TACE: TOKEN-AWARE CHUNKED ENCODING

Parkirat Sandhu[*] Cactus Compute
Henry Ndubuaku[†] Cactus Compute
Justin Lee[‡] Cactus Compute
Satyajit Kumar[§] Cactus Compute
Noah Cylich[¶] Cactus Compute
Karen Mosoyan[‖] Cactus Compute
Jakub Mróz[**] Cactus Compute

ABSTRACT

RoPE-enabled Transformer encoder–decoders deliver strong Automatic Speech Recognition (ASR), but full-context self-attention scales quadratically with utterance length and real deployments face an additional systems bottleneck: highly variable audio durations induce highly variable tensor shapes, which is unfriendly to kernel autotuning and compilation. Prior ASR stacks often (i) rely on absolute positional embeddings that couple models to a fixed index space and encourage padding/bucketing to a small set of maximum lengths, or (ii) adopt relative schemes such as RoPE that extrapolate better, yet are typically served with full-context encoder passes whose variable shapes remain hard to optimize. We introduce Token-Aware Chunked Encoding (TACE), a post-training conversion that executes an existing ASR model on fixed-duration chunks, batches chunks for stable kernels, and deterministically stitches encoder states back into the original sequence so decoding is unchanged. To compensate for the loss of cross-chunk encoder attention, we post-train with parameter-efficient LoRA using teacher-alignment losses (encoder-state regression and logit distillation). TACE also provides a simple streaming contract: fixed-shape chunked encoding with append-only stitched memory, avoiding encoder recomputation as audio arrives. On 7 ESB corpora, $C{=}2\,\mathrm{s}$ and $L{=}0.5\,\mathrm{s}$ yields a $1.33\times$ average encoder speedup (up to $2.05\times$ on 25–30 s utterances) while keeping normalized word error rate (WER) within 5 points of the base model.

1 INTRODUCTION AND RELATED WORK

Long-form ASR increasingly relies on Transformer encoder–decoders with full self-attention, often equipped with relative position mechanisms such as RoPE (Shaw et al., 2018; Su et al., 2021; Vaswani et al., 2017). While accurate, full-context encoders incur $\mathcal{O}(T^2)$ time and memory in the encoder length T, which becomes increasingly expensive for long utterances. In production, the systems cost is compounded by *shape variability*: serving audio with highly variable durations yields highly variable sequence lengths, pushing inference stacks toward many shape-specialized kernels or compiled variants and reintroducing tuning overhead (Chen et al., 2018; Chetlur et al., 2014).

A large body of work improves ASR efficiency by changing architectures and/or training objectives. On the encoder side, Conformers augment self-attention with convolution to better model local structure (Gulati et al., 2020), while Zipformer proposes a fast encoder design that reduces compute without sacrificing accuracy (Yao et al., 2024). On the objective and interface side, CTC (Graves et al., 2006) and RNN-T (Graves, 2012) provide streaming-friendly training criteria, and hybrid pipelines using WFST decoding and n-gram language models remain competitive in many settings

[*] parkirat@cactuscompute.com
[†] henry@cactuscompute.com
[‡] justin@cactuscompute.com
[§] satyajit@cactuscompute.com
[¶] noah@cactuscompute.com
[‖] karen@cactuscompute.com
[**] jakub@cactuscompute.com

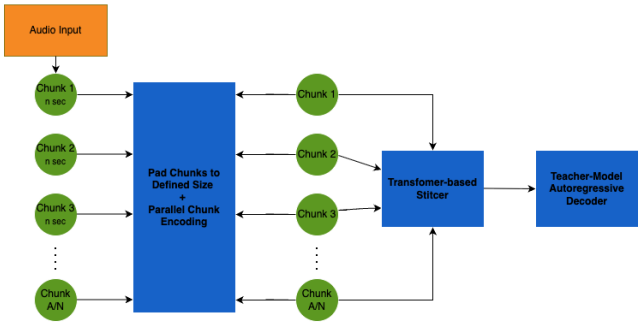

Figure 1: TACE architecture diagram. The encoder is evaluated on fixed-duration chunks (optionally with left-context overlap), chunk encodings are stitched into a single sequence by dropping overlap tokens, and decoding proceeds unchanged.

(Kneser and Ney, 1995; Mohri et al., 2002). There is also extensive work on streaming attention and limited-lookahead mechanisms, including chunked attention and explicit memory/state (e.g., Emformer and variants) (Liu et al., 2020; Shi et al., 2020). These approaches are effective but typically require *architecture changes* or *training streaming behavior from scratch*.

This paper targets a different operating point: **post-training conversion** of a deployed RoPE-based encoder–decoder. We seek a recipe that (i) preserves the model interface and decoding procedure, (ii) executes the encoder using *fixed shapes* to simplify deployment, and (iii) yields a natural streaming contract. Our contribution is TACE: we run the original encoder on fixed-duration chunks in batch-parallel fashion, then stitch chunk encodings into a single sequence and decode normally. Because chunking removes cross-chunk attention, we restore teacher behavior through lightweight post-training with LoRA and teacher-alignment losses. Empirically, TACE provides sizable encoder gains that grow with utterance length and reach **2**$\times$-class speedups on long buckets, while maintaining a small degradation in normalized WER.

2 Token-Aware Chunked Encoding

TACE is a wrapper around an existing ASR model with convolutional subsampling and a RoPE-enabled Transformer encoder–decoder (e.g., Jeffries et al., 2024). It leaves all network blocks intact and changes only the *execution strategy* of the encoder, followed by a small amount of post-training to align the resulting behavior with the original model.

2.1 Fixed-shape chunked encoding and token-aware stitching

Let x denote the input waveform and $\mathrm{Enc}(\cdot)$ the original encoder (including the conv front-end). Given a chunk duration C seconds and left context L seconds, TACE forms waveform segments

$$x_i = \mathrm{crop}(x;\ t \in [iC - L,\ (i+1)C])\,, \qquad i = 0, 1, \ldots, N-1,$$

with x_0 having no left-context prepend. Each x_i is padded to a fixed sample count so that all chunks share the same tensor shapes through subsampling and all Transformer layers. We then stack chunks on the batch dimension and evaluate the encoder in one call: $h_i = \mathrm{Enc}(x_i)$.

Stitching concatenates chunk encodings into a single sequence by dropping overlap states:

$$\hat{h} = [h_0;\ \mathrm{drop}(h_1, L);\ \ldots;\ \mathrm{drop}(h_{N-1}, L)].$$

The key detail is *token awareness*: with convolutional subsampling, a duration in seconds corresponds to an *integer* number of encoder steps once the front-end stride is fixed. TACE expresses C and L in seconds for usability, but implements $\mathrm{drop}(\cdot)$ in encoder-step units so boundary alignment is exact and the stitcher is a deterministic tensor slice/concat. RoPE is compatible with this execution because attention depends on relative position differences within each chunk; the trade-off is the absence of cross-chunk attention, addressed next.

2.2 LEARNED STITCHER

The overlap-drop concatenation produces a stitched sequence:

$$s = [h_0;\ \mathrm{drop}(h_1, L);\ \ldots;\ \mathrm{drop}(h_{N-1}, L)].$$

Rather than using s directly, we apply a lightweight Transformer encoder $\mathrm{Stitch}(\cdot)$ to refine the stitched representation. The stitcher operates at the encoder hidden dimension and is designed to correct boundary artifacts by enabling attention across chunk seams. In practice, we optionally restrict stitcher attention to a local window around boundaries to keep the overhead small. This cross-token attention helps recover interactions lost when full-context self-attention is removed, especially across chunk seams.

In our implementation, the stitcher is a Transformer encoder with 2 layers and 4 attention heads (using the same d_{model} as the base encoder). Its GPU forward-pass time is reported as *"Stitch (ms)"* in all tables and figures. The stitcher is trained jointly during post-training using the same teacher-alignment objectives as Section 2.3 (encoder-state regression and decoder logit distillation), and gradients flow through both the chunked encoder and the stitcher.

2.3 TEACHER-ALIGNED POST-TRAINING

Chunking restricts the encoder's receptive field, so we post-train the chunked model to recover the full-context teacher's behavior. Let h_T denote teacher encoder states from running the original model on the full utterance, and let $\hat{h}$ be stitched states from TACE. We combine (i) encoder-state regression and (ii) decoder logit distillation under teacher forcing:

$$\mathcal{L}_{\text{state}} = \left\| M \odot (\hat{h} - h_T) \right\|_2^2, \tag{1}$$

$$\mathcal{L}_{\text{KL}} = \sum_t \mathrm{KL}\Big(p_T(y_t \mid y_{<t}, h_T) \parallel p_S(y_t \mid y_{<t}, \hat{h}) \Big), \tag{2}$$

$$\mathcal{L} = \lambda_{\text{state}} \mathcal{L}_{\text{state}} + \lambda_{\text{KL}} \mathcal{L}_{\text{KL}}, \tag{3}$$

where M masks padding and can optionally emphasize boundary regions, and KL denotes Kullback–Leibler divergence (Hinton et al., 2015). To keep conversion lightweight, we adapt the base model with LoRA (Hu et al., 2021) (enabled on attention projections, and in our runs applied to both encoder and decoder). Implementation and training details are provided in Appendix A.

2.4 STREAMING ALIGNMENT

TACE yields a simple streaming contract without modifying the encoder–decoder API. As audio arrives, the system encodes each fixed-shape chunk once, drops overlap tokens, and appends the new encoder states to an ever-growing buffer; previously computed encoder outputs are never recomputed. On the decoder side, standard incremental decoding can reuse cached decoder self-attention, and cross-attention can reuse cached encoder projections while appending new states. Compared to streaming architectures that introduce explicit memory modules (Shi et al., 2020) or retrain streaming behavior end-to-end, TACE focuses on deployment constraints: stable kernel shapes and post-training conversion of an existing model.

3 RESULTS AND DISCUSSION

Table 1 summarizes macro-averaged results over 7 corpora with $L{=}0.5$ s. The recommended setting ($C{=}2$ s) improves the encoder from 18.20 ms to 13.71 ms on average (**1.33**× speedup), and even after accounting for stitching (1.74 ms), the net encoder+stitch speedup remains **1.18**×. At the same time, $\mathrm{WER_n}$ increases by 4.96 points, staying within a 5-point quality budget. Shorter chunks ($C{=}1$ s) slightly increase average encoder speedup but degrade $\mathrm{WER_n}$ beyond this budget, consistent with more frequent boundary seams. Longer chunks ($C{=}5$ s) reduce boundary frequency but perform poorly in our post-training regime, suggesting that converting a full-context model is sensitive to the train/test chunk-size mismatch and to optimization dynamics that overfit to smaller receptive fields.

Figure 2 shows how speedup and quality scale with utterance length for $C{=}2$ s, $L{=}0.5$ s. Encoder speedups increase monotonically with length and reach **2.05**× in the 25–30 s bucket, which is the

Table 1: Macro-average over 7 end-to-end speech benchmark (ESB) corpora (AMI, Earnings22, GigaSpeech, LibriSpeech clean/other, TED-LIUM, VoxPopuli; SPGISpeech excluded). Encoder times are GPU ms/utterance. "Stitch" is overlap-drop+concatenate on GPU. BASE timing differs slightly across blocks because each left-context sweep was measured in a separate evaluation run.

Setting	C (s)	L (s)	WER_n (%)	$\Delta\text{WER}_\text{n}$ (pts)	BASE enc (ms)	TACE enc (ms)	Stitch (ms)	Speedup (enc / enc+stitch)
Left context $L = 0.0$ s								
BASE	–	–	14.33	–	18.82	–	–	1.00 / 1.00
TACE	1.0	0.0	33.87	+19.54	18.82	13.47	1.83	1.40 / 1.23
TACE	2.0	0.0	23.02	+8.69	18.82	13.55	1.80	1.39 / 1.23
TACE	5.0	0.0	28.98	+14.65	18.82	14.08	1.81	1.34 / 1.18
Left context $L = 0.2$ s								
BASE	–	–	14.33	–	17.84	–	–	1.00 / 1.00
TACE	1.0	0.2	19.95	+5.62	17.84	13.27	1.69	1.34 / 1.19
TACE	2.0	0.2	18.25	+3.92	17.84	13.26	1.65	1.34 / 1.20
TACE	5.0	0.2	26.30	+11.97	17.84	13.77	1.62	1.30 / 1.16
Left context $L = 0.5$ s								
BASE	–	–	14.33	–	18.20	–	–	1.00 / 1.00
TACE	1.0	0.5	22.08	+7.75	18.20	13.73	1.77	1.33 / 1.17
TACE	2.0	0.5	19.29	+4.96	18.20	13.71	1.74	1.33 / 1.18
TACE	5.0	0.5	27.01	+12.68	18.20	14.19	1.73	1.28 / 1.14

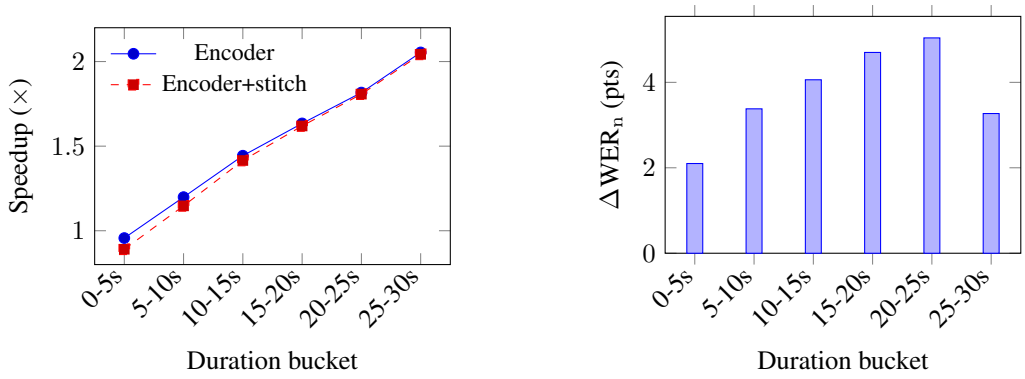

Figure 2: Length scaling for $C = 2$ s and $L = 0.5$ s (macro-averaged over 7 corpora; SPGISpeech excluded). Encoder speedups increase with utterance length and reach **2.05**× in the 25–30 s bucket; stitching adds a small, nearly length-independent overhead.

regime where quadratic full-context attention is most expensive. The stitch step adds a small overhead that is largely length-independent because it is dominated by simple tensor slicing and concatenation; this makes encoder-only and encoder+stitch speedups track closely for long utterances. The results per set (Appendix B) show the same qualitative pattern: conversational speech is most sensitive to the boundaries of segments, while read or structured speech tends to be more robust, but in all cases the speedup increases with duration buckets.

Overlap is an explicit quality–speed knob. Appendix B.2 reports the L=0 sweep: removing overlap improves speed but causes a clear WER_n degradation, consistent with boundary artifacts in the conv front-end and restricted encoder context. From a deployment perspective, the ability to trade quality for compute via a single parameter is useful, and the fixed-shape execution path remains identical across settings.

Finally, TACE is well-aligned with streaming constraints. Unlike full-context encoders, compute grows roughly linearly in the number of processed chunks and never revisits earlier audio. Unlike architecture changes that require re-training and re-validation, TACE converts a fixed deployed model with parameter-efficient post-training. This combination is attractive when the primary objective is to stabilize inference (fixed shapes, chunk-parallel kernels) while retaining the existing decoder, language-model fusion, and downstream tooling.

4 CONCLUSION

We propose TACE as a post-training recipe for transforming RoPE-based ASR encoder–decoder models into a fixed-shape, chunk-parallel encoder that naturally supports streaming via append-only stitched memory. Across 7 ESB corpora, we achieved 1.33× average encoder speedup (up to 2.05× on long buckets) with a modest increase in normalized WER at our recommended setting. We expect boundary-aware objectives, chunk-size jitter during post-training, and limited cross-chunk context mechanisms to further reduce the remaining quality gap, while providing considerable decreases to latency and computation.

REFERENCES

Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Megha Ghodrat, Haichen Shen, Mu Li, Yutian Tang, Ian Buck, Carlos Guestrin, and Arvind Krishnamurthy. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In *OSDI*, 2018.

Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cuDNN: Efficient Primitives for Deep Learning. arXiv:1410.0759, 2014.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. In *ICML*, 2006.

Alex Graves. Sequence Transduction with Recurrent Neural Networks. arXiv:1211.3711, 2012.

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. arXiv:2005.08100, 2020.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. arXiv:1503.02531, 2015.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685, 2021.

Nat Jeffries, Dima Gorin, Max Schwieterman, and Alon Grinar. Moonshine: Speech Recognition for Live Transcription and Voice Commands. arXiv:2410.15608, 2024.

Reinhard Kneser and Hermann Ney. Improved Backing-Off for M-Gram Language Modeling. In *ICASSP*, 1995.

Lidan Liu and others. Streaming Speech Recognition with Chunk-based Attention. arXiv:2001.09306, 2020.

Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted Finite-State Transducers in Speech Recognition. *Computer Speech & Language*, 16(1):69–88, 2002.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-Attention with Relative Position Representations. In *NAACL*, 2018.

Yuanhang Shi, Yongqiang Wang, Chunyang Wu, Zhengqin Li, Pingjian Zhang, Changliang Zou, and Xiangang Li. Emformer: Efficient Memory Transformer Based Acoustic Model for Low Latency Streaming Speech Recognition. arXiv:2010.10759, 2020.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced Transformer with Rotary Position Embedding. arXiv:2104.09864, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *NeurIPS*, 2017.

Yao Yao, Yang Yang, Haowen Xue, Minghui Zhou, and Daniel Povey. Zipformer: A Faster and Better Encoder for Automatic Speech Recognition. arXiv:2310.11230, 2024.

A Implementation and training details

We implement TACE as a thin wrapper around the original model. Chunking is performed in waveform time, and each chunk is padded to a fixed sample count. After the conv front-end, we compute the exact number of encoder steps corresponding to L seconds using the subsampling stride, and drop those overlap steps before concatenating chunk states. The stitch step is implemented as device-side slicing and concatenation, and we time it separately from the encoder forward pass.

We post-train the chunked student starting from the BASE weights with LoRA adapters (Hu et al., 2021). We use the combined objective in Section 2: masked encoder-state regression against the full-context teacher and KL distillation on decoder logits under teacher forcing (Hinton et al., 2015). Hyperparameters (LoRA ranks, target modules, loss weights, batch sizes, training steps) and additional ablations are provided in the companion training logs.

B Additional results

This appendix reports tables generated directly from the provided ESB JSON summaries. Unless otherwise noted, "macro average" assigns equal weight to each dataset in the set.

B.1 Aggregate summaries

Table 2: Aggregate results (7 corpora; excluding SPGISpeech) for left context $L = 0.5$ s. "Enc" is encoder forward time; "Stitch" is overlap-drop+concat; "Enc+St" is their sum. "Peak enc" is the maximum encoder-only speedup observed over all dataset × bucket pairs in this run.

Setting	C	L	WER_n	$\Delta\text{WER}_\text{n}$	BASE (ms)	Enc (ms)	Stitch (ms)	Enc+St (ms)	Speedup enc	Peak enc
BASE	–	–	14.33	–	18.20	–	–	–	1.00	–
TACE	1.0	0.5	23.83	9.50	18.20	13.55	2.77	16.32	1.34	1.92
TACE	2.0	0.5	19.29	4.96	18.20	13.71	1.74	15.45	1.33	2.05
TACE	5.0	0.5	33.10	18.77	18.20	20.39	5.60	25.99	0.89	1.36

B.2 Overlap sweep

Table 3: Aggregate results (7 corpora; excluding SPGISpeech) for left context $L = 0$ s. Removing overlap increases speed but degrades WER_n, consistent with boundary artifacts.

Setting	C	L	WER_n	$\Delta\text{WER}_\text{n}$	BASE (ms)	Enc (ms)	Stitch (ms)	Enc+St (ms)	Speedup enc	Peak enc
BASE	–	–	14.33	–	18.82	–	–	–	1.00	–
TACE	1.0	0.0	28.32	13.99	18.82	11.10	1.69	12.79	1.70	2.23
TACE	2.0	0.0	23.02	8.69	18.82	9.94	0.89	10.83	1.89	2.84
TACE	5.0	0.0	42.03	27.70	18.82	14.06	2.05	16.11	1.35	1.70

Table 4: Per-dataset WER_n (%) for $L = 0.5$ s across chunk sizes.

Dataset	BASE	TACE-1s	TACE-2s	TACE-5s
ami	13.32	20.54	20.87	37.15
earnings22	44.00	62.89	58.18	77.42
gigaspeech	10.55	20.21	16.03	27.94
librispeech_clean	3.48	8.61	4.68	7.79
librispeech_other	10.09	20.10	14.70	22.93
tedlium	13.78	22.18	19.05	30.46
voxpopuli	5.07	12.27	7.51	12.99

Table 5: Per-dataset WER (%) for $L = 0.5$ s across chunk sizes.

Dataset	BASE	TACE-1s	TACE-2s	TACE-5s
ami	23.56	36.32	36.91	65.63
earnings22	48.83	69.80	64.58	85.95
gigaspeech	13.90	26.63	21.12	36.78
librispeech_clean	15.17	37.56	20.41	33.97
librispeech_other	20.47	40.79	29.84	46.53
tedlium	15.65	25.18	21.63	34.60
voxpopuli	13.40	32.41	19.84	34.29

Table 6: Per-dataset encoder speedups for $L = 0.5$ s (relative to BASE encoder time).

Dataset	BASE enc (ms)	1s (enc)	1s (enc+st)	2s (enc)	2s (enc+st)	5s (enc)	5s (enc+st)
ami	19.63	1.25	1.05	1.26	1.20	0.86	0.73
earnings22	11.69	1.38	1.03	1.39	1.22	0.92	0.72
gigaspeech	18.76	1.20	1.00	1.19	1.10	0.88	0.72
librispeech_clean	23.89	1.42	1.06	1.46	1.26	0.94	0.72
librispeech_other	20.40	1.23	1.03	1.21	1.14	0.88	0.74
tedlium	19.89	1.35	1.06	1.38	1.25	0.94	0.73
voxpopuli	13.13	1.55	1.04	1.44	1.12	0.89	0.67

Table 7: Per-dataset $\mathrm{WER_n}$ (%) for $L = 0$ s across chunk sizes.

Dataset	BASE	TACE-1s	TACE-2s	TACE-5s
ami	13.32	27.65	27.12	46.52
earnings22	44.00	71.25	65.90	92.03
gigaspeech	10.55	21.12	19.10	34.77
librispeech_clean	3.48	16.72	12.30	26.23
librispeech_other	10.09	24.58	21.68	34.01
tedlium	13.78	30.06	25.15	41.93
voxpopuli	5.07	21.84	19.89	42.70

Table 8: Per-dataset WER (%) for $L = 0$ s across chunk sizes.

Dataset	BASE	TACE-1s	TACE-2s	TACE-5s
ami	23.56	48.92	47.98	82.33
earnings22	48.83	79.09	73.15	102.16
gigaspeech	13.90	27.83	25.16	45.77
librispeech_clean	15.17	72.89	53.60	114.39
librispeech_other	20.47	49.88	44.00	68.96
tedlium	15.65	34.14	28.57	47.57
voxpopuli	13.40	57.69	52.54	112.77

Table 9: Per-dataset encoder speedups for $L = 0$ s (relative to BASE encoder time).

Dataset	BASE enc (ms)	1s (enc)	1s (enc+st)	2s (enc)	2s (enc+st)	5s (enc)	5s (enc+st)
ami	18.62	1.74	1.53	2.07	1.88	1.41	1.23
earnings22	10.96	1.86	1.54	2.19	1.86	1.57	1.22
gigaspeech	18.52	1.70	1.50	1.95	1.78	1.44	1.20
librispeech_clean	24.44	1.81	1.56	2.48	2.16	1.64	1.24
librispeech_other	20.54	1.63	1.47	1.89	1.74	1.43	1.21
tedlium	20.25	1.78	1.56	2.07	1.84	1.52	1.22
voxpopuli	18.44	1.37	1.22	1.56	1.45	1.43	1.21

Table 10: Bucketed macro averages for $C = 2$ s, $L = 0.5$ s across 7 corpora (excluding SPGISpeech).

Bucket	BASE enc (ms)	TACE enc (ms)	Stitch (ms)	Speedup enc	Speedup enc+stitch	$\Delta\text{WER}_\text{n}$ (pts)
0-5s	8.07	8.52	1.08	0.96	0.89	2.10
5-10s	12.06	10.06	0.90	1.20	1.15	3.38
10-15s	15.45	10.70	0.77	1.44	1.42	4.06
15-20s	18.97	11.61	0.62	1.63	1.62	4.70
20-25s	23.11	12.72	0.55	1.82	1.81	5.04
25-30s	27.21	13.25	0.53	2.05	2.04	3.27

Table 11: Bucketed macro averages for $C = 2$ s, $L = 0$ s across 7 corpora (excluding SPGISpeech).

Bucket	BASE enc (ms)	TACE enc (ms)	Stitch (ms)	Speedup enc	Speedup enc+stitch	$\Delta\text{WER}_\text{n}$ (pts)
0-5s	7.81	6.64	0.72	1.19	1.08	5.49
5-10s	12.08	6.63	0.47	1.90	1.77	8.05
10-15s	15.45	6.47	0.32	2.40	2.28	10.34
15-20s	19.06	6.30	0.23	3.03	2.92	10.23
20-25s	23.15	6.18	0.19	3.75	3.64	9.31
25-30s	27.31	6.09	0.18	4.49	4.37	7.78

Table 12: SPGISpeech results (reported separately; excluded from main macro averages).

Setting	C (s)	L (s)	WER_n (%)	BASE enc (ms)	Speedup enc	Speedup enc+st
BASE	–	–	15.03	18.70	1.00	1.00
TACE	1.0	0.5	15.89	18.70	1.25	1.00
TACE	2.0	0.5	15.28	18.70	1.18	1.01
TACE	5.0	0.5	15.71	18.70	1.05	0.95

Table 13: Bucketed results on ami for $C = 1$ s, $L = 0.5$ s.

Bucket	n	BASE WER_n	TACE WER_n	$\Delta\text{WER}_\text{n}$	Speedup enc	Speedup enc+stitch
0-5s	100	13.28	15.04	1.76	0.93	0.78
5-10s	100	12.96	20.10	7.14	1.11	0.93
10-15s	100	12.23	19.87	7.64	1.40	1.25
15-20s	100	13.73	20.53	6.80	1.61	1.52
20-25s	100	14.89	23.12	8.23	1.73	1.67
25-30s	100	12.37	20.42	8.05	1.78	1.73

Table 14: Bucketed results on ami for $C = 2\,\text{s}$, $L = 0.5\,\text{s}$.

Bucket	n	BASE WER_n	TACE WER_n	$\Delta\text{WER}_\text{n}$	Speedup enc	Speedup enc+stitch
0-5s	100	13.28	12.99	-0.29	0.85	0.75
5-10s	100	12.96	19.48	6.52	1.09	1.01
10-15s	100	12.23	21.17	8.94	1.38	1.33
15-20s	100	13.73	21.62	7.89	1.55	1.52
20-25s	100	14.89	25.89	11.00	1.65	1.62
25-30s	100	12.37	20.97	8.60	1.77	1.74

Table 15: Bucketed results on ami for $C = 5\,\text{s}$, $L = 0.5\,\text{s}$.

Bucket	n	BASE WER_n	TACE WER_n	$\Delta\text{WER}_\text{n}$	Speedup enc	Speedup enc+stitch
0-5s	100	13.28	19.03	5.75	0.58	0.45
5-10s	100	12.96	33.39	20.43	0.77	0.65
10-15s	100	12.23	37.20	24.97	1.00	0.87
15-20s	100	13.73	43.64	29.91	1.19	1.07
20-25s	100	14.89	45.92	31.03	1.35	1.24
25-30s	100	12.37	43.55	31.18	1.34	1.23