# Tracing Computation Density in LLMs

**Anonymous ACL submission**

## Abstract

Transformer-based large language models (LLMs) are comprised of billions of parameters arranged in deep and wide computational graphs, but it is not clear that they exploit their full capacity for all inputs. We introduce the *s-Trace* method to efficiently estimate the subgraph of size $s$ that best approximates a full model output. With this method, we find the computation in a variety of LLMs to be organized in two distinct phases. A small subgraph mostly composed of early-layer nodes can reconstruct the head of the full model output distribution. Adding further nodes, mostly located in later layers and increasingly consisting of attention heads, leads to incremental refinements in approximating the full output distribution. We find moreover that the amount of necessary computation per input correlates with model uncertainty, and that sparser subgraphs encode shallow statistics, such as unigram frequency. Overall, our results suggest a consistent modular organization in effective LLM computation, with a sparse early-layer core providing a rough prediction that is further refined through denser computations in later layers.

## 1 Introduction

Modern transformer-based large language models (LLMs) can be represented as deep and wide computational graphs whose execution entails billions of operations that collectively update internal representations to yield a prediction. While standard inference executes the full graph for every input, it remains unclear whether the *effective* computation required to generate a faithful prediction utilizes the model's full capacity or relies on a considerably sparser sub-network. Furthermore, the structural nature of this effective computation is largely unknown: is it uniformly distributed across the network, or is it composed of distinct dense and sparse subgraphs? It is also an open question whether this effective amount of computation is dependent on specific input properties.

Answering these questions has both practical and theoretical implications. From a practical point of view, precisely quantifying the true sparsity of LLM computation and predicting which inputs require more computation can help building more efficient models through pruning, routing, distillation or other techniques. Establishing sparsity is also a prerequisite for meaningful application of interpretability techniques such as circuit analysis, that assume that many language/knowledge processes in LLMs can be reduced to the mechanics of a sparse subgraph (Cammarata et al., 2020; Ferrando et al., 2024; Ameisen et al., 2025). From a theoretical point of view, our pursue bears on two fundamental questions in cognitive science. On the one hand, language competence has traditionally been modeled as a sparse, symbolic, rule-based system greatly at odds with LLMs' highly distributed nature (Smolensky et al., 2022). If LLMs turned out to rely on much sparser circuits during average processing, this would pave the way for a reconciliation between the symbolic and distributed views of language (Boleda, 2025). On the other hand, the nature of linguistic complexity and how to measure it are fundamental topics in linguistics (e.g., Jackendoff and Wittenberg, 2014). If the amount of effective computation in LLMs depends on specific properties of their inputs, then LLMs could provide a practical and precise way to measure complexity.

We introduce here a general method to calculate the amount of effective computation performed by an LLM in response to an input. The method, based on automated circuit discovery (Conmy et al., 2023; Ferrando and Voita, 2024; Syed et al., 2024), extracts the sub-network of size $s$ (the *s-Trace*) that, given an input, best reconstructs the output probability distribution generated by the full model. Varying the computation budget, measured as the size of the sub-network, allows for a systematic

examination of the performance/complexity trade-off in LLMs (Fig. 2).

We analyze 10 different LLMs, ranging from 7B to 14B parameters, and find pervasive sparsity in LLM computation. By analyzing the trace across varying granularities, we identify two distinct phases. First, for highly sparse subgraphs (beginning at $10^{-4}$ of full model size), LLMs undergo a *construction phase*, in which each operation strongly affects the output prediction, and there is a predominance of early layers being added. Within this phase (at $\approx 10^{-3}$), a highly stable *minimal core* emerges, sufficient to accurately recover the head of the output distribution. Beyond $10^{-2}$, having already reconstructed $40\%$ to $60\%$ of the prediction probability mass, the trace enters a *refinement phase*. Here, the marginal utility of additional operations, which are mostly located in later layers, becomes diffuse. At this stage, the network only fine-tunes the prediction, integrating the nuances required to encode the full richness of the final output distribution.

We also find that different LLMs distribute computation in strikingly similar patterns across similar inputs, suggesting that resource allocation depends on intrinsic input properties, more than on the architectural details of specific models. Two results suggest that computation density, as measured by the *s-Trace* method, is indeed related to the complexity of the input: 1) the density distribution is positively correlated with LLM output entropy (LLMs' own uncertainty estimate); 2) density increases for rarer words (plausibly harder to guess).[1]

## 2 Background

**Sparsity in LLMs** Our work builds on a rich literature establishing structural sparsity in neural networks (LeCun et al., 1989; Frankle and Carbin, 2019), which extends directly to LLMs (Voita et al., 2023). While static pruning techniques identify permanently redundant parameters or layers across a model (Michel et al., 2019; Lad et al., 2024; Li et al., 2024), a growing body of evidence suggests that computation density is highly dynamic and input-dependent. Modern efficiency frameworks exploit this by dynamically bypassing modular units, attention heads, or entire layers on the fly based on token-level complexity and input difficulty (Zhang et al., 2022; Liu et al., 2023; Fan et al., 2024; Lee et al., 2025). While these methods focus on task-specific efficiency, our goal is analytical: we seek to quantify the computational footprint required by an LLM to faithfully reconstruct its own output distribution during generic next-token prediction.

**Mechanistic Interpretability** To quantify the amount of computation in LLMs, we leverage methods from mechanistic interpretability. This line of research generally views LLMs as computational graphs composed of discrete, human-interpretable "circuits" (Olah et al., 2018; Cammarata et al., 2020; Ameisen et al., 2025). Notable examples include circuits responsible for the greater-than operation (Hanna et al., 2023), indirect object identification (Wang et al., 2022), subject-verb agreement (Ferrando et al., 2024), and knowledge retrieval (Meng et al., 2022). Circuit discovery can be automated, for instance using greedy search (Conmy et al., 2023), sparse dictionary learning (Bricken et al., 2023), activation patching (Syed et al., 2024), or gradient-based pruning (Bhaskar et al., 2024). More recently, the granularity of circuit discovery has transitioned from coarse attention heads to individual features using sparse autoencoders (Bricken et al., 2023) and cross-layer transcoders (Ameisen et al., 2025). We specifically build upon the Information Flow Route (IFR) framework (Ferrando and Voita, 2024), which leverages attribution to efficiently identify circuits at scale. However, rather than applying it to the identification of task-specific circuits, we repurpose it as a general method to estimate the maximum degree of output reconstruction that can be achieved by a subgraph of a certain size.

**Structure of Computation in LLMs** Our *s-Trace* approach is motivated by the "unraveled view" of residual networks (Veit et al., 2016), which states that the transformer architecture behaves not as a monolithic pipeline, but as an implicit ensemble of shallower subnetworks. Our method exploits this by using an unconstrained graph traversal algorithm to identify and extract these non-sequential, unraveled paths without forcing a rigid layer-by-layer order. This aligns with the iterative inference hypothesis (Jastrzebski et al., 2018; Belrose et al., 2023), which posits that each residual update in the transformer incrementally modifies a token's hidden state to progressively shape the next-token distribution (Geva et al., 2022). In contrast, Lad et al. (2026) identified

[1]The code and data required to reproduce the experiments will be made publicly available on a Github repository upon acceptance.

hierarchical "stages of inference" where processing goes through four sequential, depth-dependent phases: detokenization, feature engineering, prediction ensembling, and residual calibration. Our study helps validate and reconcile both of these views by identifying distinct phases in the trace.

## 3 Methodology

### 3.1 LLM as a Causal Graph

Our work is grounded in the formal representation of a language model's internal processes as a computational graph (Conmy et al., 2023; Ferrando and Voita, 2024; Syed et al., 2024). We consider an LLM, that accepts $n$ tokens as inputs, $t = (t_1, \ldots, t_n)$, and computes a probability distribution over the next token, $P(t_{n+1} \mid t)$. This process is modeled as a forward pass through a directed acyclic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$; nodes $\mathcal{V}$ represent intermediate states of the tokens and edges $\mathcal{E}$ represent operations affecting the information flow.

We focus on the standard transformer architecture (Vaswani et al., 2017). The forward pass consists of a sequence of residual blocks (or layers). First, discrete tokens are embedded into continuous vectors, $h^0 = (h_1^0, \ldots, h_n^0)$, forming the initial nodes of the graph. These representations are iteratively transformed through $L$ layers. The state at layer $l$, denoted $h^l = (h_1^l, \ldots, h_n^l)$, is a function of the previous layer's activations.

Each layer $l$ is itself composed of two sub-layers: a Multi-Head Attention (MHA) and a Multi-Layer Perceptron (MLP). For a specific token $i$ at layer $l$, the MHA update rule to produce the sub-layer output $z_i^l$ is:

$$z_i^l = \underbrace{h_i^l}_{\text{Residual Edge}} + \sum_{k,j} \underbrace{\phi^{l,k}(h_i^l, h_j^l)}_{\text{Attention Edges}} \quad (1)$$

where $k \in [1, N_H]$ with $N_H$ the number of attention heads, $j \in [1, n]$, and $\phi^{l,k}(h_i^l, h_j^l)$ representing an *attention head contribution*: the vector "moved" from source token $j$ to target token $i$ by head $k$ (Ferrando and Voita, 2024). This pairwise decomposition is central to our graph formulation. Similarly, the MLP update rule is:

$$h_i^{l+1} = \underbrace{z_i^l}_{\text{Residual Edge}} + \underbrace{\text{mlp}_l(z_i^l)}_{\text{MLP Edge}} \quad (2)$$

By unrolling Equations 1 and 2, we observe that the vector representation at any node is simply the sum of vectors from its incoming edges. This yields a granular view of information flow. An MHA sub-layer is decomposed into $N_H \times n + 1$ edges per target token,[2] and an MLP sub-layer gives rise to 2 edges per target token.

| Component | Formal Definition of $\mathcal{G}$ |
|---|---|
| Nodes $\mathcal{V}$ | $\{h_{i+1}^l, z_{i+1}^l \mid l \in [0, L], i \in [1, n]\}$ |
| Edges $\mathcal{E}$ | $\begin{cases} \text{Attn:} & \phi^{l,k}(h_i^l, h_j^l) \quad \text{(Eq. 1)} \\ \text{MLP:} & \text{mlp}_l(z_i^l) \quad \text{(Eq. 2)} \\ \text{Resid:} & h_i^l, z_i^l \end{cases}$ |

Table 1: Components of the computational graph $\mathcal{G}$.

Formally, the computational graph $\mathcal{G}$ is defined as in Table 1. Note that both nodes *and* edges are defined token-wise. This allows for finer modeling of information routing compared to approaches that aggregate head outputs into a single edge per target token (e.g., Syed et al., 2024). As an illustration, given an input of length 100, a model with 32 attention heads and 32 layers corresponds to a graph with $6,302$ nodes and $5,022,103$ edges.

### 3.2 s-Trace Definition

**Definition 1** ($s$-Trace)**.** *Given a budget $s \in \mathbb{N}^+$, the s-Trace $\mathcal{T}_s = (\mathcal{V}_s, \mathcal{E}_s)$ is the optimal subgraph of $\mathcal{G}$ such that $\mathcal{V}_s \subseteq \mathcal{V}$, $\mathcal{E}_s \subseteq \mathcal{E}$, $|\mathcal{E}_s| = s$, and:*

$$\mathcal{E}_s = \underset{\mathcal{E}_s \subset \mathcal{E},\, |\mathcal{E}_s| = s}{\arg\min} \; \mathcal{D}\left(P(t_{n+1} \mid t), \hat{P}_{\mathcal{T}_s}(t_{n+1} \mid t)\right)$$

*where $\mathcal{D}$ is a distance metric and $\hat{P}_{\mathcal{T}_s}$ is the prediction restricted to subgraph $\mathcal{T}_s$.*

In words, the trace $\mathcal{T}_s$ is the subgraph of size $s$ whose predictions are closest, among all possible subgraphs of the same size, to those of the full model. During inference, the s-Trace's prediction $\hat{P}_{\mathcal{T}_s}$ is obtained by applying a binary mask, $M = \{m_e\}_{e \in E}$, to the computational graph, where $m_e = 1$ if edge $e \in E_T$ and 0 otherwise. For instance, the masked update rule for the MHA sub-layer (Eq. 1) becomes:

$$z_i^l = m_{i,\text{res}}^l \cdot h_i^l + \sum_{k,j} m_{i,j}^{l,k} \cdot \phi^{l,k}(h_i^l, h_j^l) \quad (3)$$

where $m \in \{0, 1\}$ are the binary decisions for the residual and attention edges respectively. In the rest of the paper, we abuse the size symbol $s$ to refer to the relative number of edges $\bar{s} = |\mathcal{E}_s|/|\mathcal{E}| \in [0, 1]$. This quantity more intuitively denotes the size of the chosen subgraph relative to the full model.

[2] In causal masking, some edges are set to 0.

### 3.3 Trace Extraction via Graph Traversal

The challenge of trace extraction is to identify the subset of edges, $\mathcal{E}_s \subset \mathcal{E}$ with $|\mathcal{E}_s| = s$, that best reconstruct the model's output distribution (cf. Def. 1). Because of the large scale of the models analyzed (7B to 14B parameters) and the dense and intricate nature of the computational graph, identifying the *optimal* trace is intractable. Given that computationally expensive methods like greedy search (Conmy et al., 2023) or activation patching (Syed et al., 2024) are infeasible for identifying all edges, we approximate the trace with a training-free, activation-based strategy that just requires a single pass.

**Importance Score** We compute the importance score associated to each edge $e \in \mathcal{E}$ using the L1-norm, positing that the impact of an operation is proportional to the magnitude of its update vector. To ensure fair comparison across the graph, scores are normalized node-wise. For any edge $e$ carrying a vector $\mathbf{v}_e$ (e.g., an attention head output $\phi^{l,k}$) into a target node $v$, the score is computed relative to all other edges feeding into $v$, such that $\mathcal{I}(e) = \|\mathbf{v}_e\|_1 / \sum_{e' \to v} \|\mathbf{v}_{e'}\|_1$. We experimented with alternatives, such as L2-norm or similarity between the node and the edges, and found the L1-norm to be either comparable or superior (yielding lower reconstruction error at a comparable size).

**Graph Traversal** Trace extraction is based on *Greedy Best-First Search*, a graph traversal method that iteratively increases the trace size by adding the most important edge available, starting from the last-token node in the last layer, until the budget $s$ is reached. Every time an edge $e$ is added, the nodes visited by $e$ are added, too, and the edges connected to these nodes become available for selection. App. A provides the full definition.

### 3.4 Evaluating Trace Faithfulness

To evaluate the quality of a trace, we measure its *reconstruction error*: how well the masked model output $\hat{P}_{\mathcal{T}_s}(t_{n+1} \mid t)$ approximates the full model output $P(t_{n+1} \mid t)$. Unlike prior work, that often focuses on binary task success (e.g., whether the correct token is the top-1 prediction), we adopt a more conservative and general approach by measuring the preservation of the full output distribution. We quantify this using the *Total Variation* (TV) distance:

$$\delta_{TV}(s) = \frac{1}{2} \sum_{v \in \mathbb{V}} |P(v) - \hat{P}_{\mathcal{T}_s}(v)| \qquad (4)$$

where $\mathbb{V}$ is the token vocabulary, and $\hat{P}_{\mathcal{T}_s}(v)$ and $P(v)$ are probabilities assigned to token $v$. A low $\delta_{TV}$ indicates that the trace is *sufficient* to recover the model's behavior.[3] We select TV over alternatives such as the Kullback-Leibler divergence because the latter is unbounded and sensitive to relative errors, meaning that it can be dominated by discrepancies in the low-probability tail. In contrast, TV measures the absolute difference in probability mass. This ensures that our metric focuses on the dominant modes of the distribution—where the model's actual predictions lie—rather than being disproportionately affected by the noisy tail.

### 3.5 Experimental Details

**Dataset** We use a dataset of $5,000$ sequences from Wikitext (Merity et al., 2017). They were selected to contain 40 words on average (one or two sentences), making sure that they start at a sentence beginning and end with either a full word or punctuation (no word fragment). Following trace extraction, we discard any instances that encountered runtime anomalies or numerical instability (such as out-of-memory errors or NaN values). We get minimally 4,250 successfully processed sequences per model (Table 2 in App. C.3).

**Model Zoo** We use 10 different LLMs from 7 HuggingFace families (Table 2 in App. C.3), with sizes ranging in $[7, 14]$ billion parameters.

**Trace Collection** For each input, we extract the s-Trace at 26 levels of granularity, from very small to very large, through a fixed size grid (App. C.2). We collect about 1.3M traces in total.

## 4 Results

### 4.1 Stages of Reconstruction in the Trace

We first measure the extent to which the trace size influences its reconstruction error. If LLM computation is sparse, then a small trace should be able to faithfully reconstruct the full model distribution. A dense computation would result in a larger trace. Our analysis reveals that the reconstruction process

[3] To assess necessity–whether the edges in $\mathcal{T}_s$ are the *only ones* capable of producing the output–we employ a complementary evaluation in App. D.2, where we ablate the trace edges and observe the corresponding degradation.

| Phase | Description | Scale |
|---|---|---|
| **Construction Phase** | Steep decay in reconstruction error, marking the initial assemblage of the "core" architecture. Driven by a sharp increase of early-layer paths. | $s \in [10^{-4}, 10^{-2}]$ |
| **Minimal Core** | A highly stable, architecture-invariant early-layer graph backbone that is causally sufficient for recovering the salient modes of the prediction. | $s \approx 10^{-3}$ |
| **Refinement Phase** | Progressively incorporates the nuanced information required for recovering the full prediction. Emergence of distributed, late-layer attention paths. | $s > 10^{-2}$ |

Figure 1: Important stages in trace size profiles.

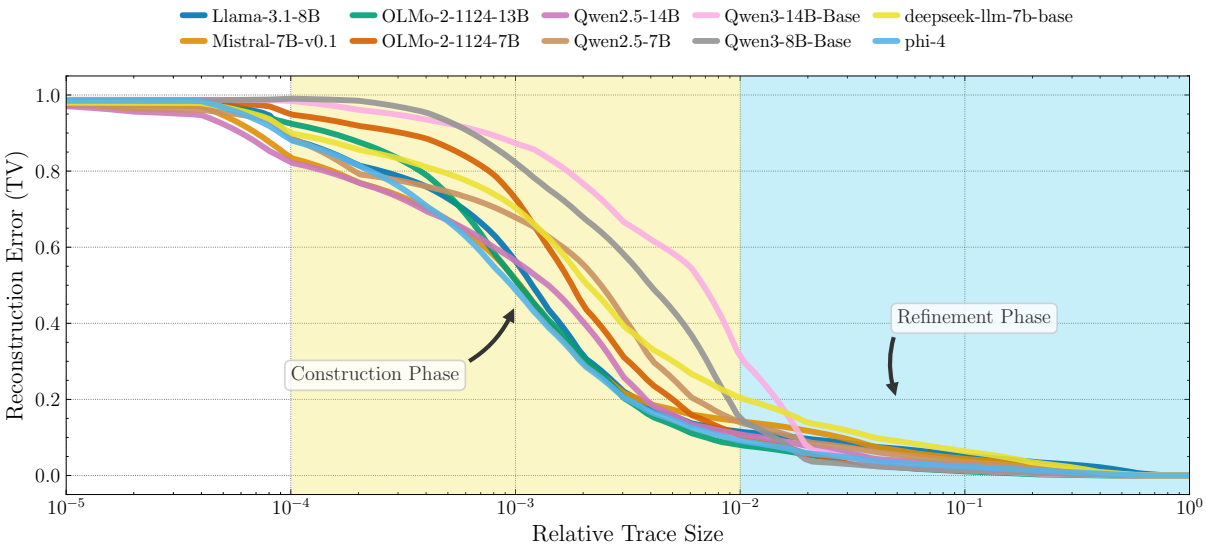



Figure 2: Relation between reconstruction error (measured with Total Variation distance) and trace size for various LLMs. The trace undergoes two distinct phases: a *construction phase*, and a *refinement phase*.

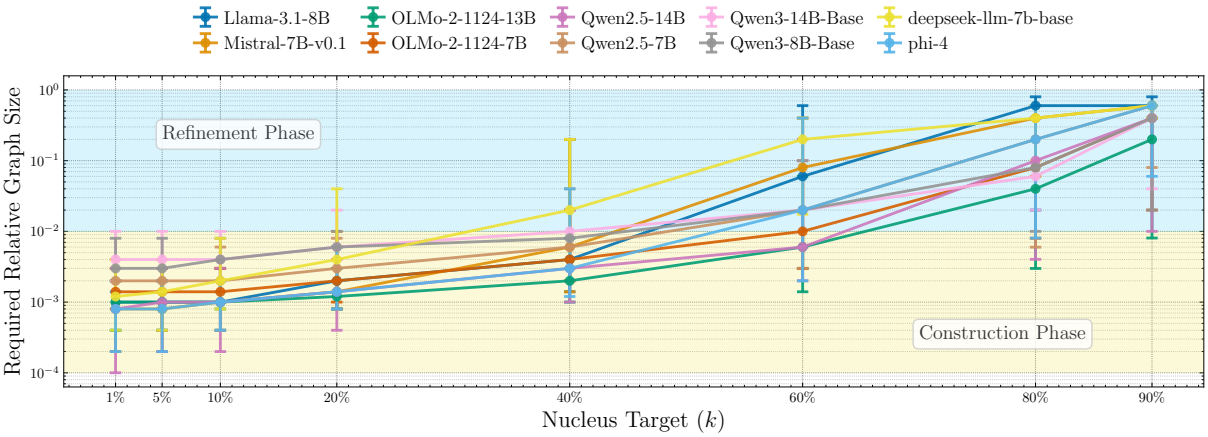



Figure 3: The minimal size $s$ required to reconstruct the top-$k\%$ (*nucleus target*) of the full model's probability mass. A very small trace ($s \approx 10^{-3}$, the *minimal core*) suffices to reconstruct the top 1% prediction, but only at the end of the *construction phase* ($s = 10^{-2}$) do we recover 40% to 60% of the probability mass.

follows three distinct steps: a *construction phase*, during which a *minimal core* is established, and a progressive *refinement phase* (Table 1).

**Construction Phase** Fig. 2 shows that very small traces ($s < 10^{-4}$) exhibit high reconstruction error. There is then a distinct *construction phase* when $s \in [10^{-4}, 10^{-2}]$ (corresponding to 0.01%–1% of the full graph). This interval is characterized by a sharp, non-linear decay in reconstruction error where the marginal utility of adding edges is high. This suggests that, while the full model is dense, its functional output is governed by a sparse subgraph that is present at this threshold.

**Minimal Core** To better understand the effect of the construction phase on model prediction, we define a *nucleus reconstruction* metric, measuring the minimal size $s$ required to reconstruct the top-$k$% of the full model's probability mass, for $k \in \{1, 5, 10, \ldots, 90\}$. This metric, inspired by nucleus sampling (Holtzman et al., 2020), identifies the computational budget necessary to recover varying amounts of the original distribution. Fig. 3 shows that a remarkably small subgraph ($s \approx 10^{-3}$) forms a *minimal core* sufficient to recover the top-1% of the distribution, which includes the most likely token. This subgraph size corresponds to the middle value in the construction phase observed in Fig. 2. While it suffices for greedy decoding, it still lacks the nuance needed for high-quality generation. The latter is reached at the end of the construction phase, where the trace is able to reconstruct the top 40% to 60% of the prediction probability mass.[4]

**Refinement Phase** Beyond $s > 10^{-2}$, we observe a ceiling effect where the already low reconstruction error slowly decreases (Fig. 2). The model enters a *refinement phase* in which achieving higher fidelity (e.g., $k > 60\%$) requires an order-of-magnitude increase in trace size (Fig. 3). In this stage, the marginal utility of adding edges is small, as the trace progressively incorporates the nuanced information required to reconstruct the full distribution. Fig. 3 shows that, once the top-10% of the probability mass is reconstructed, $\log(s)$ scales monotonically with $k$. This log-linear behavior suggests that, in this regime, the remaining distribution is constructed via the additive accumulation of relatively independent computational paths, rather than a single, densely interconnected component.

[4]The typical value for good-quality generation with nucleus sampling is $k = 60\%$ (Holtzman et al., 2020).

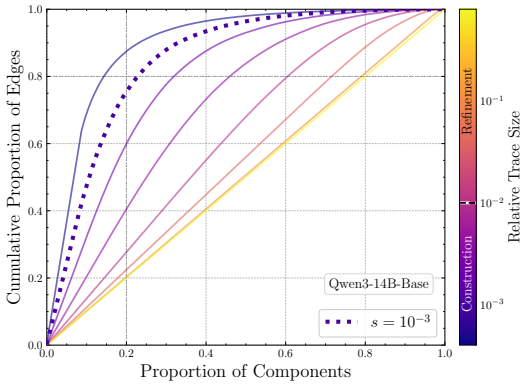



Figure 4: Cumulative edge allocation across components ranked by frequency in the dataset for `Qwen3-14B-Base` (more LLMs in App. D.6). At the minimal core ($s = 10^{-3}$, dashed line), edges are concentrated within a small subset of frequent components.

## 4.2 Structure of the Trace

**Stability of the Minimal Core** We ask whether the minimal core is shared across inputs, or whether it is input-dependent. For each trace size, we calculate the global frequency of each trace component (e.g., residual at layer $l$, MLP at layer $l$, attention head $k$ at layer $l$) across all instances, rank them, and plot the cumulative edge allocation $y$ accounted for by the top-$x$ fraction of the ranked components (Fig. 4). Around the minimal core, at $s = 10^{-3}$, the distribution is highly concentrated: For instance, in `Qwen1-14B-Base`, the top 20% most frequent components account for 80% of all edges across all dataset instances. This shows the minimal core is remarkably stable across contexts. As the trace size increases, the distribution predictably flattens.

**Layer Makeup at Different Stages** Fig. 5a illustrates the layer-wise distribution of the transformations that comprise the trace. At a very small size ($s < 10^{-4}$), the trace is dominated almost exclusively by the final layers of the model. However, coinciding with the onset of the construction phase ($s = 10^{-4}$), the proportion of early layers increases sharply, peaking at $s = 10^{-3}$. At this stage, initial layers and early-middle layers together constitute 80% of the edges. This peak closely aligns with the budget required to reconstruct the top-1% of the probability mass. Beyond this threshold, the proportion of late and late-middle layers steadily increases. Thus, (1) the minimal core relies predominantly on early-layer representations, and (2) the subsequent refinement phase ($s > 10^{-2}$) is driven by later layers, that might nuance and contextualize

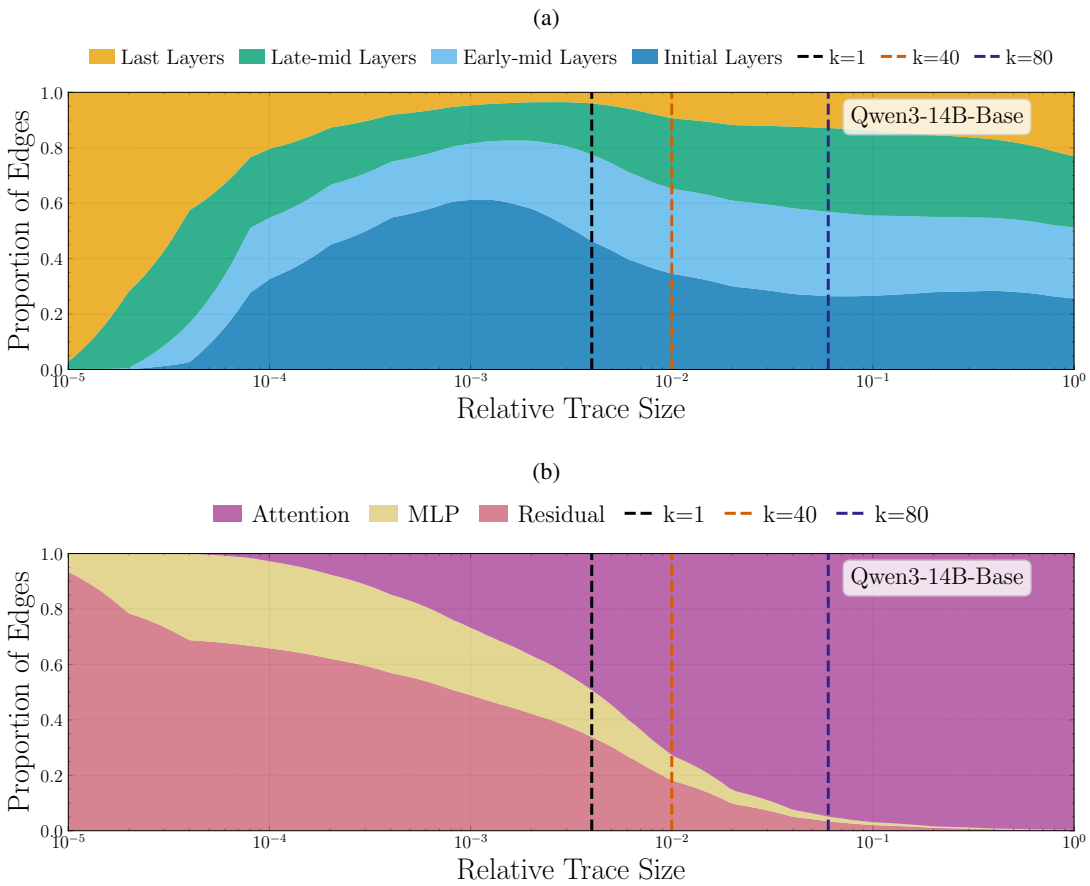



Figure 5: Trace makeup at different sizes for `Qwen3-14B-Base` (more in App. D.5). **(a)** Composition by layer depth. **(b)** Composition by transformer components. The vertical lines mark the median size for nucleus $k$ reconstruction.

the predictions established earlier.

**Extreme Sparsity in Attention** Fig. 5b displays the trace composition categorized into attention, MLP, and residual connections. By design (Sec. 3), the computational graph is heavily skewed; attention heads account for over 99% of all possible edges (see the far right of the figure, where the distribution for the full model is shown). Strikingly, however, the minimal core has a remarkably balanced proportion of trace components, with approximately 30% residual edges, 20% MLP edges, and only 50% attention edges. This disproportion implies that attention operates under extreme sparsity, as only a minimal fraction of total attention edges is required. The subsequent refinement phase ($s > 10^{-2}$) is however characterized by an increased contribution of attention, suggesting that the latter plays a key role in refinement.

### 4.3 Input-related Effects on Sparsity

We next investigate which contextual factors modulate computation density. We define a quantity $\mathcal{C}$, obtained by measuring, for each input separately, the area under the curve (AUC) of the error-vs-trace-size plot (App. C.1). The larger this area, the more resources an LLM is effectively recruiting to produce its output given the current input.

**Consistency Across LLMs** The observed computation density patterns across inputs are remarkably stable across LLMs, with positive correlations ranging from 0.26 to 0.71 ($\mu = 0.52$, full results in App. D.4). This strong cross-model alignment indicates that computation density is largely independent of architecture or parameter scale.

**More Uncertainty, More Computation** We quantify prediction uncertainty using the Shannon entropy of the output probability distribution of the full model and correlate it with the amount of computation $\mathcal{C}$. Across all models, this correlation is significantly positive, albeit moderate, spanning $[0.04, 0.37]$ ($\mu = 0.22$; full results in App. D.3). For low-entropy contexts, minimal traces achieve near-perfect reconstruction, indicating high sparsity. Conversely, high-entropy contexts require larger trace budgets to achieve equivalent error, characterized by an extended refinement phase.

**Sparser Traces Encode Shallow Statistics** We further investigate how the (Wikitext-derived) frequency of the highest-probability output token relates to computation density. We observe a consistently negative correlation between token frequency and $\mathcal{C}$ across all evaluated model families, with correlations spanning $[-0.45, -0.22]$ (full results in App. D.3). This indicates that more frequent tokens tend to be associated with less computation, and that the relationship is robust across different models. This suggests that sparse traces are sufficient to encode superficial statistics, such as unigram frequency. Capturing more complex linguistic dependencies, such as those required to predict rare tokens, necessitates larger traces.

## 5 Discussion

**Key Findings** We find two distinct regimes in how computation is allocated across recent LLMs. First, spanning from 0.01% to 1% of the full model size, the trace exhibits a *construction phase* in which each added operation exerts a major impact on the reconstruction of the prediction. Second, from 1% to 100% of the model size, the trace enters a *refinement phase*, where the prediction is slowly fine-tuned, incorporating the subtle nuances required to encode the full richness of the model's output distribution. The construction phase is marked by the emergence of the *minimal core*, which we estimate to be contained within only 0.1% of the full model. This remarkably compact set of operations appears sufficient for predicting the highest-probability token of the full distribution, while remaining stable across diverse inputs.

Although our *s-Trace* method is depth-agnostic (due to its "unraveled view"; Veit et al., 2016), our findings align with a staged view of transformer depth (Lad et al., 2024). The minimal core is predominantly composed of early-layer operations where detokenization and feature engineering might occur. Conversely, the refinement phase relies increasingly on later layers for prediction ensembling and residual calibration to fine-tune the final output distribution. Csordás et al. (2026) observed that skipping the second half of layers minimally affects representations for *future* token predictions (beyond the next token), indicating that late operations are largely decoupled and focus primarily on polishing the immediate distribution.

**Implications for Interpretability** Our findings offer several takeaways for NLP interpretability:

*(1) What is the appropriate scale for circuits?* There is a threshold below which circuits fail to faithfully represent the model's internal processing. As a rule of thumb, we suggest considering the minimal core (0.1%) as the smallest scale for basic faithfulness. While more sophisticated circuit discovery methods might push lower, the minimal core ensures MLP and attention edges are well-balanced, making it easier to isolate their respective roles (Fig. 5b).

*(2) Output distributions matter.* We observe that output distribution entropy closely relates to the model's computational volume. Thus, evaluating subgraphs using simplified binary task metrics that ignore distribution properties can yield degenerate circuits.

*(3) Not all predictions are created equal.* Just like low- and high-entropy predictions elicit different computational profiles, token frequency shapes effective graph size. This suggests that baseline statistical properties of token prediction must be considered when deriving and analyzing circuits.

*(4) Mechanistic interpretability need not restrict itself to hyper-sparse circuits.* Integrating computational graph formalisms with graph theory tools offers a promising framework for analyzing dense, distributed processing. Our computation density metric represents an initial step. We hope this work encourages the community to explore other network properties, such as centrality or modularity, to better characterize the complex internal dynamics of LLMs beyond simple circuit identification.

**Implications for Cognitive Science** The robust link between computation density and input properties, and the fact that density varies in similar ways across models open exciting avenues for research. Computational density could potentially serve as a novel metric for *linguistic complexity*. Moreover, our results suggest that LLMs recruit fewer resources for less demanding inputs (such as frequent words), pointing to the idea that LLMs implement a form of *efficient processing*, in line with the emerging consensus on how humans process language (Gibson et al., 2019). Last, analyzing which linguistic features modulate computation density can shed light on whether LLMs naturally find a trade-off between more distributed and more symbolic processing, akin to the one conjectured for human language (Boleda, 2025).

## Limitations

*(a)* Our findings directly depend on how we identify the trace. Better methods could lead to smaller reconstruction error for equivalent budgets, or highlight emerging patterns we could not identify with the current approach. As a sanity check, we experimentally compared our approach with a strong baseline consisting in a random graph traversal that favors residual and MLP edges, and found that our method provides better results (App. D.1).

*(b)* While we evaluate reconstruction error via zero-ablation of the edges, recent work notes that zero-ablation can push activations off the data manifold, a limitation often mitigated by mean-ablation or activation patching (Li and Janson, 2024). As a sanity check, we evaluated zeroing attention edges *before* the softmax (a softer ablation since the softmax operation rebalances weights across remaining edges) and observed results consistent with our primary approach. We ultimately retain the after-softmax ablation for its superior interpretability. We leave more nuanced interventions, such as mean-edge replacement, to future work due to the technical challenges they introduce in our fine-grained setup.

*(c)* Our study is limited to English data and the transformer architecture. We plan in the future to apply it to other languages and architectures to broaden the generality of our results.

## Impact Statement

This work advances the field of Mechanistic Interpretability, which aims to elucidate the internal decision-making processes of Large Language Models (LLMs). By characterizing how computation is allocated in LLMs, our research contributes to the development of more transparent and accountable AI systems. Understanding the true computation and information flow is a prerequisite for reliable model monitoring, debugging, and safety auditing. We do not foresee any direct negative societal impacts from this work. While insights into model efficiency could theoretically be used to optimize harmful systems, the primary application of our findings is to prevent oversimplified explanations of model behavior, thereby fostering more robust safety evaluations.

## AI Assistant

We acknowledge the use of an AI assistant to aid in drafting sections of this manuscript and generating code for the experiments. All AI-generated text and code were thoroughly reviewed, and the final edits and revisions were exclusively performed by the authors.

## References


Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J Hewett, Mojan Javaheripi, Piero Kauffmann, and 1 others. 2024. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*.

Emmanuel Ameisen, Jack Lindsey, Adam Pearce, Wes Gurnee, Nicholas L. Turner, Brian Chen, 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, and 8 others. 2025. Circuit tracing: Revealing computational graphs in language models. *Transformer Circuits Thread*.

Nora Belrose, Igor Ostrovsky, Lev McKinney, Zach Furman, Logan Smith, Danny Halawi, Stella Biderman, and Jacob Steinhardt. 2023. Eliciting latent predictions from transformers with the tuned lens. *arXiv preprint arXiv:2303.08112*.

Adithya Bhaskar, Alexander Wettig, Dan Friedman, and Danqi Chen. 2024. Finding transformer circuits with edge pruning. *Advances in Neural Information Processing Systems*, 37:18506–18534.

Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, and 1 others. 2024. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*.

Gemma Boleda. 2025. LLMs as a synthesis between symbolic and distributed approaches to language. In *Findings of the ACL: EMNLP 2025*, pages 9365–9379.

Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Zac Hatfield-Dodds, Alex Tamkin, Karina Nguyen, and 6 others. 2023. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*. Https://transformer-circuits.pub/2023/monosemantic-features/index.html.

Nick Cammarata, Shan Carter, Gabriel Goh, Chris Olah, Michael Petrov, Ludwig Schubert, Chelsea Voss, Ben Egan, and Swee Kiat Lim. 2020. Thread: Circuits. *Distill*. Https://distill.pub/2020/circuits.

Arthur Conmy, Augustine Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso.

2023. Towards automated circuit discovery for mechanistic interpretability. In *Proceedings of NeurIPS*, volume 36, pages 16318–16352, New Orleans, LA.

Róbert Csordás, Christopher D Manning, and Chris Potts. 2026. Do language models use their depth efficiently? *Advances in Neural Information Processing Systems*, 38:160313–160362.

Siqi Fan, Xin Jiang, Xiang Li, Xuying Meng, Peng Han, Shuo Shang, Aixin Sun, Yequan Wang, and Zhongyuan Wang. 2024. Not all layers of llms are necessary during inference. *arXiv preprint arXiv:2403.02181*.

Javier Ferrando, Gabriele Sarti, Arianna Bisazza, and Marta Costa-jussá. 2024. A primer on the inner workings of transformer-based language models. https://arxiv.org/abs/2405.00208.

Javier Ferrando and Elena Voita. 2024. Information flow routes: Automatically interpreting language models at scale. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*.

Jonathan Frankle and Michael Carbin. 2019. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Proceedings of ICLR*, New Orleans, LA. Published online: https://openreview.net/group?id=ICLR.cc/2019/conference.

Mor Geva, Avi Caciularu, Kevin Wang, and Yoav Goldberg. 2022. Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space. In *Proceedings of the 2022 conference on empirical methods in natural language processing*, pages 30–45.

Edward Gibson, Richard Futrell Steven Piantadosi, Isabelle Dautriche, Kyle Mahowald, Leon Bergen, and Roger Levy. 2019. How efficiency shapes human language. *Trends in Cognitive Science*, 23(5):389–407.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Michael Hanna, Ollie Liu, and Alexandre Variengien. 2023. How does gpt-2 compute greater-than?: Interpreting mathematical abilities in a pre-trained language model. *Advances in Neural Information Processing Systems*, 36:76033–76060.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *International Conference on Learning Representations*.

Ray Jackendoff and Eva Wittenberg. 2014. What you can say without syntax: A hierarchy of grammatical complexity. In Frederick Newmeyer and Laurel Preston, editors, *Measuring Grammatical Complexity*, pages 65–82. Oxford University Press, Oxford, UK.

Stanisław Jastrzebski, Devansh Arpit, Nicolas Ballas, Vikas Verma, Tong Che, and Yoshua Bengio. 2018. Residual connections encourage iterative inference. In *International Conference on Learning Representations*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego Casas, and 1 others. 2024. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Vedang Lad, Jin Hwa Lee, Wes Gurnee, and Max Tegmark. 2024. The remarkable robustness of llms: Stages of inference? *arXiv preprint arXiv:2406.19384*.

Vedang Lad, Jin Hwa Lee, Wes Gurnee, and Max Tegmark. 2026. Remarkable robustness of llms: Stages of inference? *Advances in Neural Information Processing Systems*, 38:130050–130083.

Yann LeCun, John Denker, and Sara Solla. 1989. Optimal brain damage. *Advances in neural information processing systems*, 2.

Ayeong Lee, Ethan Che, and Tianyi Peng. 2025. How well do llms compress their own chain-of-thought? a token complexity approach. *arXiv preprint arXiv:2503.01141*.

Maximilian Li and Lucas Janson. 2024. Optimal ablation for interpretability. *Advances in Neural Information Processing Systems*, 37:109233–109282.

Wei Li, Lujun Li, Mark Lee, and Shengjie Sun. 2024. Adaptive layer sparsity for large language models via activation correlation assessment. *Advances in Neural Information Processing Systems*, 37:109350–109380.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, and 1 others. 2023. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pages 22137–22176. PMLR.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual associations in gpt. In *Advances in Neural Information Processing Systems*, volume 35, pages 17359–17372.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *Proceedings of ICLR Conference Track*, Toulon, France. Published online: https://openreview.net/group?id=ICLR.cc/2017/conference.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32.

Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. 2018. The building blocks of interpretability. *Distill*.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, S Arora, and 1 others. 2024. 2 olmo 2 furious. *arXiv preprint arXiv:2501.00656*.

Paul Smolensky, Richard McCoy, Roland Fernandez, Matthew Goldrick, and Jianfeng Gao. 2022. Neurocompositional computing: From the central paradox of cognition to a new generation of ai systems. *AI Magazine*, 43(3):308–322.

Aaquib Syed, Can Rager, and Arthur Conmy. 2024. Attribution patching outperforms automated circuit discovery. In *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pages 407–416.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30.

Andreas Veit, Michael J Wilber, and Serge Belongie. 2016. Residual networks behave like ensembles of relatively shallow networks. *Advances in neural information processing systems*, 29.

Elena Voita, Javier Ferrando, and Christoforos Nalmpantis. 2023. Neurons in large language models: Dead, n-gram, positional. https://arxiv.org/abs/2309.04827.

Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. 2022. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. *arXiv preprint arXiv:2211.00593*.

An Yang, Baosong Li, B Yang, B Zhang, B Hui, B Zheng, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. 2022. Moefication: Transformer feed-forward layers are mixtures of experts. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 877–890.

# A Graph Traversal

The graph traversal algorithm used for trace extraction is described in Algorithm 1.

# B Code and Resources

We developed the code for extracting and evaluating the trace from Hugging Face[5] models. We modified LLM's modeling source code from Hugging Face in order to (1) extract LLM's internal representations when constructing the computational graph, and (2) ablate edges during the evaluation. The code and data required to reproduce the experiments will be made publicly available on a Github repository upon acceptance.

[5] https://huggingface.co/

# C Methods: Additional Information

## C.1 Estimating Computational Density

To capture the global sparsity of a model across different relative trace sizes $s$, we define the scalar *Computational Density* metric $\mathcal{C}$ as the Area Under the Curve of the reconstruction error plotted against the log of the trace size. This metric is *instance-wise*, meaning that we get one $\mathcal{C}$ per model prediction in the dataset. We obtain it by evaluating the traces on a fixed grid of size $S = \{s_0, s_1, \ldots, s_K\}$, where $0 = s_0 < s_1 < \cdots < s_K = 1$. We estimate $\mathcal{C}$ using the trapezoidal rule:

$$D = \frac{1}{2} \sum_{k=1}^{K} (s_k - s_{k-1}) \cdot [\epsilon(s_k) + \epsilon(s_{k-1})] \quad (5)$$

where $\epsilon(s) = \delta_{TV}(s)$ is the reconstruction error for a trace of size $s$. Intuitively, $\mathcal{C}$ measures the model's resistance to pruning:
*Low $\mathcal{C}$ (Sparse):* The error $\epsilon(s)$ drops rapidly to zero even for small $s$, implying that a few edges suffice to replicate the computation;
*High $\mathcal{C}$ (Dense):* The error remains significant until $s$ approaches 1, implying that information is distributed diffusely across the graph.

## C.2 Grid Size For Trace Extraction

In order to extract the trace at different sizes, we empirically set $s \in [1e^{-5}, 2e^{-5}, 4e^{-5}, 8e^{-5}, 1e^{-4}, 2e^{-4}, 4e^{-4}, 8e^{-4}, 1e^{-3}, 1.2e^{-3}, 1.4e^{-3}, 2e^{-3}, 3e^{-3}, 4e^{-3}, 6e^{-3}, 8e^{-3}, 1e^{-2}, 2e^{-2}, 4e^{-2}, 6e^{-2}, 8e^{-2}, 1e^{-1}, 2e^{-1}, 4e^{-1}, 6e^{-1}, 8e^{-1}]$. These values span diverse trace sizes from sparse to dense, and reconstruction errors, from random to almost-perfect reconstruction. They were obtained by manually testing them on a few instances.

## C.3 Model Zoo

The LLMs used in this work are detailed in Table 2.

# D Additional Results

## D.1 Random trace extraction baseline

As a sanity check, we compare our trace extraction method to a random one. The approach is identical to Algorithm 1, but the importance function is

**Algorithm 1** Greedy BeFS for Trace Construction

**Require:** Computational graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, Output node $h_n^L$, Edge budget $s$, Importance function $\mathcal{I}$
**Ensure:** Trace graph $\mathcal{T}_s = (\mathcal{V}_s, \mathcal{E}_s)$

```
V_s ← {h_n^L}, E_s ← ∅
PQ ← empty max-priority queue
for each incoming edge e = (u, h_n^L) ∈ E connected to h_n^L do
    PQ.insert(e, I(e))                          ▷ Seed the priority queue with initial edges
end for
while PQ is not empty and |E_s| < s do
    e = (u, v) ← PQ.extract_max()               ▷ Select edge with highest score
    E_s ← E_s ∪ {e}
    if u ∉ V_s then
        V_s ← V_s ∪ {u}
        for each incoming edge e' = (w, u) ∈ E connected to u do
            PQ.insert(e', I(e'))                ▷ Expand frontier
        end for
    end if
end while
return T_s = (V_s, E_s)
```

Table 2: Overview of the models used in this study. All models are decoder-only Transformers. *Parameters* refers to total parameters. Training data size is denoted in tokens ($T$ for Trillion, $B$ for Billion).

| Model Name | Parameters | Tokens | Hugging Face ID | # (Valid Instances) |
|---|---|---|---|---|
| OLMo-2-1124-7B (OLMo et al., 2024) | 7.3B | 4.0T | `allenai/OLMo-2-1124-7B` | 4921 |
| OLMo-2-1124-13B (OLMo et al., 2024) | 13.7B | 5.0T | `allenai/OLMo-2-1124-13B` | 4975 |
| Qwen3-8B-Base (Yang et al., 2025) | 8.2B | 36.0T | `Qwen/Qwen3-8B-Base` | 4944 |
| Qwen3-14B-Base (Yang et al., 2025) | 14.8B | 36.0T | `Qwen/Qwen3-14B-Base` | 4959 |
| Qwen2.5-7B (Yang et al., 2025) | 7.6B | 18.0T | `Qwen/Qwen2.5-7B` | 4954 |
| Qwen2.5-14B (Yang et al., 2025) | 14.7B | 18.0T | `Qwen/Qwen2.5-14B` | 4275 |
| Llama-3.1-8B (Grattafiori et al., 2024) | 8.0B | 15.0T | `meta-llama/Llama-3.1-8B` | 4953 |
| Mistral-7B-v0.1 (Jiang et al., 2024) | 7.3B | Not disclosed | `mistralai/Mistral-7B-v0.1` | 5000 |
| DeepSeek-LLM-7B (Bi et al., 2024) | 7.0B | 2.0T | `deepseek-ai/deepseek-llm-7b-base` | 4893 |
| Phi-4 (Abdin et al., 2024) | 14.0B | 9.8T | `microsoft/phi-4` | 4964 |

replaced by a random attribution that samples importance weights from a uniform distribution. As the proportion of residual, MLP and attention edges is highly unbalanced (dominated by attention), a naive random sampling would disfavor residuals and MLPs. Therefore, we use a stronger baseline that, at each step, selects the residual and MLPs edges first. We use the same inputs described in Section 3. We observe that the trace extraction method used in this paper outperforms the random baseline at any size (for an equivalent reconstruction score, our method leads to a smaller trace in average). Results are shown in Fig. 6.

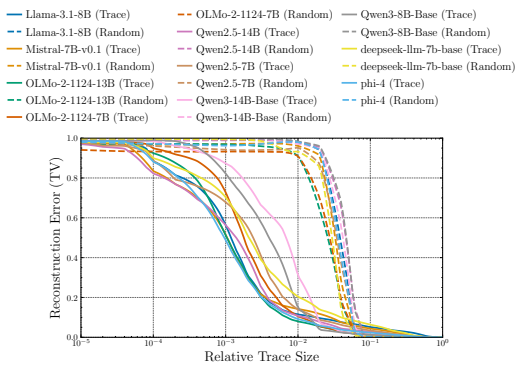



Figure 6: We compare our trace extraction with a strong random baseline that selects the MLPs and residual edges first. The trace extraction method used in this paper outperforms the random baseline at any size.

### D.2 Sufficient vs. Necessary

As a second sanity check, we measure the effect of ablating only the trace. Specifically, we zero

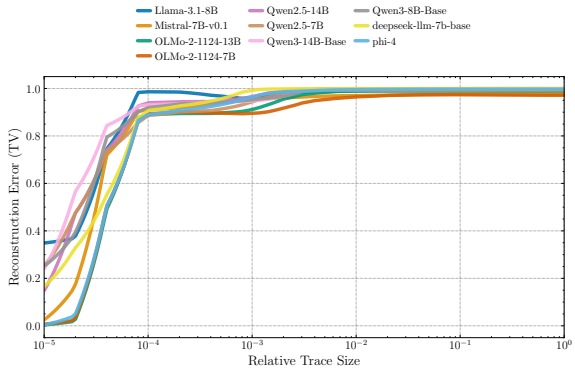



Figure 7: We measure the effect of zeroing the edges belonging to the trace. At $s < 10^{-4}$, the reconstruction error is low, demonstrating the resilience of LLMs to a small number of edge ablations. However, quickly (starting from $s = 10^{-4}$), ablating the trace leads to a very high reconstruction error, indicating that these edges are necessary to reconstruct the prediction. This contrasts with Fig. 2, which illustrates the complementary effect of zeroing edges that do *not* belong to the trace.

the edges in the computational graph that belong to the trace (instead of keeping only those edges). Because it is trivial that ablating residual edges would lead to a strong drop in performance, we never ablate them in this setup. Thus, we measure the effect of ablating only the MLP and attention edges that are in the trace. We use the same inputs described in Section 3. Results are shown in Fig. 7. We observe that the edges identified by our trace extraction method are indeed necessary to reconstruct a faithful output.

### D.3 Density vs. Entropy, Loss and Frequency

Fig. 8 shows the correlation between density and the entropy of the LLM output, the Wikitext-based frequency of the input, and the loss.

### D.4 Pairwise correlation of LLMs

Fig. 9 shows the Spearman correlation of $\mathcal{C}$ across pairs of distinct language models evaluated on identical sentences.

### D.5 Structure of the Trace (all models)

Fig 10 provide the structure of the trace for all models tested in this study.

### D.6 Component frequency (all models)

Fig 11 provides component frequency at different sizes for all models.

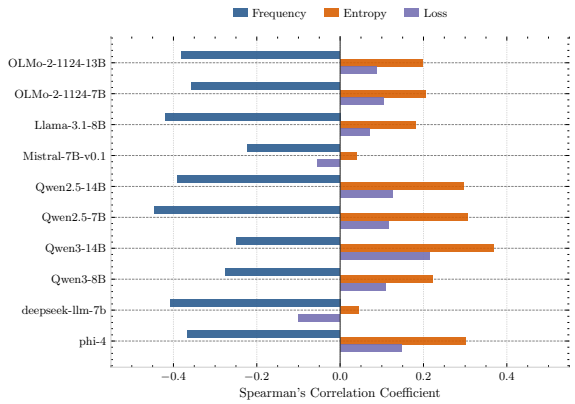



Figure 8: Spearman correlation between computation density and: the entropy of the output probability distribution, the language modeling loss, and the frequency of the top-1 predicted token. All correlations are highly statistically significant ($p < 0.01$).

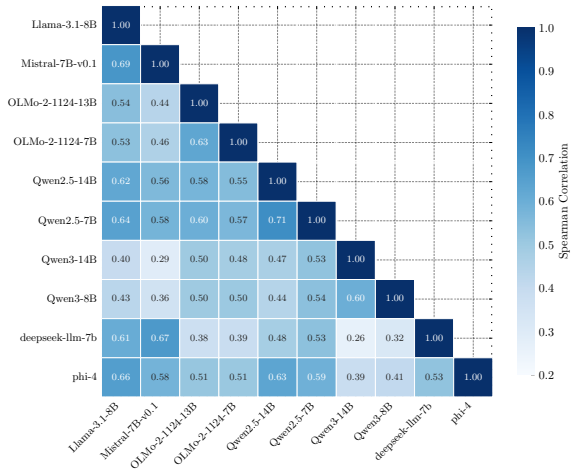
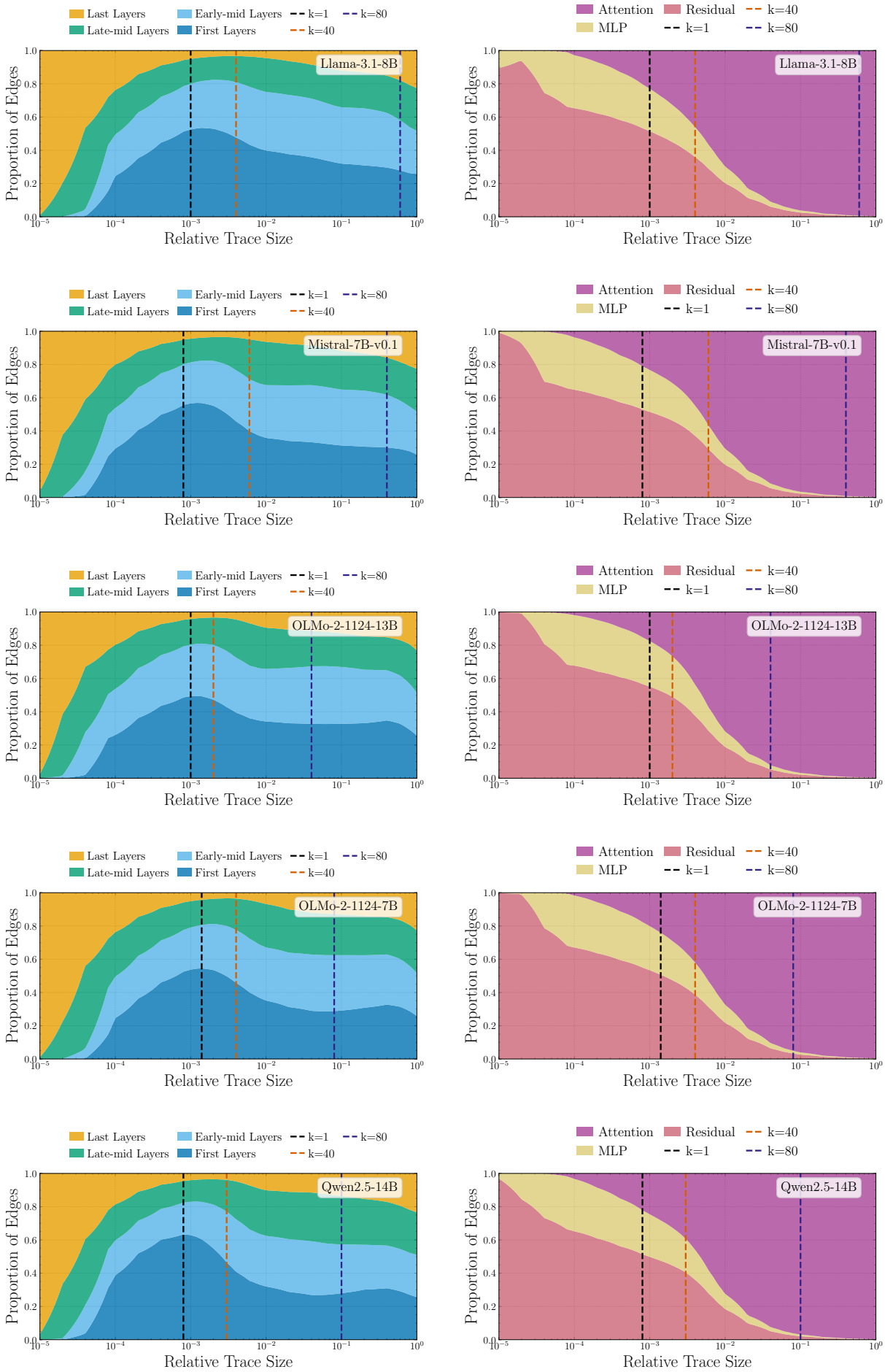



Figure 9: We compute the Spearman correlation of $\mathcal{C}$ across pairs of distinct language models evaluated on identical sentences.


Last Layers
Early-mid Layers
k=1
k=80
Late-mid Layers
First Layers
k=40
Llama-3.1-8B
Mistral-7B-v0.1
OLMo-2-1124-13B
OLMo-2-1124-7B
Qwen2.5-14B
Attention
Residual
MLP
Proportion of Edges
Relative Trace Size

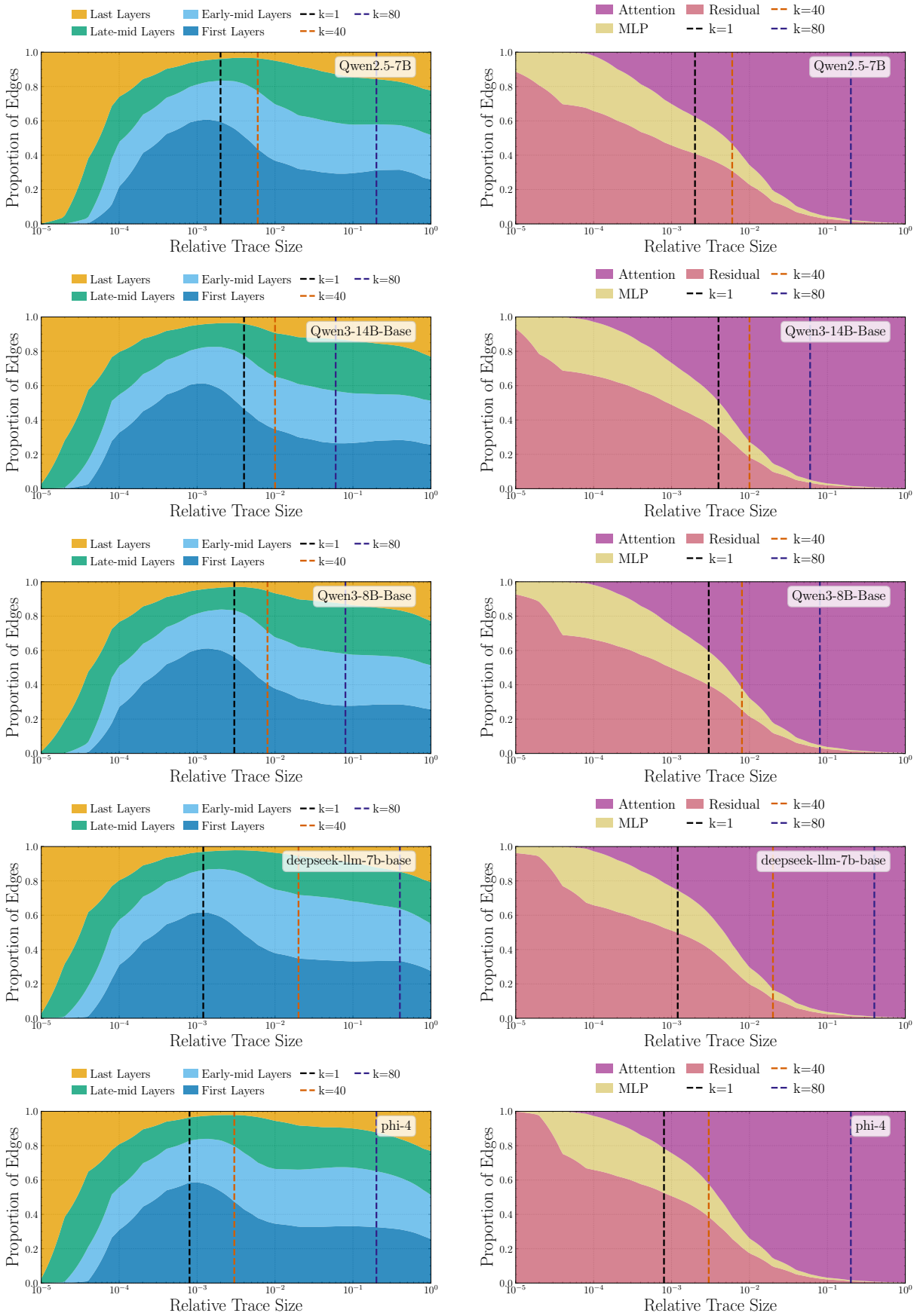



Figure 10: Structure of the trace at different sizes.

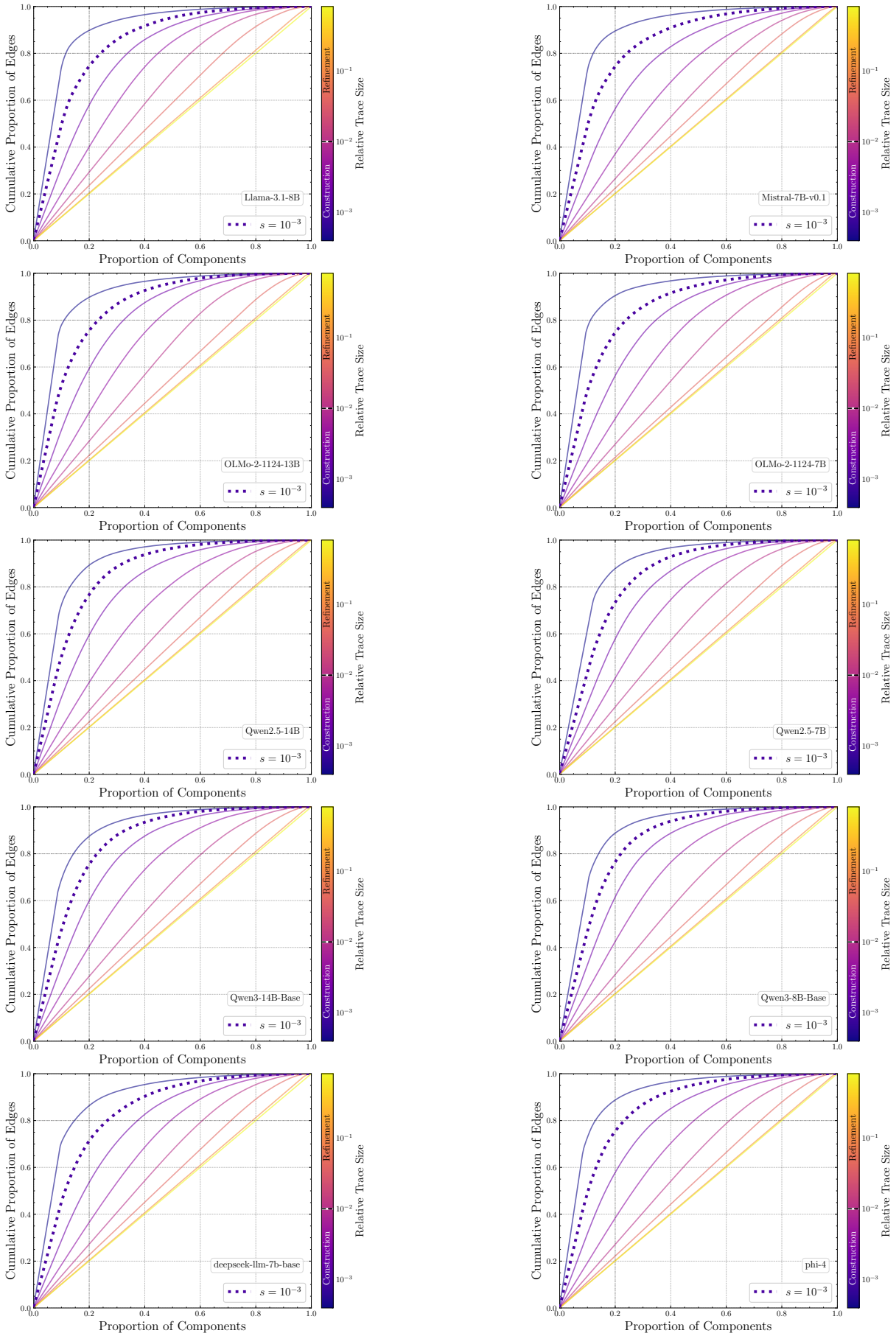



Figure 11: Component frequency at different sizes for all models.