A Low-Level Representation for Transformer Computation

Noah Rousell
Brown University

Abstract: Language model algorithms are represented with an ad-hoc combination of natural language, algebra, stylized graphs, and visualizations. The lack of a standardized formal representation to represent model algorithms is an impediment to uncovering more precise and complete circuits. In this work, I introduce TLang, a representation that can can express model computation as low-level operations over features, as well as a simple procedure to compile written TLang algorithms to model weights. I also introduce higher level primitives inspired by the feature geometries exhibited in much of model computation. I precisely represent idealized versions of the induction and indirect object identification circuits reverse-engineered by previous works in TLang, compile them to weights, and verify their accuracy.

1 Introduction

Deep Learning methods have been applied to tasks in language modeling [1], computer vision [2], and biology [3] with increasing success, yielding models that often generalize well beyond their training distribution. This has motivated the field of mechanistic interpretability, which aims to reverse-engineer the algorithms, often called ”circuits” [4], by which models accomplish such feats. Previous work has uncovered circuits behind modular [5] and regular[6] arithmetic, in-context learning[7], indirect object identification [8], factual recall[9], linebreak prediction [10], and many other tasks.

Currently, algorithms are represented and explained via a combination of natural language, algebra [5], stylized graphs[9], and visualizations[10] of model representations. This ad-hoc approach has several drawbacks:

- **Precision**. Algorithms are specified at a fairly high level, with the result that they do not give a firm understanding of the low-level computation with which each component accomplishes its proposed role. Consider the algorithm discovered to accomplish the Indirect Object Identification (IOI) task in GPT-2 Small (which is responsible for completing prompts of the form ”When John and Mary went to the store, John gave a drink to ” with ”Mary” instead of ”John”)[8]. In the IOI circuit, the ”Name Mover” attention heads are

explained to attend to previous name tokens that haven't been marked by a previous "Inhibition Head", and copy this name to be predicted. This explanation does not help us answer questions such as "How does the head identify which tokens are names?" and "How does the model copy the name to be predicted?".

- **Completeness.** With vague algorithm specifications, we also run the risk of not capturing all of the relevant computation. To illustrate, consider the induction circuit[7], which up-weights the probability of the second token of observed bigrams in the context via two attention heads in series. The typical explanation is as follows: the first attention head attends to and marks the previous token for every token in the sequence. The second attention head attends to tokens for which it is the previous token, and then up-weights the probability of predicting that token. Yet, upon rigorous measurement[11], this explanation was found to explain only 35% of the improvement in loss due to the induction head on an induction-heavy dataset. In order to explain more of the improvement, it was required to expand this explanation in ways such as allowing the previous token head to attend to the previous three tokens and incorporating two other heads upstream of the induction head that attend to tokens that are the same as the current token. This level of rigor in validating explanations of algorithms is quite rare, presumably because of the high amount of work involved to design experiments and the incentive not to undermine the credibility of a proposed explanation. Note that completeness is similar to, but distinct from, precision, as a precise explanation of induction might explain how the previous token was attended to and copied, but still fail to account for the other relevant attention heads.

- **Tool for Thought.** The algorithms learned by models are often fairly complex. Furthermore, the algorithms learned by models are markedly different from algorithms researchers encounter in other settings, such as programming. While algorithms specified in programming languages such as Python contain a single or handful of threads of execution, discrete control flow, and exact computations, model algorithms are smoother and often consist of many components computing approximate values by acting in parallel[6] [12]. Reasoning about complex computer programs is also a challenging mental endeavor, but programmers are able to represent and manipulate programs using standardized abstractions with semantics they have become deeply familiar with over time. We have yet to develop analogous abstractions that enable us to reason more productively about these challenging aspects of model computation. An important difference between the abstractions that underpin programming and potential abstractions for model computation is that, excluding considerations such as hardware, programming abstractions were designed by humans, while useful abstractions for model computation will have to be largely derived out of the structure inherent in neural network architectures as well as the implicit operations models converge to during the learning process.

Our current explanations for model algorithms are imprecise, incomplete, and hard to understand (beyond small, idealized circuits) because model computation is complicated, verifying ad-hoc explanations is time-consuming, and we lack good abstractions to reason about model computation. It is possible to make progress on all three fronts by designing a *formal represen-*

tation of transformer computation. Due to the "messy" nature of model computation, this is a challenging task, but has many potential benefits:

- Specifications of algorithms in a precise representation could allow **automated measurement of algorithmic faithfulness** to target models, as well as further mechanistic analysis and control, such as the ability to "reprogram"-and-patch model algorithms.
- Algorithms written in this representation can be **compiled to semi-realistic model weights**. This could be used to test unsupervised decomposition or circuit-discovery methods with ground-truth computation known.
- Such a representation could act as a "**decompilation target**" for reverse-engineering methods. This may be important in scaling up interpretability to larger models.
- Such a representation could act as a tool for thought, enabling **better explanation of model algorithms** and better reasoning about what computation is natural for models to implement.

In this work, I make a first step towards a low-level representation that might enable these improvements. My contributions are as follows:

- I introduce a low-level representation, TLang, that allows full representation of RoPE transformers with ReLU activations.
- I introduce a simple procedure to compile algorithms expressed in TLang to model weights, grounding the representation in precise semantics.
- I showcase several known circuits from the literature in TLang.

An accompanying Python library, as well as code to reproduce experiments and visualizations, will be released shortly.

2 Related Work

Transformer Computation Representations and Compilers. Previous works have proposed algorithmic languages meant to mirror some aspects of transformer computation, as well as compilers to convert written algorithms to transformer weights. The most notable language is RASP[13], along with the accompanying Tracr compiler[14]. These works predominantly focus on ergonomics while authoring algorithms at the expense of the ability to represent realistic model computation. In contrast, the current work aims to introduce a representation capable of representing realistic model computation by allowing features in superposition, supporting Rotary Positional Embedding (RoPE), and employing more-expressive lower-level primitives.

Feature Geometry. A central hypothesis of mechanistic interpretability is the Linear Representation Hypothesis (LRH)[15], which posits that much of the information encoded by models is linearly decodable. One explanation for this is that models represent information as *features* that have geometries amenable to linear decoding. The classical conception of a

feature is a one-dimensional subspace whose magnitude represents the intensity or presence of a concept. However, there is increasing evidence that models use more sophisticated geometries to represent information, and it is useful to consider these as features as well. Models have been found to employ circular geometries when representing concepts with a cyclical pattern such as the months of the year[16], polytopes to represent categorical concepts such as types of animals or the language of text[17], and helices or one-dimensional manifolds with high curvature to represent continuous values (without a cyclical pattern)[18] [10]. These geometries have been hypothesized to arise due to the co-occurrence statistics of the concepts they represent[19]. They also come with attractive representational and computational properties, such as rotation on a circle cleanly corresponding to incrementing a feature representing "`April`" to "`May`".

3 Representation

In this section I will explain key design decisions and introduce the representation, TLang, with a simple example.

3.1 Design Goals

Expressivity. This representation was created with the aim of being able to represent all realistic model computation. Due to the highly expressive nature of model computation, this motivates a low-level representation close to the operations being performed by the architecture. Yet, for ease of understanding and authoring algorithms in the representation, it is useful to have higher level abstractions that represent larger idealized primitives of computation. To satisfy both of these goals, a multi-level approach is taken. The base representation is a low-level representation that can fully express arbitrary RoPE/ReLU transformer computation, albeit verbosely. To limit verbosity while authoring algorithms, I introduce some higher level primitives which decompose to sets of low-level primitives.

Feature-first. Much of mechanistic interpretability is predicated on the existence of features. TLang casts all computation in terms of a set of features. Furthermore, high-level primitives generalize to more complicated feature geometries, such as one-dimensional curved manifolds to represent continuous variables[19], and polytopes[17] to represent categorical variables. Framing computation in terms of features also make future integration with activation decomposition methods such as Sparse Autoencoders (SAEs)[20] and Cross-Layer Transcoders (CLTs)[21] straightforward.

3.2 Low-Level Representation

TLang consists of `Features` and `Operations`. There are two operations, `Head` and `Neuron`, which represent a single attention head and MLP neuron respectively. `Operations` are composed of multiple `Exprs`, which represent a single read-in or write-out direction to the residual stream.

We will introduce these central primitives by constructing a simple algorithm that detects if both the previous and current tokens are names, and, if so, writes a value signaling to predict another name.

3.2.1 `Features`

The core "variables" of TLang are `Features`. In low-level TLang, `Features` are named, 1-dimensional subspaces in the residual stream. In the following example, we use the Python library to create three features, `prev_tok_is_name`, `curr_tok_is_name`, and `predict_name`.

```
prev_tok_is_name = Feature("prev_tok_is_name")
curr_tok_is_name = Feature("curr_tok_is_name", embed: lambda t: 1 if
↪  is_name(t) else 0)
predict_name     = Feature("predict_name")
```

Users are not required to define a concrete direction while initializing `Features`. `Features` can be initialized abstractly without a direction, with the direction to be populated later. `Features` have an associated real value at each point in the residual stream, obtained by taking the dot-product of their direction with the residual stream at that point. If a roughly-orthogonal set of directions is chosen for the `Features`, there will be minimal interference.

Each `Feature` has a corresponding value in the embed and unembed weight matrices for each token. In the snippet above, we specify that tokens for which the `is_name` predicate returns true will have a value of 1.0 in the embed. The value of a feature at the embed and unembed is 0 by default.

3.2.2 `Exprs`

We can construct `Exprs` over sets of `Features`. The simplest example of an `Expr` is simply a `Feature` with coefficient 1.0, e.g., `1.0 * prev_tok_is_name`. We can also combine features, e.g., `prev_tok_is_name + curr_tok_is_name`. This allows reading and writing to multiple `Features` at once. For example, this makes AND gates easy to express.

`Exprs` are constrained to be *linear* to match the expressivity of reads and writes to the residual stream.

3.2.3 The `Head` Operation

We will now construct an attention head that attends to the previous token and writes to `prev_tok_is_name` with the value of that token's `curr_tok_is_name`.

Attention heads can be considered to consist of two circuits, the *QK*-circuit, which determines the attention pattern of the head, and the *OV*-circuit, which determines the contribution to the residual stream of each source token [22]. In modern language models, Rotary Position Embeddings (RoPE)[23] are used to encode the relative position of queries and keys during

attention via pairwise rotations of query and key dimensions at different frequencies. Language models can learn to place queries and keys at particular phases relative to each other to achieve a peak in attention contribution at desired offsets. The *QK*-circuit for a head in TLang can be specified as a list of `RoPePair` objects, each of which specifies a query `Expr`, key `Expr`, relative phase offset, and dimension pair index. If the phase offset and dimension pair are omitted, the receptivity to relative position is assumed unimportant, e.g. `RoPEPair(q=f1, k=f2 + f3)`.

In this example, we make use of a helper function, `n_tokens_back`, to construct the `RoPEPairs` needed to attend to the previous token.

```
constant = Feature("constant", embed: 1)
prev_head = Head(
    qk=n_tokens_back(1, 10 * constant),
    ov_in=[curr_tok_is_name]  # reads from the source (previous token)
    ov_out=[prev_tok_is_name] # writes to the destination (current token)
)
```

We created a `constant` feature to use in the offset match. Multiplying it by 10 yields a sharper attention pattern after the softmax is applied. The `ov_in` and `ov_out` fields each take in a list. These (same-length) lists are considered in parallel, with the *i*th entry of each constituting a single read-in, write-out pair. In this case, there is only a single entry, but we will see later that there can be multiple read-in, write-out channels within a single `OV` circuit.

3.2.4 The `Neuron` Operation

After the first head, the residual stream for the current token should contain 1.0 for both `curr_tok_is_name` (from the embed) and `prev_tok_is_name` (from `prev_head`) when the current and previous tokens are names. We can use a single `Neuron` to perform an AND operation, writing a 1.0 to `predict_name` only if both `curr_tok_is_name` and `prev_tok_is_name` have a value of 1.0.

TLang views MLPs as soft key-value maps, where the key is a linear read from the residual stream, the threshold and ReLU determine whether the key is matched strongly enough to fire, and the value is a linear write out to the residual stream. With this view, we can represent an AND as a sum of both input features with a threshold of 1.0.

```
both_names_neuron = Neuron(
    inp=prev_tok_is_name + curr_tok_is_name,
    threshold=1.0,
    out=predict_name
)
```

3.3 High-Level Geometric Primitives

Low-level TLang is expressive, but also verbose. There is often larger structure in algorithms that can be employed to design higher-level representational and computational abstractions. Specifically, when feature geometries are involved in a circuit, we can replace much of the low-level TLang with primitives representing the feature geometries and the computations on them. These high-level features and their operations have natural decompositions to low-level TLang `Features` and operations.

3.3.1 `Categorical` Features

`Categorical` features are an abstraction over polytopes. They can be created in the Python library by passing in a list of categories and the dimension of the subspace the polytope will live in.

```
cat  = Categorical("cat", ["a", "b", "c"], n=3)
cat2 = Categorical("cat2", [1, 2, 3], n=3)

cat[1] # returns the (low-level) Feature for "b"
cat.all() # returns a list containing the (low-level) Features for "a", "b", "c"
cat.match(cat2) # returns a list of RoPEPairs matching each item from cat (key-side) to the same
↪  indexed item from cat2 (query-side)
```

See Sections 5.1 and 5.2 for concrete examples employing `Categorical` features.

3.3.2 `Range` Features

`Range` features are an abstraction over one-dimensional manifolds with high curvature. These manifolds have been found to be used by models to represent continuous values[10] [16] [19]. By representing quantities on a curved manifold, the quantity can be encoded in the distance along the manifold, and read off via the curvature. This allows the magnitude to stay relatively uniform across the manifold. Comparing two manifolds is also a natural operation, as the model can simply rotate the two manifolds so that they align at whatever offset is desired, e.g. via the W_Q and W_K matrices of an attention head.

TLang uses the analytical form for the geometry derived in [19], where the point c on the manifold is encoded as a direct sum of Fourier modes,

$$R[c] = \bigoplus_{p=1}^{k} \sqrt{\lambda_p} \cdot (\cos(\theta_p c), \sin(\theta_p c))$$

where $\theta_p = 2\pi p/N$ and each amplitude is the Discrete Fourier Transform (DFT) of a desired kernel f at frequency p, $\lambda_p = \hat{f}(p)$. This construction yields the property that the dot product of two points on the manifold conforms to the desired kernel,

$$R[c_1] \cdot R[c_2] \approx f(c_1 - c_2)$$

TLang uses a gaussian kernel by default.

```
r1 = Range("char_count", 150, k=4)
r2 = Range("line_width", 150, k=4)

r1[50] # returns an Expr over the low-level Features that encodes r1 = 50
r1.match(r2, offset=5) # returns a list of RoPEPairs that peak when r1 - r2 = 5
r1.chord(10, 30) # return an Expr representing the direction from r1[10] to r1[30], useful for
↪  constructing the manifold
```

4 Compilation

This section describes a simple process to compile TLang algorithms into model weights, producing a `HookedTransformer` from the `transformer-lens` library[24].

The process takes in a user-provided `Algorithm` that contains a list of high- and low-level features, as well as `Neurons` and `Heads` grouped into layers. Then, the compiler follows this process:

1. Lower high-level features into low-level `Features` and pick concrete unit directions to associate to each abstract `Feature` (only for `Features` without specified directions).
2. Convert `Exprs` to vectors.
3. Manipulate and stack these vectors appropriately into weight matrices to populate a `HookedTransformer` model.

4.1 Feature Lowering

Each high-level Feature defines a procedure that lowers to n low-level (one-dimensional) `Features` that are orthogonal to the existing `Feature` directions. For `Features` without a direction specified, an orthonormal set of directions is chosen to avoid interference, while still allowing users to opt-in to interference and `Feature` superposition [25] by choosing their own `Feature` directions.

4.2 `Exprs` → Vectors

In this stage, `Exprs`, such as `feat_one + 0.5 * feat_two`, are converted into vectors. Let an `Expr` have associated `Feature` matrix F where $F_{\cdot,i} = f_i$, where f_i is the direction associated to `Feature` i, and coefficient vector c where c_i is the coefficient for `Feature` i. Then, the vector for an `Expr` is given by a linear combination over F given by c, $v = Fc$. When taking the inner product of residual stream vector z with this vector, it yields the value of evaluating the abstract `Expr` over the scalar `Feature` values,

$$z \cdot v = z \cdot Fc = \sum_{i=1}^{n} c_i f_i \cdot z$$

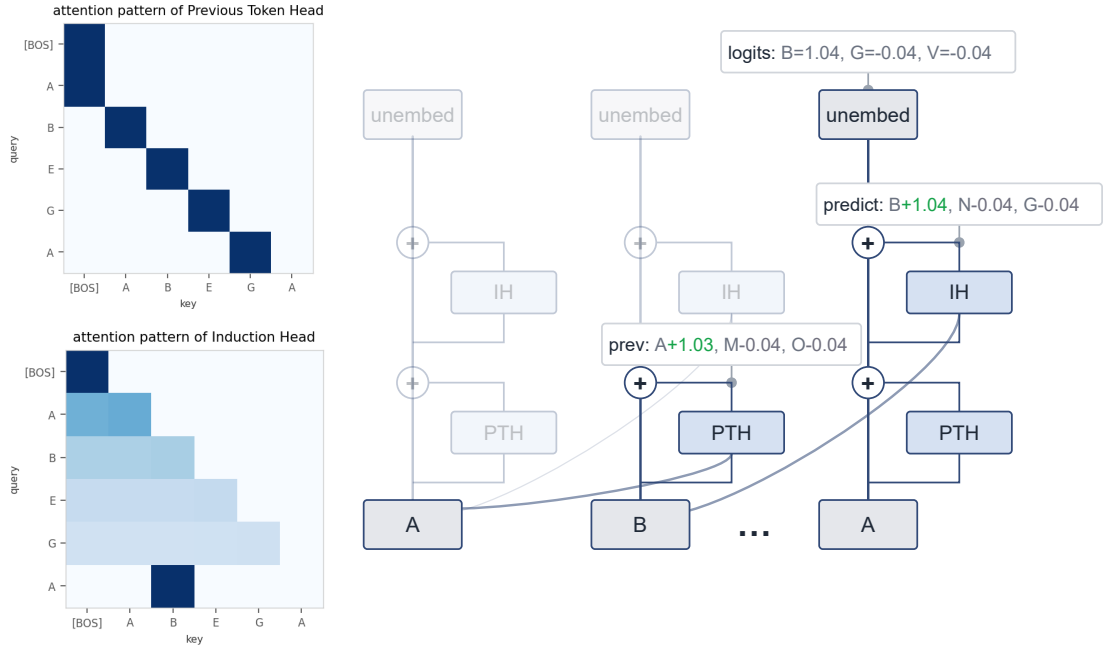

Figure 1 Idealized induction circuit represented in TLang. (Left) shows the attention patterns of the Previous Token Head (PTH) and Induction Head (IH). (Right) visualizes this with a concrete forward pass. *Feature readout* boxes show the values of select features at points in the residual stream or the updates to features from a head.

4.3 `Head` $\rightarrow W^Q, W^K, W^V, W^O$

To construct the W^Q, W^K matrices, we first populate the dimension pair and phase offset for `RoPEPairs` where they are not specified with the highest available dimension pair and phase offset 0. The highest dimension pairs have very low frequencies, so at normal context lengths, the effect of RoPE is negligible and this achieves semantic-only attention contribution. Previous work has found evidence that models employ a similar tactic of putting offset-free information in the most slowly oscillating RoPE dimension pairs [26].

Let the `RoPEPair` of dimension pair i have associated q_i, k_i `Exprs`, query phase ϕ_i, and phase offset ψ_i. Assuming adjacency of RoPE dimension pairs, we populate rows $(2i, 2i+1)$ of W^Q and W^K according to the following equations,

$$
\begin{aligned}
W^Q_{2i} &= \cos(\phi_i) q_i \\
W^Q_{2i+1} &= \sin(\phi_i) q_i \\
W^K_{2i} &= \cos(\phi_i + \psi_i) k_i \\
W^K_{2i+1} &= \sin(\phi_i + \psi_i) k_i
\end{aligned}
$$

A RoPE pair has functional symmetry under any query phase ϕ; TLang uses $\phi = 0$ unless otherwise specified.

Given n OV channels with associated $(v^i_{\text{in}}, v^i_{\text{out}})$ `Exprs`, we create W^V by row-stacking all v^i_{in}, $W^V_{i,\cdot} = v^i_{\text{in}}$ and W^O by column-stacking v^i_{out}, $W^O_{\cdot,i} = v^i_{\text{out}}$,

$$W^O W^V = v^1_{\text{out}}(v^1_{\text{in}})^\top + \cdots + v^n_{\text{out}}(v^n_{\text{in}})^\top = W^{OV}$$

4.4 Set of `Neurons` $\to W^{\text{up}}, W^{\text{down}}$

Let the ith `Neuron` have input `Expr` v^i_{in}, threshold θ, and output `Expr` v^i_{out}. Then $W^{\text{up}}_{i,\cdot} = v^i_{\text{in}}$, ReLU bias $b_i = -\theta$, and $W^{\text{down}}_{\cdot,i} = v^i_{\text{out}}$.

5 Circuits

In this section I walk through the TLang representation of the induction and IOI circuits from the literature [7] [8]. In order to represent these circuits in TLang (such that they can be compiled to weights), we will be forced to consider low-level implementation details that can be ignored when considering circuits only at a high level.

5.1 Induction

The induction circuit is a simple algorithm that has been replicated across models of all sizes. It has been hypothesized to account for a large portion of the in-context learning of models, as well as to explain a significant reduction in loss early in the training of models [7].

The induction circuit, for prefixes of the type `[A] [B] ... [A]`, upweights the probability of `[B]` as the next token. In other words, it upweights the probability of in-context bigrams, thus improving model performance as the context grows. It is implemented with two attention heads, a `Previous Token Head` (PTH) and an `Induction Head` (IH). The PTH attends always to the directly preceding token, and marks that token's identity into a subspace of the current token's residual stream. The IH attends to tokens whose previous token (as indicated in this subspace) matches the query token's identity, and copies the identity of the key token to a subspace that upweights it's probability. See Figure 1 for a visualization of a concrete forward pass of the algorithm (compiled from the TLang representation below).

To represent this circuit in TLang, we can make use of high-level `Categorical` features over the vocabulary. The PTH utilizes the `n_tokens_back` utility to construct `RoPEPairs` to attend sharply to the previous token. Since the `token` and `prev` `Categorical` features are defined over the same `ALPHABET`, we can use the `.all()` method on each to produce a set of channels directly mapping from the `token` feature subspace to the `prev` feature subspace. We can use the same trick to specify the IH's QK circuit, producing pairwise matches with the `.match()` method. Finally, we can copy the `token` identity from the attended token to the `predict` feature subspace.

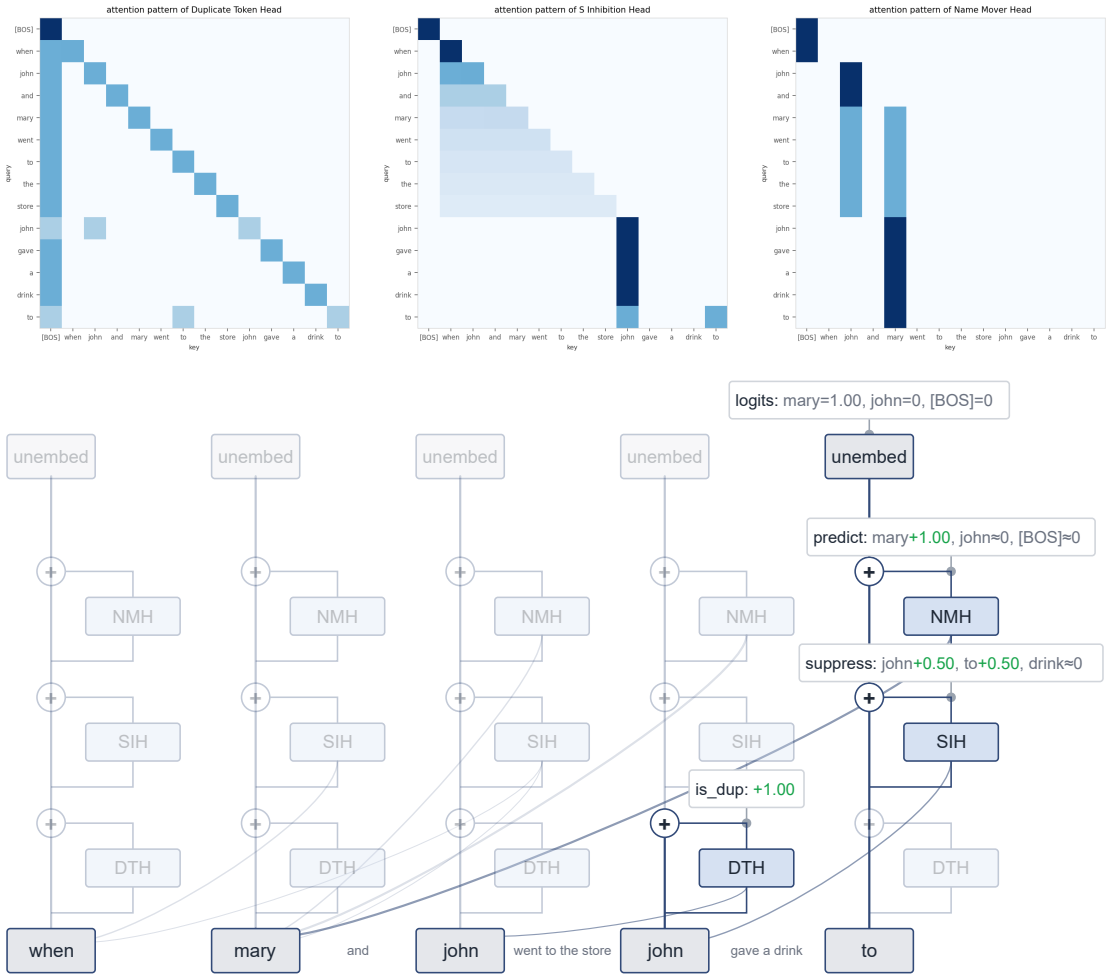

Figure 2 Idealized Indirect Object Identification (IOI) circuit represented in TLang. (Above) shows the attention patterns of the Duplicate Token Head (DTH), Subject Inhibition Head (SIH) and Name Mover Head (NMH). (Below) visualizes this with a concrete forward pass. *Feature readout* boxes show the values of select features at points in the residual stream or the updates to features from a head.

```
token    = Categorical("token", ALPHABET, embed: lambda t: t)
prev     = Categorical("prev", ALPHABET)
predict  = Categorical("predict", ALPHABET)
constant = Feature("constant", embed: 1)

prev_head = Head(
    qk=n_tokens_back(1, 10*constant),
    ov_in=token.all(),
    ov_out=prev.all(),
)

induction_head = Head(
    qk=token.match(prev, q_coef=10, k_coef=10),
    ov_in=token.all(),
    ov_out=predict.all(),
)
```

5.2 Indirect Object Identification

The Indirect Object Identification (IOI) circuit is responsible for completing prompts of the form "When John and Mary went to the store, John gave a drink to" with "Mary"[8]. To do this, it needs to track which names have already been repeated in the sentence, and predict the name which has not been duplicated yet.

The circuit consists of three heads:

- **Duplicate Token Head (DTH).** The DTH attends to tokens that are the same as the current token, and marks the current token as a duplicate in the residual stream. One implementation detail we encounter when implementing this is that the current token is treated similarly to duplicates by the QK circuit. We can handle this by placing an equal amount of attention on the `[BOS]` token, and using a OV channel active on `is_bos` to counteract the `is_duplicate` signal.
- **Subject Inhibition Head (SIH).** The SIH attends to previous tokens that are marked with `is_duplicate` and marks their identity in the `suppress` feature subspace.
- **Name Mover Head (NMH).** The NMH attends to all name tokens (denoted by the `is_name` feature populated in the embed) except for those marked in the `suppress` feature subspace, copying their identity to the `predict` feature subspace.

See Figure 2 for a visualization of the compiled IOI circuit on a concrete prompt.

```
token    = Categorical("token",   VOCAB, n_dims=N_DIMS, embed=lambda t: t)
predict  = Categorical("predict", VOCAB, n_dims=N_DIMS, unembed=lambda t: t)
suppress = Categorical("suppress", VOCAB, n_dims=N_DIMS)
constant     = Feature("constant", embed=1.0)
is_name      = Feature("is_name",  embed=lambda t: 1.0 if t in NAMES else 0.0)
is_duplicate = Feature("is_dup")
is_bos = token[0]

# Attends to tokens that are the same as the current token, with an attention sink on [BOS].
# When no duplicate, attends evenly to self and [BOS], with canceling contributions to
↪  `is_duplicate` feature
duplicate_token_head = Head(
    qk=[RopePair(q=10 * constant, k=10 * is_bos)] # attention sink on [BOS]
       + token.match(token, q_coef=10, k_coef=10), # attend to tokens that are same as self
    ov_in =[constant,         is_bos],
    ov_out=[3 * is_duplicate, -6 * is_duplicate],
)

# Attends to tokens marked as duplicates, copies their identity to `suppress` feature
s_inhibition_head = Head(
    qk=[RopePair(q=10 * constant, k=10 * is_duplicate)],
    ov_in=token.all(),
    ov_out=suppress.all(),
)

# Attends to tokens where `is_name` > 0, ignoring tokens with identity marked in the `suppress`
↪  feature
name_mover_head = Head(
    qk=(
        [RopePair(q=10 * constant, k=10 * is_name)] # attend to names
        + suppress.match(token, q_coef=-10, k_coef=10) # ignoring tokens marked in `suppress`
    ),
    ov_in=token.all(),
    ov_out=predict.all(),
)
```

6 Discussion

Currently, the representation and explanation of reverse-engineered model circuits is ad-hoc. A representation of model computation could make it easier to reverse-engineer circuits more precisely, verify the completeness of proposed explanations, and explain model computation to others. I have argued that a low-level expressive representation alongside high-level geometric primitives has the ability to represent circuits precisely. To fully benefit from such a representation, more abstractions and tools will need to be built to handle the distributed-ensemble nature of model computation, the compression of model circuits and features, as well as the noise that comes from the learning process.

References

[1] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent Abilities of Large Language Models, October 2022.

[2] Thaddäus Wiedemer, Yuxuan Li, Paul Vicol, Shixiang Shane Gu, Nick Matarese, Kevin Swersky, Been Kim, Priyank Jaini, and Robert Geirhos. Video models are zero-shot learners and reasoners. October 2025.

[3] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, August 2021. ISSN 1476-4687. doi: 10.1038/s41586-021-03819-2.

[4] Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom In: An Introduction to Circuits. *Distill*, 5(3):10.23915/distill.00024.001, March 2020. ISSN 2476-0757. doi: 10.23915/distill.00024.001.

[5] 38998623 · Progress measures for grokking via mechanistic interpretability. https://slideslive.com/embed/presentation/38998623?js_embed_version=3&embed_init_token=eyJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE3NzgzNTA4NjcsImV4cCI6MTc3ODQ4MDQ2NywidSI6eyJ1dWlkIjoiN2VjMmUwYTYtNDhlNi00YWEzLTk4ZmItM2NmOTQxMGI0MmJiIiwiaSI6bnVsbCwiZSI6bnVsbCwibSI6ZmFsc2V9LCJkIjoiaWNsci5jYyJ9.IUN1DQm0ZXOsw58nHOJ-eQPQghmrw_PDlTQvabp-qxY&embed_parent_url=https%3A%2F%2Ficlr.cc%2Fvirtual%2F2023%2Foral%2F12572&embed_origin=https%3A%2F%2Ficlr.cc&embed_container_id=presentation-embed-38998623&auto_load=true&auto_play=false&zoom_ratio=&disable_fullscreen=false&locale=en&vertical_enabled=true&vertical_enabled_on_mobile=false&allow_hidden_controls_when_paused=true&fit_to_viewport=true&custom_user_id=&user_uuid=7ec2e0a6-48e6-4aa3-98fb-3cf9410b42bb.

[6] Yaniv Nikankin, Anja Reusch, Aaron Mueller, and Yonatan Belinkov. Arithmetic Without Algorithms: Language Models Solve Math with a Bag of Heuristics. In *The Thirteenth International Conference on Learning Representations*, October 2024.

[7] In-context Learning and Induction Heads. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

[8] Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the Wild: A Circuit for Indirect Object Identification in GPT-2 Small. In *The Eleventh International Conference on Learning Representations*, September 2022.

[9] Authors Jack Lindsey†, Wes Gurnee*, Emmanuel Ameisen*, Brian Chen*, Adam Pearce*, Nicholas L. Turner*, Craig Citro*, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly Templeton, Trenton Bricken, Callum McDougall♢, Hoagy

Cunningham, Thomas Henighan, Adam Jermyn, Andy Jones, Andrew Persic, Zhenyi Qi, T. Ben Thompson, Sam Zimmerman, Kelley Rivoire, Thomas Conerly, Chris Olah, and Joshua Batson*‡ Affiliations Anthropic Published March 27. On the Biology of a Large Language Model. https://transformer-circuits.pub/2025/attribution-graphs/biology.html.

[10] When Models Manipulate Manifolds: The Geometry of a Counting Task. https://transformer-circuits.pub/2025/linebreaks/index.html.

[11] LawrenceC, Adrià Garriga-alonso, Nicholas Goldowsky-Dill, ryan_greenblatt, Tao Lin, jenny, Ansh Radhakrishnan, Buck, and Nate Thomas. Causal scrubbing: Results on induction heads — LessWrong. December 2022.

[12] Andreas Veit, Michael Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 550–558, Red Hook, NY, USA, December 2016. Curran Associates Inc. ISBN 978-1-5108-3881-9.

[13] Gail Weiss, Yoav Goldberg, and Eran Yahav. Thinking Like Transformers. In *Proceedings of the 38th International Conference on Machine Learning*, pages 11080–11090. PMLR, July 2021.

[14] David Lindner, János Kramár, Sebastian Farquhar, Matthew Rahtz, Thomas McGrath, and Vladimir Mikulik. Tracr: Compiled transformers as a laboratory for interpretability. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, pages 37876–37899, Red Hook, NY, USA, December 2023. Curran Associates Inc.

[15] Kiho Park, Yo Joong Choe, and Victor Veitch. The linear representation hypothesis and the geometry of large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *ICML'24*, pages 39643–39666, Vienna, Austria, July 2024. JMLR.org.

[16] Joshua Engels, Eric J. Michaud, Isaac Liao, Wes Gurnee, and Max Tegmark. Not All Language Model Features Are One-Dimensionally Linear. In *The Thirteenth International Conference on Learning Representations*, October 2024.

[17] Kiho Park, Yo Joong Choe, Yibo Jiang, and Victor Veitch. The Geometry of Categorical and Hierarchical Concepts in Large Language Models. In *The Thirteenth International Conference on Learning Representations*, October 2024.

[18] Subhash Kantamneni and Max Tegmark. Language Models Use Trigonometry to Do Addition. In *ICLR 2025 Workshop on Building Trust in Language Models and Applications*, March 2025.

[19] Dhruva Karkada, Daniel J. Korchinski, Andres Nava, Matthieu Wyart, and Yasaman Bahri. Symmetry in language statistics shapes the geometry of model representations, February 2026.

[20] Towards Monosemanticity: Decomposing Language Models With Dictionary Learning. https://transformer-circuits.pub/2023/monosemantic-features.

[21] Circuit Tracing: Revealing Computational Graphs in Language Models. https://transformer-circuits.pub/2025/attribution-graphs/methods.html.

[22] A Mathematical Framework for Transformer Circuits. https://transformer-circuits.pub/2021/framework/index.html#residual-comms.

[23] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with Rotary Position Embedding. *Neurocomput.*, 568(C), February 2024. ISSN 0925-2312. doi: 10.1016/j.neucom.2023.127063.

[24] Neel Nanda and Joseph Bloom. Transformerlens. `https://github.com/TransformerLensOrg/TransformerLens`, 2022.

[25] Toy Models of Superposition. https://transformer-circuits.pub/2022/toy_model/index.html.

[26] Federico Barbero, Alex Vitvitskyi, Christos Perivolaropoulos, Razvan Pascanu, and Petar Veličković. Round and Round We Go! What makes Rotary Positional Encodings useful? In *The Thirteenth International Conference on Learning Representations*, October 2024.