The Smallest Recognizable Transformer: A Literature Review from *Attention Is All You Need* to a Modern Block, Read in the Universal Model

Claude and MJC

2026-11-14

Abstract

This track trains a small standard transformer on prefixes of enwik9, re-expresses it as a UM $u = (E, T, P, f, \omega)$ in the sense of the CMP paper that behaves the same up to a small, measured translation error, and publishes both, with every pattern open to inspection. This review covers the work that fixes the three choices we have to make before writing any code: what counts as a *transformer* (the 2017 architecture and the decoder-only block it became); what counts as *typical transformer behavior* (previous-token heads, induction heads and in-context learning, attention sinks, key–value memories in the MLP, superposition); and what is the *smallest* model that still has all of it (two layers, several heads, an MLP, byte-level input). For each component we give its UM reading. Every softmax is an event space: the output softmax is the byte ES, and every attention head at every query position has an ES whose events are "this head reads from source position j." ReLU is the doubled-E binary ES of the RNN mapping, with nothing changed. RMSNorm is the one component with no direct UM counterpart. The review ends with the architecture we train, called tt-2 below.

1 What this review has to decide

Three things are fixed before the first line of training code is written.

1. **Recognizability.** A reader who knows transformers should recognize the model as one without being told: token embedding, a residual stream, multi-head causal self-attention, a position-wise MLP, normalization, an unembedding to a softmax over the vocabulary, trained by next-token cross-entropy with Adam-family optimization. Sections 2–4.

2. **Typical behavior.** The model must show the behaviors that interpretability work has found in transformers of every size, so that what we find in it says something about large models. Section 6.

3. **Minimality.** Within those two constraints, as small as possible: every extra parameter is something the UM translation has to account for. Section 5.

Section 7 reviews mechanistic interpretability of small transformers, the work this track is in conversation with. Section 8 gives each component its UM reading, and Section 9 states the resulting architecture.

UM notation follows the CMP paper and is not restated: $u = (E, T, P, f, \omega)$, T is log support in $[0, 255]$, softmax within each ES, and the standard update $(f_p(t))_j = \max_i \min(t_i, p_{ij})$. The RNN precedent is the doubled-E mapping (`/hutter/archive/20260131/rnn-um-mapping.pdf`), which re-expressed a 128-unit Elman RNN as a UM with 0.00% bpc difference.

2 The 2017 architecture

Attention as a learned soft alignment over source positions was introduced for translation by Bahdanau, Cho and Bengio [1], on top of a recurrent encoder. Vaswani et al. [2] removed the recurrence. Their *Transformer* is an encoder–decoder stack of $N = 6$ identical layers, each consisting of

- multi-head scaled dot-product attention, $\mathrm{Attn}(Q, K, V) = \mathrm{softmax}(QK^\top/\sqrt{d_k})\,V$, with $h = 8$ heads of width $d_k = 64$ whose outputs are concatenated and projected;
- a position-wise feed-forward network $\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$ with inner width $d_{ff} = 4d_{model}$;
- a residual connection around each sublayer, followed by layer normalization [3] (*post-LN*: $x \leftarrow \mathrm{LN}(x + \mathrm{Sublayer}(x))$), the residual idea taken from He et al. [4];
- sinusoidal absolute position encodings added to the input embeddings;
- a causal mask in the decoder, so position i attends only to $j \le i$.

Training used Adam with a linear warmup followed by inverse-square-root decay, label smoothing, and dropout. The paper's argument for attention over recurrence was path length: any two positions are one attention step apart, against $O(n)$ steps in an RNN, and all positions are processed in parallel.

3 Decoder-only language models and byte-level text

Decoder-only. GPT [5] kept only the decoder (masked self-attention, no cross-attention) and trained it as a left-to-right language model. GPT-2 [6] moved layer normalization to the input of each sublayer and added a final LN before the unembedding, scaled residual initialization by depth, and tied the input and output embeddings, a practice introduced by Press and Wolf [7]. Every model discussed below descends from this decoder-only form, and so does ours: predicting the next byte is exactly the Hutter-prize task.

Pre-LN. Xiong et al. [8] showed that post-LN puts large gradients near the output at initialization, which is why it needs warmup, while pre-LN ($x \leftarrow x + \mathrm{Sublayer}(\mathrm{LN}(x))$) keeps gradients well scaled and trains with little or no warmup. In pre-LN the residual stream is never normalized in place; it is a running sum that each sublayer reads (through a norm) and writes to (additively). Elhage et al. build their whole interpretability framework on this fact (Section 7).

Character and byte level on enwik8. Enwik8, the first 10^8 bytes of the enwik9 file we use, has been a standard character-level benchmark since before transformers. Al-Rfou et al. [9] trained a 64-layer character transformer (235M parameters, 512-byte context, with auxiliary losses at intermediate layers and positions) to 1.06 bpc on enwik8, ahead of the best LSTMs of the time. Transformer-XL [10] added segment-level recurrence (caching the previous segment's hidden states) and relative position encodings, reaching 1.03 bpc with 12 layers and 0.99 bpc with 24 layers (277M parameters). Both results show that transformers work on raw bytes with no tokenizer.

Transformers as compressors. Bellard's NNCP [11] is a lossless compressor whose model is a transformer trained online as it compresses; it has been at or near the top of the Large Text Compression Benchmark on enwik9. Delétang et al. [12] make the equivalence explicit: a sequence model plus an arithmetic coder is a compressor, and the model's log-loss is the code length. For us the log-loss in bits per byte is the same quantity the Hutter track already measures.

4 The modern block

Between 2019 and 2023 every component of the block was replaced at least once. LLaMA [13] is the usual reference point because it gathered the surviving choices into one open recipe that most later open models copy. Table 1 summarizes; the paragraphs give the reasons.

Component	2017 Transformer	Modern consensus (LLaMA-style)
Stack	encoder–decoder	decoder-only
Norm placement	post-LN	pre-norm, plus a final norm
Norm	LayerNorm (center + scale, bias)	RMSNorm (scale only)
FFN	ReLU, $4d$	SwiGLU, $\approx \frac{8}{3}d$ (so params match $4d$ ReLU)
Positions	sinusoidal absolute, added	RoPE (rotary, relative) in Q, K
KV heads	one per query head	grouped (GQA) or one (MQA) at scale
Biases	everywhere	none
Embeddings	tied (in+out)	tied in small models, untied in large
Optimizer	Adam, inverse-sqrt	AdamW, warmup + cosine decay
Regularization	dropout, label smoothing	none at pretraining scale

Table 1: The block, then and now.

Normalization. RMSNorm [14] drops LayerNorm's mean subtraction and bias: $\mathrm{RMSNorm}(x) = g \odot x/\sqrt{\frac{1}{d}\sum_k x_k^2 + \epsilon}$. It is cheaper and empirically as good. More recent large models also normalize queries and keys before the dot product (*QK-norm* [15]), which bounds attention logits and stabilizes training at scale.

Feed-forward. GELU [16] replaced ReLU in GPT and BERT. Shazeer [17] showed that gated linear units, $\mathrm{FFN}_{\mathrm{SwiGLU}}(x) = (\mathrm{Swish}(xW) \odot xV)\, W_2$, improve perplexity at matched parameter count; the inner width is reduced to $\approx \frac{8}{3}d$ to compensate for the third matrix. Geva et al. [18] read the FFN as a key–value memory: rows of W_1 are keys that detect patterns in the input, and the matching rows of W_2 are values that push the output toward particular tokens.

Positions. Sinusoidal and learned absolute embeddings gave way to relative schemes. Transformer-XL [10] used learned relative terms; RoPE [19] rotates each 2-dimensional slice of q and k by an angle proportional to position, so that $q_i \cdot k_j$ depends on $i - j$ only; ALiBi [20] adds no embedding at all and instead adds a fixed per-head penalty $-m_h(i - j)$ to the attention logits, which extrapolates to longer contexts. RoPE is the modern default.

Attention heads. Multi-query attention [21] shares one key/value head among all query heads, and grouped-query attention [22] shares one per group; both shrink the inference KV cache and matter only at scale. FlashAttention [23] changes how attention is computed, not what is computed. Xiao et al. [24] found that trained models put large attention mass on the first tokens regardless of content (*attention sinks*): when a head has nothing to read, the softmax still has to put its mass somewhere.

Optimization. AdamW [25] (decoupled weight decay), linear warmup, and cosine decay to a small fraction of peak are standard. Newer optimizers such as Muon [26], which orthogonalizes the update for matrix parameters, are used at small scale (for example in the nanoGPT speedruns) but are not yet a settled standard; we treat them as out of scope.

Left out on purpose. Mixture-of-experts layers [27, 28] and state-space or linear-attention replacements are either additions that matter only at scale or are not transformers. Dropout and label smoothing are omitted in modern pretraining because a single pass over abundant data does not overfit; our setting is the same.

5 Scale: how small can it be?

Scaling laws. Kaplan et al. [29] found loss to be a power law in parameters, data and compute, with shape (depth versus width) mattering little at fixed parameter count over a wide range. Hoffmann et al. [30] (Chinchilla) corrected the compute-optimal ratio to about 20 training tokens per parameter. For a model of

10^5 parameters that is 2×10^6 bytes; enwik9 prefixes of 10^7 bytes and more let us train well past compute-optimal, which is what small deployed models do.

Small models that still behave like large ones. TinyStories [31] showed coherent multi-paragraph generation from models of a few million parameters, and as few as one layer, when the data distribution is narrow enough. Karpathy's nanoGPT character-level Shakespeare model (6 layers, 384 wide, about 10M parameters) is the usual teaching example. The interpretability literature works much smaller: one- and two-layer models, often attention-only, with d_{model} in the hundreds, and toy models with a handful of dimensions [32, 33, 34].

The floor is two layers. Elhage et al. [32] showed analytically what each depth can do. A zero-layer transformer (embed, unembed) is a bigram table. A one-layer attention-only transformer is a bigram table plus a sum of *skip-trigrams* ("A somewhere earlier, B now $\Rightarrow$ C next"), one family per head. Only from two layers can heads *compose*: a head in layer 2 can form its query, key or value from what a layer-1 head wrote into the residual stream. K-composition with a previous-token head gives the *induction head* ("$[A][B] \ldots [A] \Rightarrow [B]$"), which Olsson et al. [33] identify as the mechanism behind most in-context learning, and which forms in a sudden phase change visible as a bump in the training loss. One layer cannot implement it. So the smallest model with the central transformer behavior has two layers.

6 Typical transformer behaviors

These are the behaviors we will require tt-2 to show before interpreting it; each has a measurable signature at our scale.

1. **Previous-token heads**: a layer-1 head whose attention puts most of its mass on $j = i - 1$.
2. **Induction heads**: a layer-2 head that, on a repeated random byte sequence, attends from the second occurrence of A to the byte after the first occurrence, and copies it [33]. Signature: loss on the second copy of a repeated sequence far below the first.
3. **In-context learning score**: loss at late context positions lower than at early ones, on natural text, by a margin that appears with the induction-head phase change [33].
4. **Attention sinks**: heads that park attention on position 0 (or on a frequent byte such as space or newline) when they have nothing to read [24].
5. **Key–value memories**: MLP neurons whose input direction detects a context and whose output direction promotes specific next bytes [18].
6. **Superposition**: more features than neurons, so individual neurons are polysemantic [34, 35].

7 Mechanistic interpretability of small transformers

The circuits framework. Elhage et al. [32] treat the residual stream as a shared linear channel. Each attention head factors into a *QK circuit* $W_E^\top W_Q^\top W_K W_E$ (which source tokens a query token attends to) and an *OV circuit* $W_U W_O W_V W_E$ (what attending to a source token does to the output logits). Because the stream is a sum, an attention-only model expands into a sum over *paths*: the direct path, one path per head, and *virtual heads* through pairs of heads in different layers. Each path is a separately readable bilinear map from input tokens to output logits.

Specific circuits. Indirect-object identification in GPT-2 small was traced to 26 heads in seven functional classes [41]. Nanda et al. [42] fully reverse-engineered a one-layer transformer trained on modular addition (it uses discrete Fourier features and trigonometric identities) and used the mechanism to define progress measures across grokking. ACDC [43] automates circuit search by activation patching. ROME [44] localizes and edits factual associations in mid-layer MLPs, consistent with the key–value reading.

Features and superposition. Toy models [34] show that networks store more sparse features than they have dimensions by placing them in nearly orthogonal directions, at the cost of interference. Sparse autoencoders trained on activations recover many monosemantic features from a one-layer transformer [35, 36] and, at scale, from a production model [37]. Transcoders [38] replace an MLP with a wider sparse MLP trained to imitate it, and cross-layer transcoders underlie *attribution graphs* [39, 40]: a *replacement model* that matches the original's outputs closely, whose remaining error is carried as explicit error nodes, and whose computation on a given prompt is a sparse graph of feature-to-feature edges.

Programs in and out of transformers. RASP [45] is a programming language whose primitives (select, aggregate, elementwise maps) are what a transformer layer computes. Tracr [46] compiles RASP programs into transformer weights, the synthesis direction. Friedman et al. [47] train transformers constrained so that they can be decompiled into discrete programs. On the theory side, hard (argmax) attention gives Turing-completeness constructions and expressivity limits [48], and *saturated* attention, the limit where the softmax becomes a uniform choice among the maximizing positions, puts transformers in constant-depth threshold circuits [49].

What this track adds. The replacement-model idea is the closest point of contact: attribution graphs replace the transformer with something more readable, measure the error, and read the graph. Three things are different here. First, the target representation is not an auxiliary dictionary trained beside the model but a UM, a complete model on its own that carries both the patterns and the event spaces. Second, we replace the whole model, not one prompt's computation, and measure the translation error in bits per byte over held-out data. Third, the CMP paper's equivalence thesis (CMP, *The Equivalence of Interpretability and Efficiency*) predicts that the right factorization is also the cheaper one: as events get names, the pattern set should get sparser and the model smaller, not merely clearer. The saturated-attention result says the same thing from the other side: in the hard limit, softmax attention becomes a max, which is where f_p already lives.

8 The transformer read in the UM

The RNN mapping established the method: every softmax marks an ES, other nonlinearities are rewritten as ES competitions, and a component that resists this is where the work is. Table 2 applies it to every component of the modern block.

Attention is an ES over time. This is the main structural point. The RNN had to have its ESs discovered inside an unstructured hidden vector. A transformer declares one ES per head, per layer, per query position, with named events: "read from position j." The induction head becomes two chained patterns: layer 1, offset-1 event $\Rightarrow$ "previous byte was A" written at i; layer 2, (current byte is A) $\wedge$ (source j's previous byte was A) $\Rightarrow a_{ij}$, then $a_{ij}\wedge$ (byte at j is B) $\Rightarrow$ predict B. That is a pattern chain in the sense of the 2026-02-08 pattern-chains paper, with time offsets as events. Elhage et al.'s path expansion is the same statement in linear-algebra terms: each path through the residual stream is a chain of patterns.

Where the work is. Three components are not exact on first reading. (i) The OV write is an expectation under the attention ES; under saturation it becomes the copy that f_p performs, and the gap between them is measurable per head. (ii) SwiGLU's multiplicative gate is a conjunction; the CMP paper's Boolean-circuits appendix says AND costs depth when patterns are positive, so the gate either maps to an explicit intermediate event or carries a translation cost. (iii) RMSNorm divides by a statistic of the whole residual vector. In log support, multiplying every logit downstream by $1/\text{rms}$ is a temperature: it changes how sharp each downstream ES is without changing the order of its events. It has no f_p form. We will either fold it into a per-position *certainty* event or measure what removing it costs.

Component	UM reading	Status
Byte input	input ES of 256 events, "the byte at i is x"	exact
Output softmax	output ES of 256 events; logits are log support	exact (as RNN)
Attention softmax, head h, layer ℓ, query i	an ES $A_i^{\ell h}$ over source positions $\{0..i\}$; event a_{ij} = "head h at i reads from j"	exact
QK logit $q_i \cdot k_j$	log support for a_{ij}, a bilinear (pairwise-AND) pattern from (feature of i, feature of j) to a_{ij}	exact in floats; tropical cost to measure
OV write	pattern from (a_{ij}, feature of j) to feature of i; under hard attention a copy	expectation under the ES, not max
RoPE / ALiBi	events "offset $i - j = k$" with log support into a_{ij}; ALiBi is literally additive log support $-m_h k$	ALiBi exact; RoPE mixes offset with content
ReLU MLP	doubled-E binary ES per neuron, $t(n) = 2\max(0, x)$, as RNN	exact
SwiGLU MLP	product of two features: an AND (min in f_p) gating a value	smooth; needs its own mapping
Residual stream	skip patterns; each sublayer adds support	exact
RMSNorm	divides all coordinates by one per-position scalar: a context-dependent temperature, not f_p	**no direct counterpart**

Table 2: Each component's UM reading. "Exact" means a real-valued UM reproduces it with no error, as the doubled-E mapping did for the RNN; the tropical (max-min, $[0, 255]$) form is a second step with its own measured cost (see `/hutter/archive/20260312/price-of-tropical-semiring.pdf`).

9 The architecture: tt-2

The constraints above leave little freedom. We call the model **tt-2** (tiny transformer, 2 layers).

Two choices depart from the most common modern recipe, and each has a variant we train beside it so the cost is measured rather than argued:

- **ReLU, not SwiGLU.** ReLU has an exact UM reading and SwiGLU does not. **tt-2-swiglu** ($d_{ff} = 176$) measures the perplexity cost of the choice and gives a model on which to develop the gate mapping.
- **RMSNorm kept.** It is the hard component, and large models have it, so the interpretation has to deal with it. **tt-2-nonorm** measures what dropping it costs in perplexity.

A one-layer **tt-1** serves as the control for Section 6: it should show previous-token and skip-trigram behavior and no induction heads.

Training runs in C, as the sat-rnn did, on two CPU cores. At $6ND \approx 7 \times 10^{12}$ FLOPs for 10^7 bytes, one epoch takes minutes. The next paper reports training and the behavior checklist; the one after reports the UM translation.

References

[1] D. Bahdanau, K. Cho, Y. Bengio. Neural machine translation by jointly learning to align and translate. ICLR 2015.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin. Attention is all you need. NeurIPS 2017.

[3] J. L. Ba, J. R. Kiros, G. E. Hinton. Layer normalization. arXiv:1607.06450, 2016.

[4] K. He, X. Zhang, S. Ren, J. Sun. Deep residual learning for image recognition. CVPR 2016.

[5] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever. Improving language understanding by generative pre-training. OpenAI, 2018.

[6] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever. Language models are unsupervised multitask learners. OpenAI, 2019.

Choice	tt-2	Reason
Vocabulary	256 bytes, no tokenizer	Hutter task; byte ES; tokenization is itself a factor map we leave inside the model
Layers L	2	smallest depth with head composition and induction (Sec. 5)
Heads H	4 per layer, $d_{head} = 16$	room for previous-token, induction, and sink heads to separate
d_{model}	64	
MLP	ReLU, $d_{ff} = 256$	key–value memories; ReLU is the exact doubled-E mapping
Norm	pre-RMSNorm + final RMSNorm	modern standard; the hard component, kept on purpose
Positions	RoPE	modern standard
Biases	none	modern standard; also removes a constant-support event per unit
Embeddings	tied	standard at this size
Context n	128 bytes	long enough for in-context repeats inside one window
Parameters	$\approx 1.1 \times 10^5$	16K embedding + 2 × (16K attention + 32K MLP)
Data	enwik9 bytes $[0, 10^7)$ train, $[10^7, 10^7{+}10^6)$ held out	about 90 bytes per parameter
Optimizer	AdamW, warmup, cosine	standard
Metric	held-out bpc; translation error	translation error = bpc(UM) − bpc(tt-2), and mean KL(tt-2 ‖ UM) per byte

Table 3: tt-2.

[7] O. Press, L. Wolf. Using the output embedding to improve language models. EACL 2017.

[8] R. Xiong et al. On layer normalization in the Transformer architecture. ICML 2020.

[9] R. Al-Rfou, D. Choe, N. Constant, M. Guo, L. Jones. Character-level language modeling with deeper self-attention. AAAI 2019.

[10] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, R. Salakhutdinov. Transformer-XL: attentive language models beyond a fixed-length context. ACL 2019.

[11] F. Bellard. NNCP v2: lossless data compression with Transformer. 2021. `https://bellard.org/nncp/`

[12] G. Delétang et al. Language modeling is compression. ICLR 2024.

[13] H. Touvron et al. LLaMA: open and efficient foundation language models. arXiv:2302.13971, 2023.

[14] B. Zhang, R. Sennrich. Root mean square layer normalization. NeurIPS 2019.

[15] M. Dehghani et al. Scaling vision transformers to 22 billion parameters. ICML 2023.

[16] D. Hendrycks, K. Gimpel. Gaussian error linear units (GELUs). arXiv:1606.08415, 2016.

[17] N. Shazeer. GLU variants improve Transformer. arXiv:2002.05202, 2020.

[18] M. Geva, R. Schuster, J. Berant, O. Levy. Transformer feed-forward layers are key-value memories. EMNLP 2021.

[19] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, Y. Liu. RoFormer: enhanced Transformer with rotary position embedding. arXiv:2104.09864, 2021.

[20] O. Press, N. A. Smith, M. Lewis. Train short, test long: attention with linear biases enables input length extrapolation. ICLR 2022.

[21] N. Shazeer. Fast Transformer decoding: one write-head is all you need. arXiv:1911.02150, 2019.

[22] J. Ainslie et al. GQA: training generalized multi-query Transformer models from multi-head checkpoints. EMNLP 2023.

[23] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, C. Ré. FlashAttention: fast and memory-efficient exact attention with IO-awareness. NeurIPS 2022.

[24] G. Xiao, Y. Tian, B. Chen, S. Han, M. Lewis. Efficient streaming language models with attention sinks. ICLR 2024.

[25] I. Loshchilov, F. Hutter. Decoupled weight decay regularization. ICLR 2019.

[26] K. Jordan et al. Muon: an optimizer for hidden layers in neural networks. 2024. `https://kellerjordan.github.io/posts/muon/`

[27] N. Shazeer et al. Outrageously large neural networks: the sparsely-gated mixture-of-experts layer. ICLR 2017.

[28] W. Fedus, B. Zoph, N. Shazeer. Switch Transformers. JMLR 2022.

[29] J. Kaplan et al. Scaling laws for neural language models. arXiv:2001.08361, 2020.

[30] J. Hoffmann et al. Training compute-optimal large language models. NeurIPS 2022.

[31] R. Eldan, Y. Li. TinyStories: how small can language models be and still speak coherent English? arXiv:2305.07759, 2023.

[32] N. Elhage et al. A mathematical framework for Transformer circuits. Transformer Circuits Thread, 2021.

[33] C. Olsson et al. In-context learning and induction heads. Transformer Circuits Thread, 2022.

[34] N. Elhage et al. Toy models of superposition. Transformer Circuits Thread, 2022.

[35] T. Bricken et al. Towards monosemanticity: decomposing language models with dictionary learning. Transformer Circuits Thread, 2023.

[36] H. Cunningham, A. Ewart, L. Riggs, R. Huben, L. Sharkey. Sparse autoencoders find highly interpretable features in language models. ICLR 2024.

[37] A. Templeton et al. Scaling monosemanticity: extracting interpretable features from Claude 3 Sonnet. Transformer Circuits Thread, 2024.

[38] J. Dunefsky, P. Chlenski, N. Nanda. Transcoders find interpretable LLM feature circuits. NeurIPS 2024.

[39] E. Ameisen et al. Circuit tracing: revealing computational graphs in language models. Transformer Circuits Thread, 2025.

[40] J. Lindsey et al. On the biology of a large language model. Transformer Circuits Thread, 2025.

[41] K. Wang, A. Variengien, A. Conmy, B. Shlegeris, J. Steinhardt. Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. ICLR 2023.

[42] N. Nanda, L. Chan, T. Lieberum, J. Smith, J. Steinhardt. Progress measures for grokking via mechanistic interpretability. ICLR 2023.

[43] A. Conmy, A. Mavor-Parker, A. Lynch, S. Heimersheim, A. Garriga-Alonso. Towards automated circuit discovery for mechanistic interpretability. NeurIPS 2023.

[44] K. Meng, D. Bau, A. Andonian, Y. Belinkov. Locating and editing factual associations in GPT. NeurIPS 2022.

[45] G. Weiss, Y. Goldberg, E. Yahav. Thinking like Transformers. ICML 2021.

[46] D. Lindner et al. Tracr: compiled Transformers as a laboratory for interpretability. NeurIPS 2023.

[47] D. Friedman, A. Wettig, D. Chen. Learning Transformer programs. NeurIPS 2023.

[48] M. Hahn. Theoretical limitations of self-attention in neural sequence models. TACL 2020.

[49] W. Merrill, A. Sabharwal, N. A. Smith. Saturated Transformers are constant-depth threshold circuits. TACL 2022.