 20 = 1280$ dimensions) is projected linearly to the model dimension $d$, followed by an initial dropout of $0.1$.

The shared encoder consists of $N = 4$ Transformer blocks with rotary positional embeddings (RoPE, base $10{,}000$) and FFN expansion ratio 4. All configurations use a fixed per-head dimension $d_h = 64$, so the number of attention heads is $H = d/d_h$. We experiment with model widths $d \in \{384, 768, 1024, 1280\}$, corresponding to $H \in \{6, 12, 16, 20\}$ heads and FFN hidden dimensions $\{1536, 3072, 4096, 5120\}$.

Unless otherwise stated, the LARM loop runs for $K = 12$ steps with checkpoint interval $c = 4$, yielding $|\mathcal{S}| = 3$ supervised checkpoints. The FiLM depth-conditioning network uses hidden dimension 64. Prediction feedback is computed in previous-frame mode. The mixing scalars $\alpha$ and $\beta$ are initialized to $0.5$ and learned jointly.

### B.2 Training Details

All models are optimized with AdamW (Loshchilov & Hutter, 2019) using $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$, and weight decay $5 \times 10^{-3}$. Gradient norms are clipped to 1.0. The learning rate follows a cosine decay schedule with linear warmup, peak learning rate $\eta = 7 \times 10^{-4}$, and minimum learning rate $0.03\,\eta$. The warmup lasts 1,000 steps for 100 h experiments and 10,000 steps for 960 h experiments.

Most models are trained for 50 epochs, with extended-budget runs noted separately. The effective batch size is 32 for 100 h experiments and for the main 960 h scaling runs, using gradient accumulation when needed.

SpecAugment (Park et al., 2019) is applied throughout training with a light policy: one frequency mask with $F_{\max} = 15$ and two time masks with time-mask ratio $p_T = 0.02$. Masked regions are filled with the utterance mean.

### B.3 Baseline Details

All baselines are trained and decoded with the same character vocabulary, frontend, augmentation, and decoding setup as LARM, and we report the best of several tested learning rates for each. Self-conditioned CTC and intermediate CTC are applied to the 16-block unshared encoder. The self-conditioned folded encoder reuses a 4-block stack for $R$ recursions; $R = 6$ corresponds to the setting emphasized in the original work, and $R = 12$ matches LARM in both parameter count and block passes. For Align-Refine, we sweep the number of encoder and refiner layers at our width ($d = 384$), up to the published depth of 12 encoder and 6 refiner layers (Table 5). Absolute results are not directly comparable with the original papers, which use subword vocabularies, Conformer blocks, different frontends, or larger widths.

Table 5: Align-Refine configurations on LibriSpeech 100h, WER (%). Only the largest configuration was decoded on all conditions.

| | | **Greedy CTC** | | **+ 4-gram LM** | |
|---|---|---|---|---|---|
| **Configuration** | **#Params** | `test-clean` | `test-other` | `test-clean` | `test-other` |
| 2 encoder + 2 refiner layers | 8.83M | 19.44 | – | – | – |
| 6 encoder + 2 refiner layers | 15.9M | 14.51 | – | – | – |
| 12 encoder + 6 refiner layers | 36.0M | 12.11 | 33.35 | 9.35 | 27.38 |
| LARM ($K = 12$, $d = 384$) | 7.7M | **11.34** | **31.84** | **8.66** | **26.28** |

## C Complementary Experimental Results

### C.1 Detailed Results Across Loop Exits

Table 6 reports WER at supervised loop exits for LARM models trained with different maximum loop budgets on LibriSpeech 100h. All models use checkpoint interval $c = 4$, so supervised exits occur every four recurrent loops. These results complement Section 4.3 by showing how recognition quality evolves across loop exits for several trained loop budgets.

Across $K = 4$ to $K = 20$, greedy WER generally improves from one supervised checkpoint to the next, supporting the interpretation of additional loops as useful recurrent refinement steps. The $K = 24$ model is an exception: although later checkpoints still improve over early ones, its final WER is worse than shorter-loop models, consistent with the saturation behavior observed in Figure 3.

The table also shows that training with a larger maximum loop budget can improve intermediate exits. For example, the 8-loop exit of the $K = 12$ model improves over the final exit of the $K = 8$ model, and the 16-loop exit of the $K = 20$ model improves over the final exit of the $K = 16$ model. This suggests that later supervised checkpoints can regularize earlier recurrent states, although the effect weakens once the loop budget becomes too large.

Language-model-assisted WER is less monotonic than greedy WER. In several runs, the best LM-decoded WER occurs before the final loop exit. Retuning the LM weight and insertion bonus separately for each exit improves absolute WER at every exit but leaves this ordering unchanged: for the 100h $K = 12$ model, LM-decoded `test-clean` WER is 8.56, 8.08, 8.11, and 8.35 at loops 4, 7, 8, and 12, and loops 7 and 8 are significantly better than loop 12 ($p < 0.0001$). At 960h, the effect vanishes and LM-decoded WER plateaus at 3.50, 3.52, and 3.51 at loops 7, 8, and 12 ($p = 0.90$ and $p = 0.68$ against loop 12). We attribute this to later loops producing sharper posteriors, which leave less for the external LM to correct, while the greedy predictions of the encoder itself improve across supervised exits. We therefore use greedy decoding as the primary diagnostic for recurrent refinement and report LM results for completeness.

Table 6: Detailed loop-exit behavior on LibriSpeech 100h with checkpoint interval $c = 4$. WER (%) is reported on `test-clean`. Greedy WER is shown at supervised loop exits. LM columns report the best LM-decoded WER across all loop exits and the final-loop LM WER.

| **Max loops $K$** | **Greedy WER at supervised loop exits** | | | | | | **Best LM** | **Final LM** |
|---|---|---|---|---|---|---|---|---|
| | **4** | **8** | **12** | **16** | **20** | **24** | | |
| 4 | 14.07 | – | – | – | – | – | 9.11 (4) | 9.11 |
| 8 | 14.22 | 12.07 | – | – | – | – | 8.67 (8) | 8.67 |
| 12 | 15.20 | 11.57 | 11.34 | – | – | – | 8.38 (7) | 8.66 |
| 16 | 16.12 | 11.94 | 11.30 | 11.24 | – | – | 8.28 (7) | 8.66 |
| 20 | 16.95 | 11.83 | 11.17 | 11.05 | 11.02 | – | 8.03 (7) | 8.43 |
| 24 | 20.05 | 14.33 | 13.52 | 13.39 | 13.31 | 13.34 | 9.02 (11) | 9.45 |

### C.2 Extended Scaling Results

Table 7 reports additional larger-budget runs on LibriSpeech 100h and 960h. These experiments are not intended as a single controlled ablation, since they vary several axes across runs, but they show the headroom available when increasing optimization budget, recurrent loop budget, model width, or the number of shared Transformer blocks. All results are reported on `test-clean`.

The clearest improvements come from combining longer training with a larger loop budget. Compared with the main $d = 768$, $K = 12$, 50-epoch result in Table 3 (9.58 greedy WER and 7.57 with LM), the $d = 768$, $K = 16$, 100-epoch run improves to 8.39 greedy WER and 7.11 with LM. Similarly, compared with the main $d = 1024$, $K = 12$, 50-epoch result (8.95 greedy WER and 7.22 with LM), the $d = 1024$, $K = 16$, 100-epoch run reaches 8.27 greedy WER and 6.95 with LM. These results suggest that the main-table configurations do not saturate the potential of LARM, and that larger recurrent budgets and longer optimization remain beneficial.

Increasing width alone shows weaker returns at this scale. The $d = 1280$, $K = 12$, 50-epoch run reaches 8.92 greedy WER, only slightly better than the main $d = 1024$, $K = 12$ result, and its LM-assisted WER is slightly worse. This is consistent with the saturation trend observed in Section 4.4: once the recurrent configuration is fixed, simply increasing width provides diminishing returns. In contrast, deeper shared encoders at small width ($N = 6, K = 8$ and $N = 8, K = 6$) improve over the default small $d = 384$ model from Table 3, but remain behind the wider and longer-trained LARM variants. This suggests that additional block depth helps, but is less effective than jointly increasing width, recurrent budget, and optimization time.

At 960h, the same trend holds (bottom of Table 7). Increasing the checkpoint interval to $c = 6$ improves the $d = 1024$ model from 3.39 to 3.22 greedy WER at 50 epochs, and training for 100 epochs reaches 3.08 greedy WER on `test-clean` and 8.29 on `test-other` (2.73 and 7.27 with LM decoding), the best results we obtained. As at 100h, these runs vary several factors (checkpoint interval, training length, and warmup) and do not form a controlled ablation.

LM-assisted decoding is again less monotonic across loop exits. Several extended runs obtain their best LM WER before the final loop, even when greedy WER is best at the final loop. We therefore report both the final LM result and the best LM result across loop exits. As discussed in Appendix C.1, we interpret greedy WER as the cleaner diagnostic of recurrent acoustic refinement, because the interaction between LM output and the intermediate hypothesis is complex and not explicitly controlled.

### C.3 Statistical Significance

Table 8 lists the matched-pair bootstrap tests supporting the comparisons in the main text (Bisani & Ney, 2004). Each test uses 10,000 joint resamples of the test utterances and reports the p-value of the WER difference between the two systems.

Table 7: Extended LARM scaling results on LibriSpeech 100h and 960h `test-clean`. WER (%) is reported with greedy CTC decoding and with a 4-gram language model. $N$ is the number of shared Transformer blocks, $K$ is the maximum loop budget, and $c$ is the checkpoint interval. Numbers in parentheses give the loop at which the best WER is obtained; –: per-loop LM decoding not evaluated.

| $d$ | $N$ | $K$ | $c$ | **Epochs** | **Final greedy** | **Best greedy** | **Final LM** | **Best LM** |
|---|---|---|---|---|---|---|---|---|
| *LibriSpeech 100h* | | | | | | | | |
| 768 | 4 | 16 | 4 | 100 | 8.39 | 8.39 (16) | 7.11 | **6.83** (9) |
| 768 | 4 | 16 | 8 | 100 | 8.50 | 8.50 (16) | 7.12 | 7.12 (16) |
| 1024 | 4 | 16 | 4 | 100 | **8.27** | **8.27** (16) | **6.95** | 6.93 (12) |
| 1280 | 4 | 12 | 4 | 50 | 8.92 | 8.92 (12) | 7.38 | 7.19 (8) |
| 384 | 6 | 8 | 4 | 50 | 10.86 | 10.86 (8) | 8.25 | 8.25 (7–8) |
| 384 | 8 | 6 | 3 | 50 | 10.85 | 10.85 (6) | 8.34 | 8.33 (4–5) |
| 384 | 4 | 12 | 4 | 50 | 11.34 | 11.34 (12) | 8.66 | 8.38 (7) |
| *LibriSpeech 960h* | | | | | | | | |
| 768 | 4 | 12 | 4 | 50 | 3.45 | 3.45 (12) | 2.93 | 2.91 (8) |
| 768 | 4 | 12 | 6 | 100 | 3.15 | 3.15 (12) | 2.74 | – |
| 1024 | 4 | 12 | 4 | 50 | 3.39 | 3.39 (8) | 2.83 | 2.81 (8) |
| 1024 | 4 | 12 | 6 | 50 | 3.22 | 3.22 (12) | **2.72** | **2.72** (12) |
| 1024 | 4 | 12 | 6 | 100 | **3.08** | **3.08** (12) | 2.73 | – |
| 384 | 4 | 12 | 4 | 50 | 4.59 | 4.59 (12) | 3.51 | 3.50 (7) |

Table 8: Matched-pair bootstrap p-values (10,000 resamples); – means not evaluated.

| **Comparison** | **Setting** | `test-clean` | `test-other` |
|---|---|---|---|
| *LARM vs. baselines* | | | |
| 16-block encoder | 960h, greedy | 0.0026 | $< 0.0001$ |
| 16-block encoder | 960h, LM | 0.97 (tie) | 0.001 |
| Self-conditioned CTC | 100h, greedy | $< 0.0001$ | $< 0.0001$ |
| Self-conditioned CTC | 100h, LM | 0.016 | $< 0.0001$ |
| Intermediate CTC | 100h, greedy | $< 0.0001$ | $< 0.0001$ |
| Intermediate CTC | 100h, LM | 0.0002 | $< 0.0001$ |
| Align-Refine | 100h, greedy and LM | $< 0.0001$ | $< 0.0001$ |
| Folded encoder, $R = 6$ | 100h, greedy and LM | $< 0.0001$ | $< 0.0001$ |
| Folded encoder, $R = 12$ | 100h, greedy | $< 0.0001$ | $< 0.0001$ |
| Folded encoder, $R = 12$ | 100h, LM | 0.85 (tie) | 0.30 (tie) |
| Conformer, 4 blocks (vs. LARM-Conformer) | 960h, greedy | $< 0.0001$ | $< 0.0001$ |
| *Ablations (100h, greedy, vs. LARM with $c = 4$)* | | | |
| Dense supervision ($c = 1$) | | 0.0048 | $< 0.0001$ |
| Final-only supervision | | $< 0.0001$ | $< 0.0001$ |
| Current-frame feedback | | 0.51 | 0.48 |
| *LM decoding across exits (vs. final loop)* | | | |
| Loops 7 and 8 | 960h | 0.90, 0.68 | – |
| Loops 7 and 8 (LM retuned per exit) | 100h | $< 0.0001$ | $< 0.0001$ |

## D Broader Impact

This work studies budget-adaptive inference for ASR through recurrent encoder depth. The main broader impact of LARM is that it changes how ASR systems can be deployed under variable compute constraints. Instead of training and maintaining separate shallow and deep encoders for different

latency or accuracy targets, a single shared-parameter model can expose multiple operating points by changing the number of recurrent loops at inference. This may be useful in settings where the available compute budget changes across devices, users, or requests, while keeping the stored model size fixed.

This also provides a different form of parameter efficiency from standard model compression. LARM does not only reduce parameters relative to deep unshared encoders; it makes additional computation available as a runtime decision. Such a mechanism could support ASR systems that spend more computation on difficult utterances and less on easier ones, although the present work uses fixed loop budgets rather than a learned adaptive policy.

The main deployment cost is that the more accurate operating points require more inference computation, so LARM trades stored parameters for latency and energy. This trade-off is central to the method: the same mechanism that enables budget-adaptive inference also requires careful selection of the exit in practical systems. Future adaptive early-exit mechanisms could make this trade-off more efficient by allocating recurrent computation only when it improves recognition.

## E LIMITATIONS

The evaluation in this work is mainly limited to LibriSpeech. While this provides a controlled benchmark for studying recurrent encoder depth, it does not fully characterize the behavior of LARM on conversational speech, accented speech, multilingual data, noisy recordings, far-field audio, or overlapping speakers.

LARM currently uses fixed loop budgets. Although sparse CTC checkpoints provide natural early-exit points, the model does not learn an input-dependent halting rule. As a result, the number of loops must be chosen externally, and all utterances are processed with the same maximum budget unless an additional exit criterion is introduced. Moreover, the budget can only be selected within the trained range: running more loops than the trained budget $K$ does not further improve recognition. Training always requires the full loop budget $K$, so the savings apply at inference only; a single training run nonetheless replaces training one model per budget.

Finally, LARM is implemented with CTC prediction feedback and evaluated primarily with greedy CTC decoding, with fixed 4-gram LM decoding used for the main benchmark results. The interaction between recurrent acoustic refinement and stronger decoding methods remains open. Extending the proposed loop conditioning and delayed feedback mechanisms to transducer or attention-based ASR models is left for future work.