

Peer Review

# Review of: "k2SSL: A Faster and Better Framework for Self-Supervised Speech Representation Learning"

Evgenii Arsentev<sup>1</sup>

1. Independent researcher

## 1. What the work claims

The paper presents k2SSL, an open-source framework (released inside icefall, footnote 1) for HuBERT-style self-supervised speech pre-training aimed at downstream ASR. Three changes are proposed. First, the HuBERT recipe itself is slimmed: a single Fp32 GroupNorm is kept in the convolutional feature extractor, and the cosine-similarity HuBERT loss of Eq. 1 is replaced by a plain cross-entropy over k-means labels (Eq. 2), following MelHuBERT. Second, the Transformer encoder is swapped for Zipformer and trained with ScaledAdam and the Eden scheduler. Third, data handling goes through Lhotse with the DynamicBucketingSampler, so that a 60,000-hour Libri-Light run starts in "dozens of seconds" (§III.D).

The headline result is Table II. Against a fairseq HuBERT Base pre-trained with Adam (1878 GPU hours), a k2SSL Zipformer Base with CE loss reaches 3.67 / 7.86 / 3.80 / 7.87 WER on dev-clean / dev-other / test-clean / test-other in 531 GPU hours, which the authors report as relative reductions of 24.8% / 34.8% / 23.7% / 32.4% and a 3.53x speedup. Table III repeats the comparison with letter-level CTC fine-tuning against wav2vec 2.0, HuBERT, and WavLM Base, with and without a 4-gram LM. Table IV shows a Zipformer Large pre-trained on Libri-Light for 250k steps landing at 1.8 / 4.0 on test-clean / test-other, next to a k2SSL-fine-tuned HuBERT Large at 1.8 / 3.9.

## 2. Whether the conclusion holds on the data

What I value most here is that the comparison in Table II is staged. Rows 1 to 4 change one block of the system at a time (toolkit and optimizer, then loss, then encoder), all four are fine-tuned in the same framework with the same pruned RNN-T recipe on LibriSpeech-100h, and the speedup is given in total GPU hours rather than in wall-clock time on unequal hardware. That design lets a reader see where the

gains come from. It is also rare and welcome that the authors fine-tune public checkpoints themselves (Table III caption, Table IV text) instead of only copying numbers from older papers, and that the strongest Large claim is phrased modestly as "comparable".

I re-did the arithmetic for the reported relative numbers, and it holds. Table II, Row 4 vs Row 1, dev-other:  $(12.06 - 7.86) / 12.06 = 4.20 / 12.06 = 34.8\%$ ; test-other:  $(11.65 - 7.87) / 11.65 = 3.78 / 11.65 = 32.4\%$ ; dev-clean:  $1.21 / 4.88 = 24.8\%$ . The speedups are  $1878 / 1270 = 1.48$ ,  $1878 / 886 = 2.12$ , and  $1878 / 531 = 3.54$  (reported as 3.53x). In Table III, the claimed 28.9% (13.5 vs 9.6, dev-other, no LM), 26.2% (13.0 vs 9.6), 22.8% (5.7 vs 4.4, test-clean), 19.0% (7.9 vs 6.4), 18.0% (7.8 vs 6.4), and 11.8% (3.4 vs 3.0) all reproduce. The 19.9% memory overhead in §III.A is  $(13.14 - 10.96) / 10.96 = 19.9\%$ . No errors there.

Where I think the conclusions reach past the evidence is in attribution. Four points.

**The Zipformer gain mixes encoder and batch size.** From Table I, the k2SSL HuBERT Base trains with 87.5 s per GPU and gradient accumulation 4 on 8 GPUs, i.e.,  $87.5 \times 8 \times 4 = 2,800$  s of audio per update; Zipformer Base trains with 600 s per GPU, no accumulation, i.e.,  $600 \times 8 = 4,800$  s per update. The number of epochs is matched (300, §IV.A), and I checked that it is roughly consistent:  $2,800 \times 400k$  and  $4,800 \times 225k$  both come out near 1.1 billion s, against 300 epochs of 960 h  $\approx 1.04$  billion s. So total audio is controlled, but the effective batch is 1.7 times larger for Zipformer. The authors present the larger batch as a benefit of the architecture (§III.B), which is fair as an engineering argument. As a scientific claim about the encoder, though, Row 4 vs Row 3 cannot separate "Zipformer is a better SSL backbone" from "a larger batch helps HuBERT-style training", which §III.B itself notes via [3]. A Transformer run at the same 4,800 s effective batch (with accumulation) would settle it.

**Row 2 changes two things at once.** Row 2 vs Row 1 in Table II switches the toolkit (fairseq to k2SSL, including the Lhotse data pipeline) and the optimizer (Adam to ScaledAdam) at the same time. The 1.48x speedup is therefore not attributable to either one of them. The claim in §III.A that per-layer Fp32 LayerNorm "contributes little to performance" is not backed by a number that I could find; the only measurement given is the peak memory of the extractor (10.96 GB for the HuBERT Base configuration vs 13.14 GB for the Large configuration), and that compares two different extractor setups rather than isolating the norm.

**The "2.1x" sentence in §IV.B is ambiguous.** The text says the simplified loss "further provides a around 2.1x pre-training speedup" and attaches it to "(Row 3 vs. 2, test-other)". From Table II, Row 3 vs Row 2 is  $1270 / 886 = 1.43x$ ; the 2.12x is Row 3 vs Row 1. The WER part of the same sentence (3.8% on test-other,

$(11.53 - 11.09) / 11.53 = 3.8\%$ ) is indeed Row 3 vs Row 2. I would split these two numbers so that the baselines are explicit.

**The Large comparison uses steps as the unit.** §IV.B states that Zipformer Large matches HuBERT Large "with much fewer pre-training steps (250k vs. 400k)". Steps are only comparable if the batch size per step is comparable. Table I gives Zipformer Large 350 s per GPU on 32 GPUs, i.e., 11,200 s per step; the batch for HuBERT Large is not given in this paper, so I could not check whether 250k Zipformer steps cover less audio or more. GPU hours for the Large models are also absent, although they would be the natural continuation of Table II's main argument.

On statistics: every WER in Tables II to IV is a single run with no seed variance. For LibriSpeech dev/test sets, differences of 0.1 absolute or less (Table II, Row 2 vs Row 1, dev-other 12.05 vs 12.06; Table IV, test-other 4.0 vs 3.9) are within what seed noise usually produces, so I would avoid reading them in either direction. The large Zipformer Base gains (over 3 absolute points on the "other" sets) are well outside that range, and I am convinced they are real. The paper also evaluates only ASR on LibriSpeech. The abstract is careful to say "with a focus on downstream ASR tasks," and the Conclusions say "tailored for downstream ASR tasks," but the Introduction motivates the work with "universal representations" and generalizability across downstream tasks; a SUPERB subset, or at least one out-of-domain ASR set, would show whether the multi-rate Zipformer representations transfer as well as HuBERT's.

One mixed-source issue in Table III. Some baseline rows carry a citation (wav2vec 2.0 [2], WavLM [4], HuBERT Base 4-gram [3]) and the HuBERT Base no-LM row does not, which suggests it was re-fine-tuned by the authors. Mixing self-run and copied numbers within a table is acceptable, but each row should say which it is, because the copied rows use different fine-tuning code and decoding settings.

### 3. Reproducibility

This is the strongest side of the work. The code is public (footnote 1), the starting labels are pinned to a specific public checkpoint (footnote 3, `hubert_base_ls960.pt`), datasets are standard, hardware is named (V100 32 GB, Table I caption), and Table I lists parameters, learning rates, per-GPU batch durations, and accumulation for every setup. That is more than most SSL papers give.

What I could not find in the Qeios text:

- A commit hash or recipe path inside icefall. The repository is large and evolves; a link to the exact `egs/` recipe and commit used for Tables II to IV would make the numbers reproducible rather than

approximately reproducible.

- Direct download locations for the released k2SSL checkpoints. Footnote 1 and the Conclusions say pre-trained models are available in the icefall repository, but no direct checkpoint link or model identifier is given.
- Seeds, and the number of fine-tuning epochs or steps for each row.
- Masking probability and span length (only "The same masking strategies as in [3] are used"), the Eden scheduler parameters, and warm-up.
- For CTC + 4-gram decoding (Table III): the values of the LM weight  $w_1$  and word score  $w_2$  defined in Eq. 3, the beam size, and which 4-gram LM was used.
- How GPU hours were measured. In particular, whether the 1878 hours for the fairseq row were measured by the authors on the same V100 cluster or derived from published figures, and whether label generation (the first-iteration HuBERT model and k-means) is counted. All k2SSL rows reuse labels from the first-iteration fairseq model (§IVA, Pre-training), so if Row 1 includes the first iteration and the others do not, the speedups are inflated; if no row includes it, that should be stated.
- Peak memory during pre-training for Rows 1 to 4. "More memory-efficient" is one of the headline claims of the abstract, yet the only memory figures are for the feature extractor in isolation (§III.A).

Figure 1 appears on Qeios only as its caption in the text I reviewed, so I could not check it against §III.B.

#### 4. What specifically to fix

1. **Table II, Row 4 vs Row 3.** Add one Transformer CE run at the same effective batch as Zipformer Base (4,800 s per update, via gradient accumulation), or state plainly that the encoder gain and the batch-size gain are not separated.
2. §IV.B, second bullet: rewrite "a around 2.1x pre-training speedup" so it says 1.43x relative to Row 2 (1270 / 886) and 2.12x relative to Row 1. The current wording attaches both numbers to "Row 3 vs. 2".
3. §III.A, **LayerNorm claim.** Support "contributes little to performance" with a WER number, or soften it. The memory comparison (10.96 GB vs 13.14 GB) is between Base and Large extractor configurations, so I would also report the extractor memory for the k2SSL variant directly.
4. Split Row 2 of Table II into toolkit-only and optimizer-only changes if compute allows. If not, a sentence saying the 1.48x combines both changes is enough.
5. **Table IV and §IV.B, Large comparison.** Replace "250k vs. 400k steps" with hours of audio seen and total GPU hours for both Large models. Give the HuBERT Large batch configuration.

6. Add a peak-memory column to Table II, since memory efficiency is one of the two headline claims.
7. **Table III.** Mark each baseline row as "reported in [x]" or "fine-tuned by us from the official checkpoint".
8. State in §IV.A what the 1878 GPU hours (Table II, Row 1) include and how they were measured; say whether first-iteration label generation is counted in any row.
9. Seed variance for at least Table II Rows 3 and 4 (two or three seeds of fine-tuning would do). Then the 0.01-point and 0.1-point differences in Table II Row 2 and Table IV can be interpreted.
10. Add a commit hash, recipe path, and checkpoint links to footnote 1, plus masking, scheduler, and LM-decoding parameters (an appendix table is fine).
11. **Scope.** Either add a non-ASR or out-of-domain evaluation, or align the Introduction with the "focus on downstream ASR tasks" stated in the abstract.
12. Small consistency items. Section numbering switches from "1. Introduction" to "II.", tables are cited as both "Table 1" and "Table I", the Introduction writes "3.53 pre-training speedup" without the "x", and §IV.A refers to "Rows 1 & 2" and "Rows 3 & 4" of Table I, which actually has nine rows (pre-train and fine-tune lines for each setup).

## 5. Recommendation

**Minor revisions.** The engineering contribution is real and useful: a public framework that pre-trains a Base-size SSL model on 8 V100s, with a fully staged comparison showing both lower cost and clearly lower WER (Table II, Row 4). The numbers I could check are internally consistent. The revisions I ask for are mostly about attribution (batch size vs encoder, which change gives which speedup), units for the Large comparison, and the reproducibility details listed in §3. None of them, in my reading, threatens the main result for the Base models.

## Declarations

**Potential competing interests:** No potential competing interests to declare.