

CIRCUITTREE: COMPOSING CIRCUITS INTO A SYLLABLE TRANSCRIBER

Jiankun Wei¹, Ewan Dunbar², Gerald Penn³

¹²³University of Toronto

ABSTRACT

The goal of circuit discovery is to decompose a monolithic neural network into tiny modular subnetworks which are interpretable in the sense that each does an identifiable sub-task. We develop a practical method for composing arbitrarily many circuits into a single model. We propose a greedy, density-scaled composition algorithm that overcomes limitations of previous attempts to compose circuits by naive union, then generalize it to N circuits via CircuitTree, a new data structure built on a $\Theta(N \log N)$ tree-compose schedule. We compose circuits pruned from HuBERT for classifying individual phonemes into a syllable classifier, a capability none of the 39 phoneme circuits has individually. Chaining CircuitTrees into a hierarchy-of-circuits pipeline, we transcribe open syllables at 64.8% accuracy, within 1.2 points of a 66.0% ceiling that shares no circuitry at all, with 99.5% of circuit components shared.

Index Terms— circuit discovery, mechanistic interpretability, model merging, modularity, HuBERT

1. INTRODUCTION

Many researchers has proposed to view modern multi-task neural network models as the composition of large numbers of sub-networks (circuits) with independent capabilities. [1–6]. Early work identified circuits manually [7]; automated methods soon followed [8], reducing circuit discovery to a process of choosing a metric and a dataset to define a behaviour, and patching activations until the responsible components emerge.

While much research has proposed circuit discovery as a tool for deepening our understanding of black-box models, the ability to decompose models into small, modular sub-networks has practical implications. Researchers have envisioned circuit discovery as a way to “re-engineer” a network by decomposing the monolithic model into modular, Lego-like pieces, each responsible for a single task [1, 2, 6, 9, 10]. The practical payoff of this decomposition is modularity. Once primitive circuits are identified, they can be joined to solve complex tasks without having to rediscover every possible combination from scratch. Merullo, Eickhoff, and Pavlick [3] found that a circuit discovered for one task in

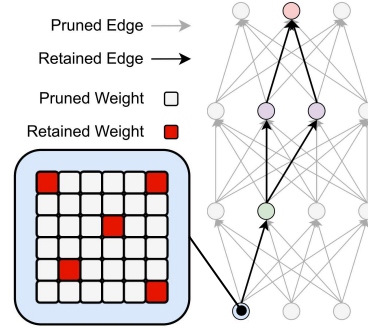


Fig. 1: Circuit discovery of weight and edge circuits.

GPT-2 is substantially reused for a related one, showing that a small hand-designed patch to four attention heads is sufficient to change the model’s behaviour. Mondorf et al. [1] went further, posing general circuit composability as a fundamental open question and combining circuits for ten string-edit operations through mask-level set operations. However, their experiments were limited to synthetic toy models generated via Tracr [11]. Most recently, Yu et al. [2] discovered that composing two independently discovered circuits pruned from GPT-2 for different tasks by simple mask union largely preserves functional performance.

This is a compelling vision, but the evidence for it remains preliminary. Existing composition results stop at a pair of circuits, or at best a handful, and success is measured by whether the composed circuit still performs the individual tasks it was pruned on. Can primitive circuits can be composed at scale, and into new capabilities? We answer both halves of this question: composition survives the jump from a hand-verified pair to dozens of independently-discovered circuits merged at once, and the resulting shared circuitry is able to do a more complex task that none of the 39 source circuits were individually pruned to solve. We do this by composing circuits pruned from an English phoneme classifier based on HuBERT, about as elementary a building block as a spoken language offers, into an syllable classifier.

The outline of this paper is as follows. First, we identify failures of plain mask union (density domination, interference) and design a greedy, density-scaled algorithm that composes a pair of circuits. Second, we generalize this to N circuits with the help of a new data structure, CIRCUITTREE.

This work was supported by [embargoed for anonymous review].

Third, we compose the resulting vowel and consonant CIRCUITREES into a hierarchy-of-circuits pipeline with 99.5% of matrices shared across phonemes. The resulting pipeline transcribes English CV and VC syllables, a task the individual circuits were not trained for, and which the composition algorithm does not optimize for either. We reach 64.8% accuracy against the ceiling of 66.0% obtained by running the circuits independently.

2. BACKGROUND ON CIRCUIT DISCOVERY

Circuit-discovery methods split along *what* they prune. **Weight-level** methods learn a differentiable mask directly over model parameters, e.g., via hard-concrete L_0 relaxations [12] or movement pruning [13], yielding a sparse sub-network but leaving the dense computation graph untouched. **Edge-level** methods instead prune the residual-stream computation graph itself while keeping weights dense: ACDC searches over activation patches [8], and Edge Pruning [14] learns a differentiable mask over edges with a straight-through relaxation. DiscoGP [2] is a circuit discovery algorithm that does both weight and edge pruning.

3. EXPERIMENTAL SETUP

Dataset. We train and evaluate on the **Articulatory Index** corpus [15], originally collected by [16], which contains pronunciations of 10,329 CV and VC monosyllables produced by 20 English speakers (12 male, 8 female). The data for each phoneme is split into train, validation, and test sets. Results reported are for test. Thresholds for ensemble classifiers described in Section 4 are tuned on the validation set. All accept/reject decisions in Algorithm 1 and its N -way extension use tolerance $\varepsilon = 0.02$ absolute F1; scale factors (Eq. 1) are clipped to $[0.05, 5.0]$; the syllable type gate (§6) is itself a DISCOGP circuit under the same protocol.

Following [6], we prune 15 vowel and 24 consonant DISCOGP circuits from HuBERT-base [17] with a frozen mean-pooled linear head trained on the binary task of identifying whether a given vowel or consonant in present the syllable. Below, we combine these binary circuits into ensemble multi-class classifiers as described in Section 4.

The resulting weight circuits consist of 144 logit tensors (12 layers $\times$ {Q,K,V,O,MLP-in, MLP-out} $\times$ {weight,bias}). Edge circuits are 49 logit tensors, one per downstream node, each holding one logit per eligible upstream head or MLP.

4. PAIRWISE CIRCUIT COMPOSITION

We start by developing algorithms for composing two phonemes’ circuits p_i, p_j . We combine pairs of binary classifiers into three-way classifiers p_i, p_j, other . Each binary classifier for phoneme p_i is associated with a threshold τ_i , tuned by validation set. When combining two classifiers, whichever

phoneme’s confidence surpasses its threshold the most will be the prediction. If neither does, we predict “other”. In this setting, the classifiers being combined can be run independently, or can be composed into a single system with shared weights and edges.

We first test a naive composition strategy by taking the element-wise union of weights and edges present in either circuit. As shown in Figure 2, the independent, unmerged classifiers (Figure 2a) substantially outperform this naive union across all vowel phoneme pairs (Figure 2b). We attribute this performance degradation to two key issues: *density domination* and *interference*.

By *density domination* we refer to the issue that circuits can have very different weight densities (2–3% for some phonemes, 15–20% for others). Simple union is dominated by the denser circuit, so the composition behaves like the dense circuit. We resolve this by scaling each mask in both circuits, or, in the general case, in all N circuits. For a shared key of the underlying model k , let $d_1(k), \dots, d_N(k)$ denote the fraction of active entries in each phoneme $p_1, \dots, p_N$ ’s mask. We calculate each phoneme’s scale factor at key k $\sigma_i(k)$ (which have the same shape as circuit masks at key k) as follows:

$$\sigma_i(k) = \text{clip}\left(\frac{t(k)}{d_i(k)}, [s_{\min}, s_{\max}]\right) \quad (1)$$

where

$$t(k) = \left(\prod_{n=1}^N d_n(k)\right)^{1/N}, \quad (2)$$

is the geometric mean of the densities.

In following MODEL FORWARD() method where N circuits are applied to the HuBERT model, we first element-wise union (naive union) the N circuits and select the kept parameter/edge connection, then scale them by element-wise multiplication of the scale factor $\sigma_i(k)$.

Even at matched density, overlaying two independently optimized masks introduces circuit *interference*: the presence of a given weight or edge may be crucial to the behaviour of one circuit but detrimental to the other. As the objects we are combining are binary masks, the techniques developed by [18] for the case of model combination by weight averaging do not apply. We instead propose a greedy algorithm to merge a subset of 90% the matrices that do not decrease the F1 score for each vowel more than some threshold.

Algorithm 1 sorts the computation graph topologically and visits every key exactly once, proposing to take the union of the two circuits’ masks for the given matrix, which we apply twice to the weight mask, then to the edge mask, only if both circuits’ binary F1 stays within ε of its independent (unmerged) performance.

Figure 2c shows the performance is comparable to independent (unmerged) performance (Figure 2a).

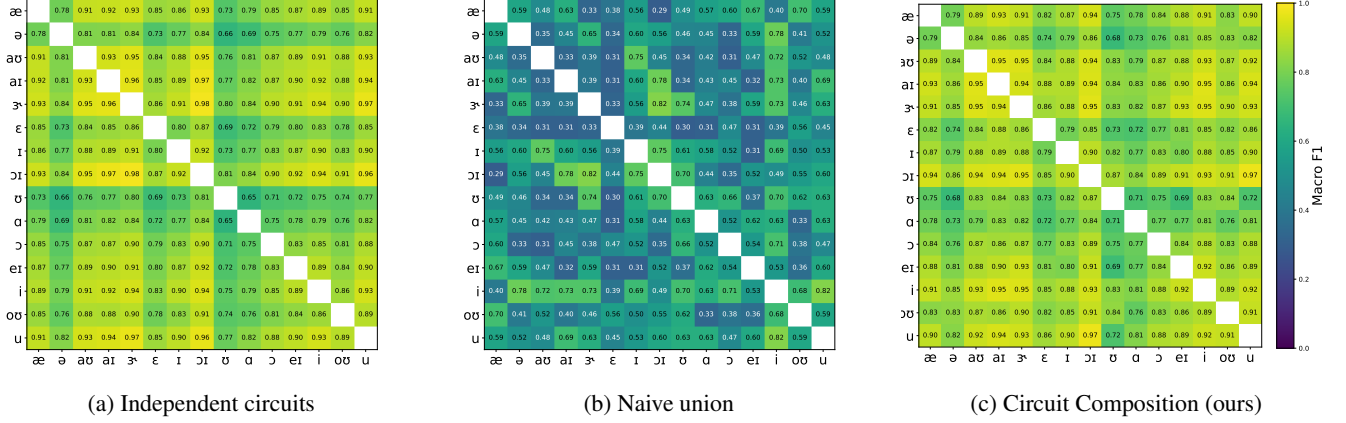


Fig. 2: Pairwise macro-F1 heatmaps across all 15 vowels under three composition strategies: (a) naive union, (b) unmerged union, and (c) merged circuits.

Algorithm 1 Merge Clusters

Require: circuits $\mathcal{C} = \{C_n\}_{n=1}^N$, each $C_n = \{M_n(k)\}_{k \in K}$ a set of $|keys|$ matrices; thresholds $\{\tau_n\}_{n=1}^N$; tolerance ε

- 1: $K \leftarrow \text{TOPOLOGICAL SORT}(keys)$
- 2: compute $\sigma_n(k) \forall k \in K, \forall n \in [N]$ via Eq. 1
- 3: $f_n^0 \leftarrow \text{MODEL FORWARD}(C_n, \emptyset, \tau_n) \quad \forall n \in [N]$
- 4: $S \leftarrow \emptyset; \Sigma \leftarrow \emptyset$
- 5: **for** each $k \in K$ **do**
- 6: $S' \leftarrow S \cup \{k\}$
- 7: $\hat{\mathcal{C}} \leftarrow \mathcal{C}$ with masks unioned according to S'
- 8: $\Sigma \leftarrow \sigma_i(k)$
- 9: $f_n \leftarrow \text{MODEL FORWARD}(\hat{\mathcal{C}}_n, \Sigma, \tau_n) \quad \forall n \in [N]$
- 10: **if** $f_n \geq f_n^0 - \varepsilon$ **for all** $n \in [N]$ **then**
- 11: $S \leftarrow S'$ ▷ accepted
- 12: **end if**
- 13: **end for**
- 14: **return** $S, \{\hat{\mathcal{C}}_n|_S\}_{n=1}^N, \Sigma|_S$

Algorithm 2 Linear schedule

- 1: **function** $\text{LINEAR}(C_1, \dots, C_n)$
- 2: $T \leftarrow C_1$
- 3: **for** $v = 2$ **to** n **do** $T \leftarrow \text{MERGECLUSTERS}(T, C_v)$
- 4: **end for**
- 5: **return** T
- 6: **end function**

5. TREE-MERGE SCHEDULES AND CIRCUIT TREE DATA STRUCTURE

Composing all 15 vowels (or 24 consonants) into a single module still raises two further questions that a single invocation of Algorithm 1 cannot answer on its own. Assuming that we combine circuits iteratively, by adding one phoneme at a time, in what *order* should circuits be combined? And, when each new phoneme is added, how do we carry forward information about sharing validated at previous iterations?

The simplest strategy composes N phonemes one at a

Schedule	Accuracy(%)	Density(%)
Unmerged	82.3	1.13
Linear ($\Theta(kn^2)$)	77.9	1.11
Balanced ($\Theta(kn \log n)$)	81.8	1.10
Kruskal merge	82.2	1.10
HuBERT Frozen [6]	86.4	100

Table 1: 15-way vowel accuracy of the N -way merge schedules vs. Full HuBERT model, with post-merge circuit density.

Algorithm 3 Balanced schedule

- 1: **function** $\text{BALANCED}(C_1, \dots, C_n)$
- 2: $Q \leftarrow [C_1, \dots, C_n]$
- 3: **while** $|Q| > 1$ **do**
- 4: $Q' \leftarrow []$
- 5: **for** $i = 0, 2, 4, \dots$ **while** $i+1 < |Q|$ **do**
- 6: $Q'.\text{append}(\text{MERGECLUSTERS}(Q[i], Q[i+1]))$
- 7: **end for**
- 8: **if** $|Q|$ odd **then** $Q'.\text{append}(Q[\text{last}])$
- 9: **end if** ▷ carried forward
- 10: $Q \leftarrow Q'$
- 11: **end while**
- 12: **return** $Q[0]$
- 13: **end function**

time (Algorithm 2). The trouble is that step i merges a single new member into a cluster that already holds $i-1$ members, so MERGECLUSTERS (Algorithm 1) must re-validate all i resulting members against every one of the k keys, costing ki MODEL FORWARD calls. These costs accumulate over the whole chain:

$$C_{\text{lin}} = k \sum_{i=2}^n i = k \left(\frac{n(n+1)}{2} - 1 \right) = \Theta(kn^2). \quad (3)$$

A more efficient strategy avoids ever growing one cluster far larger than its merge partner. Instead, singletons are merged into pairs, pairs into fours, and so on (Algorithm 3).

At round $r = 0, \dots, \log_2 n - 1$, every cluster has size 2^r , and there are $n/2^{r+1}$ merges in that round, each invoking MERGECLUSTERS on $2 \cdot 2^r$ members at a cost of $k \cdot 2^{r+1}$ MODELFORWARD calls. The result is that *every round costs the same, kn , regardless of r* :

$$k \cdot \underbrace{\frac{n}{2^{r+1}}}_{\text{merges in round } r} \cdot \underbrace{2^{r+1}}_{\text{members validated / merge}} = kn. \quad (4)$$

Summing this constant cost, kn , over the $\log_2 n$ rounds:

$$C_{\text{bal}} = kn \log_2 n = \Theta(kn \log n). \quad (5)$$

Both linear and balanced share a limitation: both are *blind* schedules, choosing merge order purely from cluster size and never from how similar two phonemes actually are. We include a third schedule as an upper bound on achievable performance. Here, the distance between two phonemes is defined by the merge ratio from §4, so that pairs sharing little of their masks are placed far apart. Kruskal’s algorithm is then applied repeatedly, always composing the two closest clusters that have not yet been joined. This schedule’s complexity ranges from $\Theta(kn \log n)$ in the best case to $\Theta(kn^2)$ in the worst case. As Table 1 shows, the balanced schedule matches the performance of Kruskal merging while remaining $\Theta(kn \log n)$, making it preferred in practice.

5.1. The CIRCUITTREE primitive

MERGECLUSTERS operates on nodes of a CIRCUITTREE: a binary tree whose leaves hold a phoneme’s original circuit, and whose internal nodes record *only* which keys are shared among their two children’s members at that level (Figure 3).

To see why this structure is necessary, consider the natural alternative: re-union the raw masks of all current members every round, and fall back to fully private masks when any member’s F1 drops too far. This memoryless approach has a serious flaw. If a later round failed by an unrelated third phoneme on some key k , the naive approach discards *all* sharing on k , including sharing between two phonemes that had already validated it several rounds earlier.

CIRCUITTREE avoids this by never forgetting a validated result. Instead of recomputing sharing from scratch, it resolves each member’s effective mask for key k by walking down the tree and taking the first (i.e. closest-to-root) level at which that key was successfully shared, falling back to the member’s own leaf if it never was:

$$\text{RESOLVE}(\mathcal{T}, m, k) = \begin{cases} U_{\mathcal{T}}(k) & k \in \text{merged}(\mathcal{T}) \\ M_m(k) & \text{at a leaf} \\ \text{RESOLVE}(\text{org}(\mathcal{T}, m), m, k) & \text{else} \end{cases} \quad (6)$$

Scale factors are accumulated along the same path. Because sharing is monotone non-decreasing along any root-to-leaf walk, a failed merge at the root can never undo a key that two phonemes already agreed to share three levels down.

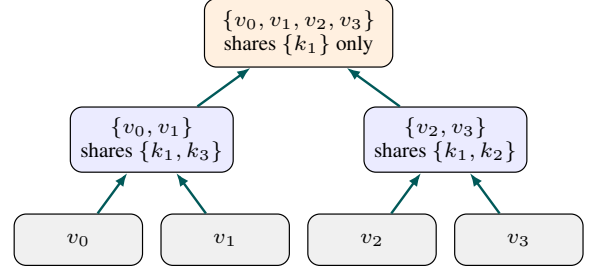


Fig. 3: A CIRCUITTREE over four phonemes. k_1 validates at every level. k_3 only validated inside $\{v_0, v_1\}$: the root-level merge failed for k_3 , but v_0, v_1 still use their shared value for it via Eq. 6, rather than reverting to private masks.

Table 2: CV/VC-syllable transcription accuracy.

System	Accuracy (%)	Merge Ratio (%)	
		All	Partial
Unmerged	66.0	0.0	0.0
CIRCUITTREE	64.8	30.3	99.5

6. HIERARCHY-OF-CIRCUITS TRANSCRIBER

We now compose the merged vowel and consonant CIRCUITTREE modules into a syllable classifier. The pipeline has two stages. First, a separately-pruned DISCOGP circuit acts as a *gate*: it determines the syllable’s consonant/vowel order (e.g. CV vs. VC). Second, this order routes the input to the corresponding vowel or consonant CIRCUITTREE module, which resolves the specific phoneme. The two stages are combined by slotting the predicted phonemes into the given template.

Table 2 contains our central result. Collapsing 39 independent, binary phoneme circuits into just two shared CIRCUITTREE modules costs only 1.2 points of accuracy ($66.0\% \rightarrow 64.8\%$), while 99.5% of keys are partially shared and 30.3% are shared across all phonemes. In other words, the vast majority of each phoneme’s circuitry is recovered from a common pool rather than stored independently, at a small and controlled cost to accuracy.

7. CONCLUSION

This paper has demonstrated circuit composition at scale: a density-scaled, interference-aware merge algorithm, generalized through CIRCUITTREE to compose many circuits at once. Composing primitive circuits for the vowels and consonants of English into a working transcriber shows that primitive circuits can be joined well beyond a handful, and into a capability none of them were individually pruned for. We believe this method is promising for much more complex problems, both in speech and in other domains.

8. COMPLIANCE WITH ETHICAL STANDARDS

This study was conducted retrospectively using the Articulatory Index corpus, a pre-existing, publicly available speech dataset distributed by the Linguistic Data Consortium. No new data involving human or animal subjects were collected as part of this work. As this research involved secondary analysis of an existing, licensed open dataset rather than the collection of new human subject data, no additional ethical approval was required.

9. REFERENCES

- [1] P. Mondorf, S. Wold, and B. Plank, “Circuit compositions: Exploring modular structures in transformer-based language models,” in *Proc. ACL*, 2025.
- [2] Lei Yu et al., “Sheaf discovery with joint computation graph pruning and flexible granularity,” in *Proc. EMNLP*, Nov. 2025, pp. 8833–8848.
- [3] J. Merullo, C. Eickhoff, and E. Pavlick, “Circuit component reuse across tasks in transformer language models,” in *Proc. ICLR*, 2024.
- [4] Xi Chen et al., “All circuits lead to rome: Rethinking functional anisotropy in circuit and sheaf discovery for LLMs,” in *Proc. ICML*, 2026, Proceedings of Machine Learning Research.
- [5] Pratim Chowdhary, Peter Chin, and Deepnab Chakrabarty, “K-MSHC: Unmasking minimally sufficient head circuits in large language models with experiments on syntactic classification tasks,” *arXiv preprint arXiv:2505.12268*, 2025.
- [6] Wei et al., “Sparse weight and edge circuit discovery in transformer-based acoustic models,” *IEEE Spoken Language Technology Workshop 2026*, 2026.
- [7] Kevin Ro Wang et al., “Interpretability in the wild: A circuit for indirect object identification in GPT-2 small,” in *Proc. ICLR*, 2022.
- [8] Arthur Conmy et al., “Towards automated circuit discovery for mechanistic interpretability,” in *NeurIPS*, 2023, vol. 37, pp. 16318–16352, Curran Associates Inc.
- [9] R. Pan and H. Rajan, “On decomposing a deep neural network into modules,” in *Proc. ESEC/FSE*, 2020, pp. 889–900.
- [10] H. Kingetsu, K. Kobayashi, and T. Suzuki, “Neural network module decomposition and recomposition,” *arXiv preprint arXiv:2112.13208*, 2021.
- [11] D. Lindner, J. Kramár, S. Farquhar, M. Rahtz, T. McGrath, and V. Mikulik, “Tracr: Compiled transformers as a laboratory for interpretability,” in *NeurIPS*, 2023.
- [12] C. Louizos, M. Welling, and D. P. Kingma, “Learning sparse neural networks through L_0 regularization,” in *Proc. ICLR*, 2018.
- [13] V. Sanh, T. Wolf, and A. M. Rush, “Movement pruning: Adaptive sparsity by fine-tuning,” in *NeurIPS*, 2020.
- [14] A. Bhaskar, A. Wettig, D. Friedman, and D. Chen, “Finding transformer circuits with edge pruning,” in *NeurIPS*, 2024.
- [15] Thomas Schatz et al., “Articulation Index LSCP (LDC2015S12),” Linguistic Data Consortium, 2015.
- [16] P. Fousek, P. Svojanovsky, F. Grezl, and H. Hermansky, “New nonsense syllables database analyses and preliminary asr experiments,” in *Proc. ICSLP*, 2004, pp. 2004–29.
- [17] Wei-Ning Hsu et al., “Hubert: Self-supervised speech representation learning by masked prediction of hidden units,” *IEEE TASLP*, vol. 29, pp. 3451–3460, 2021.
- [18] P. Yadav, D. Tam, L. Choshen, C. Raffel, and M. Bansal, “TIES-merging: Resolving interference when merging models,” in *NeurIPS*, 2023.