TinySQL: A Progressive Text-to-SQL Dataset for Mechanistic Interpretability Research

Abir Harrasse[*1,5] **Philip Quirke**[*1] **Clement Neo**[*2]
Dhruv Nathawani[3] **Luke Marks**[1] **Amir Abdullah**[4,1]
[1]Martian [2]Apart Research [3]NVIDIA
[4]Thoughtworks [5]Mohammed VI Polytechnic University

Abstract

Mechanistic interpretability research faces a gap between analyzing simple circuits in toy tasks and discovering features in large models. To bridge this gap, we propose text-to-SQL generation as an ideal task to study, as it combines the formal structure of toy tasks with real-world complexity. We introduce TinySQL, a synthetic dataset, progressing from basic to advanced SQL operations, and train models ranging from 33M to 1B parameters to establish a comprehensive testbed for interpretability. We apply multiple complementary interpretability techniques, including Edge Attribution Patching and Sparse Autoencoders, to identify minimal circuits and components supporting SQL generation. We compare circuits for different SQL subskills, evaluating their minimality, reliability, and identifiability. Finally, we conduct a layerwise logit lens analysis to reveal how models compose SQL queries across layers: from intent recognition to schema resolution to structured generation. Our work provides a robust framework for probing and comparing interpretability methods in a structured, progressively complex setting.

1 Introduction

The circuit discovery approach in mechanistic interpretability (MI) has largely focused on small models solving simple tasks, such as arithmetic (Quirke and Barez, 2024; Nanda et al., 2023a) and indirect object identification (Wang et al., 2023). While these studies revealed key mechanisms, reliance on toy tasks limits validation and comparison across interpretability methods. Recent work has shifted toward feature discovery using sparse autoencoders (SAEs) (Huben et al., 2023; Templeton et al., 2024), but linking these insights to circuit-level understanding remains difficult without intermediate tasks that support both approaches.

Text-to-SQL generation provides an ideal middle ground, as SQL's formal structure makes it more tractable than general language generation while still requiring natural language understanding. This enables systematic comparisons of interpretability methods within a task complex enough to reflect real-world challenges.

Existing text-to-SQL datasets like Spider (Yu et al., 2019b) and WikiSQL (Zhong et al., 2017) are too complex and noisy for rigorous interpretability analysis. To address this, we introduce TinySQL, a curated dataset that enables controlled analysis of how transformers learn and generate SQL queries. TinySQL progresses through text-to-SQL tasks of increasing complexity, isolating key aspects of the generation process while maintaining structural consistency. Alongside the dataset, we train and release models of 33M, 0.5B, and 1B parameters, providing a comprehensive testbed for interpretability research.

Our analysis combines multiple complementary interpretability techniques to study SQL generation. Using Edge Attribution Patching (EAP), we identify minimal circuits supporting SQL query generation. In parallel, our SAE analysis reveals consistent patterns in how the same base model fine-tuned on different tasks yield similar interpretable heads. Lastly, our logit lens analysis reveals a layered computation pattern: early layers capture query intent, middle layers resolve schema elements, and later layers synthesize structured SQL outputs.

The contributions of our paper are as follows:

1. We introduce TinySQL, a structured text-to-SQL dataset bridging toy tasks and real-world applications. By controlling SQL complexity across five subskills and releasing models trained on different subsets of TinySQL, we provide a robust testbed for mechanistic interpretability.
2. We apply multiple interpretability methods

* Equal contributions. Correspondence to: philip@withmartian.com.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 29245–29273

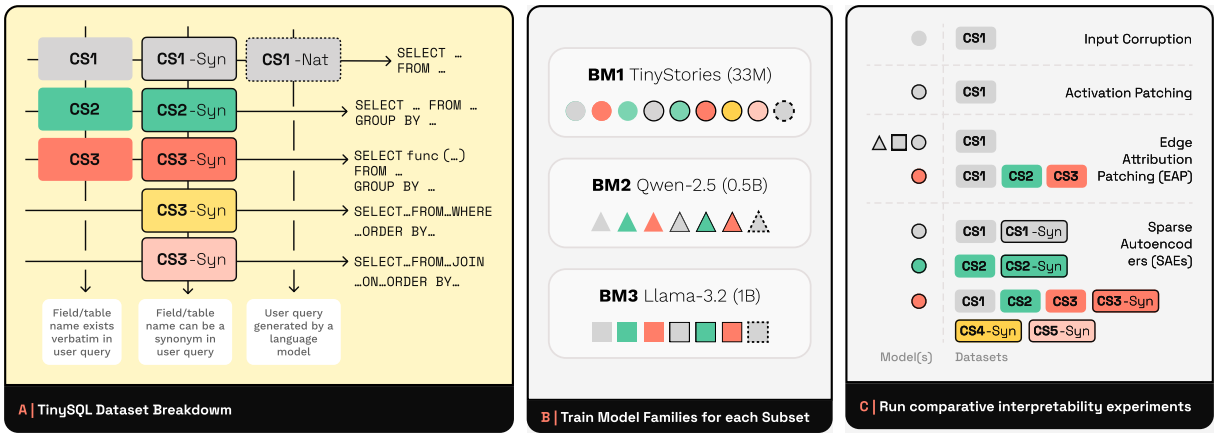


Figure 1: (a) TinySQL is broken down into 9 subsets of varying complexities, across both SQL query and user query axes. (b) We train and release a comprehensive set of models on each dataset subset.(c) We apply MI techniques across various configurations to understand model behavior and compare results.

(EAP and SAE) to identify circuits across SQL subskills. We compare them, show their local minimality, and validate their relevance through performance and identifiability studies.

3. We present a layerwise logit lens analysis revealing how models compose SQL queries: early layers identify intent, middle layers resolve schema references, and later layers integrate all elements into structured outputs.

Our work advances understanding of neural networks in structured query processing and outlines a roadmap for future MI studies. By identifying limitations and breaking points, we define clear boundaries for reliable application and highlight areas needing new approaches.

2 Background

2.1 Mechanistic Interpretability Approaches

Early mechanistic interpretability (MI) research focused on discovering specific computational circuits in small language models through carefully controlled tasks. These studies identified mechanisms for indirect object identification (Wang et al., 2023), docstring generation (Stefaan Hex, 2023), and number incrementation (Nanda et al., 2023b), but remained limited by their reliance on highly specialized tasks. However, these investigations remained limited to highly specific tasks that offered few natural paths for extension. While automated methods like ACDC (Conmy et al., 2023) and EAP (Syed et al., 2023) helped automate circuit discovery, interpreting these circuits still required extensive manual analysis.

Beyond mechanistic interpretability, recent work has leveraged synthetic languages and controlled data to probe model reasoning. Chiang and Lee (2022) demonstrate that pre-training on artificial language sequences with explicit and implicit token dependencies improves downstream performance. Allen-Zhu and Li (2023) use CFG-based synthetic grammars to study recursive reasoning, revealing interpretable internal structures and algorithmic attention patterns in generative models. These results highlight the value of structured tasks, and support our use of Text-to-SQL as a semantically grounded, interpretable setting.

Text-to-SQL generation offers a unique middle ground for advancing MI research. The task combines the formal structure needed for circuit analysis with the semantic complexity of natural language understanding, making it an ideal testbed for bridging interpretability methods across scales.

Recent work with SAEs (Huben et al., 2023; Templeton et al., 2024) has enabled large-scale analysis of model behavior. Marks et al. (2024) use SAE features to uncover interpretable circuits for tasks like subject-verb agreement, demonstrating the promise of this approach. However, such studies often focus on a single model and task, leaving questions about generalization unresolved. To address this, we introduce multiple Text-to-SQL

Code available at: https://github.com/withmartian/TinySQL, Datasets, Models and SAEs at: https://huggingface.co/collections/withmartian/tinysql-6760e92748b63fa56a6ffc9f

variants across models, enabling systematic investigation of behavioral variation.

2.2 Text-to-SQL Generation

Text-to-SQL is a task where models generate SQL queries from natural language requests. Given a database schema and a query like "Show me all employee salaries", the model must produce the correct SQL query "`SELECT salary FROM employees`".

Text-to-SQL is a step up from toy tasks in MI research while retaining structure for rigorous analysis. Each query maps to a single answer, allowing clear evaluation, unlike general code generation. The task also exhibits systematic patterns, such as "how many" mapping to `COUNT` and "highest" to `ORDER BY DESC`.

2.3 Text-to-SQL Datasets

The field evolved from simpler beginnings with `WikiSQL` (Zhong et al., 2017), which focused on single-table queries from Wikipedia to enable large-scale evaluation. `Spider` (Yu et al., 2019b) established a foundation for text-to-SQL research by focusing on cross-domain generalization through multiple tables and schemas. Its variants explore different aspects of complexity: `Spider-Syn` (Gan et al., 2021) tests semantic understanding by replacing schema terms with synonyms, while `Spider-Real` (Deng et al., 2020) aligns more closely with natural user queries by omitting explicit column names. These datasets collectively enable research into model robustness across synthetic and real-world scenarios.

Recent text-to-SQL datasets have advanced the field by incorporating conversational structure and domain-specific challenges. CoSQL (Yu et al., 2019a) introduces multi-turn interactions grounded in real user dialogues. SParC (Yu et al., 2019c) similarly emphasizes context-dependent SQL generation, where models must resolve references across sequential questions in cross-domain settings. In the medical domain, MIMICSQL (Wang et al., 2020) features naturally varied questions about electronic health records, including paraphrasing and under-specification.

Alongside these realistic benchmarks, recent efforts like Gretel's `Synthetic-Text-To-SQL` (Meyer et al., 2024) and `SQL-create-context` (b mc2, 2023) seek to synthesize datasets that capture naturalistic query patterns while retaining control over generation. These works reflect growing emphasis on bridging the gap between controlled data creation and real-world applicability.

However, these existing datasets prioritize training high-performance models through diverse, complex queries rather than supporting controlled experiments. Even synthetic resources like `Sythnetic-Text-To-SQL` lack the systematic progression of complexity needed for mechanistic interpretability research. This gap motivates our development of TinySQL, which draws inspiration from machine learning's long history of using synthetic data to control task complexity and enable clear evaluation. By providing carefully controlled progression of query complexity while maintaining consistent structure, TinySQL bridges the divide between performance-focused datasets and the controlled environment needed for interpretability research.

3 The TinySQL Dataset

To enable rigorous MI research, we need datasets that progress systematically from toy-like tasks suitable for circuit analysis to realistic tasks that capture core text-to-SQL challenges. Each level must provide enough examples for both model training and detailed experimentation. TinySQL implements this through two independent axes: SQL command complexity and language variation.

This design enables comparative analysis - training models on different complexity levels to study base task handling or examining how language variation impacts SQL generation.

3.1 Dataset Structure

TinySQL structures tasks along two dimensions: SQL complexity and query language variation, isolating specific model behaviors while maintaining consistent task patterns.

SQL Command Levels. Tasks are organized into five levels of increasing SQL complexity:

- **CS1 (Basic Queries)** focuses on fundamental `SELECT-FROM` operations. Example: "Show me the salary from employees" → `SELECT salary FROM employees`
- **CS2 (Ordered Queries)** introduces `ORDER BY` clauses. Example: "Show employee salaries from highest to lowest" → `SELECT salary FROM employees ORDER BY salary DESC`
- **CS3 (Aggregation Queries)** adds aggregation functions and grouping. Example: "How

many employees are in each department?" → `SELECT COUNT(id) FROM employees GROUP BY department`

- **CS4 (Filter Queries)**adds type-aware `WHERE` clauses to 80% of queries. Example: "Find entries where salt contains 'z' ordered by score and location" → `SELECT salt, max_score FROM data WHERE salt LIKE '\%z\%' ORDER BY max_score DESC, location DESC`

- **CS5 (Join Queries)** introduces multi-table queries with `JOIN` clauses on matching column types. Example: "List workflow templates joined with pairs on user ID ordered by score and certification" → `SELECT name, city FROM workflow_templates JOIN pairs ON workflow_templates.user_id = pairs.user_id ORDER BY average_score ASC, certification ASC`.

Examples of each Command Level are presented in Appendix.C.3

Query Language Variants. For each command level, we provide three variants of increasing linguistic complexity:

- The **Base (CSx)** variation uses rigid templates where field and table names exactly match the schema, providing a controlled baseline for studying SQL generation mechanisms.

- The **Synonyms (CSx_Syn)** variation introduces semantic mapping between query terms and schema fields (e.g., "earnings" mapping to "salary"), testing models' ability to handle semantic equivalences. This is inspired by synonym-based datasets like `Spider-Syn` (Gan et al., 2021) and `ADVETA` (Pi et al., 2022).

- The **Natural (CSx_Nat)** variation allows flexible natural language phrasing while targeting the same SQL operations. This most closely resembles real-world usage, and it is currently limited to CS1 queries due to the difficulties of ensuring dataset quality.

Dataset statistics. The complete dataset consists of nine task variants (`CS1/2/3`, `CS1/2/3/4/5_Syn`, and `CS1_Nat`), each containing 100,000 examples generated through a systematic data creation pipeline that ensures consistency with our design principles. The average query length using the `meta-llama/Llama-3.2-1B-Instruct` tokenizer ranges from 16.48 tokens for CS1_Syn, 38.76 tokens for CS2_Syn, 42.39 tokens for CS3_Syn, 53.73 tokens for CS4_Syn and 61.61 tokens for CS5_Syn, with similar figures for the non synonym variants.

Employing the approach of Mitsopoulou and Koutrika (2025), we compare the English portion of TinySQL to prior Text-to-SQL datasets and find it to be both lexically and semantically more challenging than prior datasets. This is reflected in three key metrics: (a) rarity, the ratio of rare words to content words in a natural language (NL) question (Read, 2000); (b) lexical density, the ratio of content words to total words (Johansson, 2008); and (c) Flesch reading ease, a readability measure where lower scores indicate more complex sentences with longer structures and more syllables per word on average (Flesch and Gould, 1949). As shown in Table 1, TinySQL scores higher on rarity and lexical density, and lower on readability, compared to most previous Text-to-SQL datasets.

Dataset	Rarity	Lexical Density	Readability
CS1_Syn	**0.60**	0.58	45.5
CS2_Syn	0.52	0.56	29.4
CS3_Syn	0.45	**0.61**	29.9
CS4_Syn	0.41	0.58	28.7
CS5_Syn	0.44	0.59	**25.7**
IMDb	0.23	0.43	88.0
Yelp	0.28	0.51	85.0
Academic	0.24	0.54	65.0
Spider	0.21	0.55	89.0
Geo	0.29	0.58	95.0
Restos	0.26	0.56	90.0
MIMICSQL	0.25	0.61	68.0
K-DBQA	0.26	0.59	75.0
BIRD	0.30	0.57	73.0
Atis	0.35	0.46	78.0
Scholar	0.31	0.52	77.0
Advising	0.25	0.44	80.0

Table 1: Dataset Statistics Comparison to Previous Text-to-SQL Datasets

For details on our dataset generation methodology, including schema design, query construction, and instruction templating for `CSx`, `CSx_Syn`, and `CSx_Nat`, please refer to Appendices C.1 and C.2.

3.2 Models Trained

We fine-tune three base models to perform the text-to-SQL task. We call them BM1, BM2, and BM3:

- **BM1:** `TinyStories-33M` (Eldan and Li, 2023), a 2-layer model with 33M parameters.

- **BM2:** `Qwen2.5-0.5B-Instruct` (Team, 2024), a model of approx. 500M parameters.
- **BM3:** `Llama-3.2-1B-Instruct` (Grattafiori, 2024), a 1B-parameter Llama variant.

Our datasets comprise five SQL complexity levels (`CS1`, `CS2`, `CS3`, `CS4` and `CS5`), combined with either base or synonym versions for the first three variants, resulting in nine variations (`CS1`, `CS2`, `CS3`, `CS1_Syn`, `CS2_Syn`, `CS3_Syn`, `CS4_Syn`, `CS5_Syn` and `CS1_NAT`. We use shorthand like `BM1_CS1` and `BM1_CS1_Syn` for `BM1` fine-tuned on `CS1` base and synonym versions, extending similarly to other variants. We exclude `CS1_NAT` from all trainings. We train BM1 on all 8 remaining variants, while we train BM2 and BM3 only on the 6 CS1 through to CS3 variants. This produces 20 fine-tuned checkpoints. Refer App.D for training and architecture details, and App.G.1.1 for examples of the model input format.

Performance Overview. All three base models reliably converge on all dataset variants. For example, fine-tuned `BM1` achieves over 85% exact-match accuracy on CS3_Syn, the most challenging variant, while `BM2` and `BM3` exceed 98% on most tasks. See App.D and Tab.8 for details. These results show that even smaller transformer architectures can learn robust text-to-SQL generation on TinySQL, providing a strong foundation for MI analysis.

4 Experiments

Our experiments aim to uncover how language models internally represent and execute structured code generation tasks, using interpretability tools such as Edge Attribution Patching, Sparse Autoencoders, and Logit Lens analysis.

4.1 Mechanistic Interpretability Techniques

In circuit analysis, we view the model as a computational graph $G = (V, E)$, where circuits are subgraphs implementing specific functions (Conmy et al., 2023; Wu et al., 2024). The resolution of these nodes varies from layers, to sublayers (attention, MLP), and to individual components (attention head, MLP neuron).

Edge Attribution Patching (EAP). Given clean input x_c and corrupted input x_d, EAP measures how each edge E in the computational graph affects the model's behavior. For edge E with clean activation e_c and corrupted activation e_d, we approximate its causal effect as $\Delta_E L = (e_d - e_c) \cdot \nabla_e L(x_c)$, where L is the task-specific metric. The absolute attribution score $|\Delta_E L|$ ranks edge importance, allowing identification of the most crucial edges for a given task (Syed et al., 2023). The subnetwork is typically obtained by keeping the top k edges, where k is a chosen hyperparameter.

Sparse Autoencoder Features. Neural networks often learn to encode multiple distinct features in overlapping directions, which makes their internal representations difficult to interpret. Sparse Autoencoders (SAEs) address this by learning an encoder $E : \mathbb{R}^n \to \mathbb{R}^m$ and decoder $D : \mathbb{R}^m \to \mathbb{R}^n$ that reconstruct model activations x while enforcing sparsity in the latent space, minimizing $|x - D(E(x))|^2$ under appropriate constraints. This sparsity forces the autoencoder to learn a basis set of features that activate independently and rarely, which often correspond to human-interpretable concepts (Huben et al., 2023).

4.2 Basic SQL Generation

Input Corruption. To probe how `BM1_CS1` uses context, we replace table or field names in either the instruction or schema (e.g., "table_a" vs. "table_b"). If performance remains stable, the model likely ignores the corrupted source.

We find the model relies on schema context for table names and on the instruction for field names, an effective shortcut in CS1, where names match. To prevent this, we created synonym variants (`CSx_Syn`) that require mapping between equivalent terms. We now focus on these variants to study more realistic behaviors.

4.3 Finding Locally Minimal Text-to-SQL Circuits via EAP and Ablation

To identify the locally-minimal circuits responsible for SQL generation, we analyzed how the model process components like table name, field names, and `ORDER BY` clauses. We generated 15 batches of 100 paired clean and corrupted prompts for each SQL feature (full set of features detailed in App. G.1.1). We then performed EAP, selecting the 10 most important edges ranked by the task-specific metric L, which produces scores near 0 for essential edges and near 1 for less relevant ones.

$$L = \frac{\ell(x_{\text{clean}}|\text{do}(E = e_{\text{patch}})) - \ell(x_{\text{corr}})}{\ell(x_{\text{clean}}) - \ell(x_{\text{corr}})} \quad (1)$$

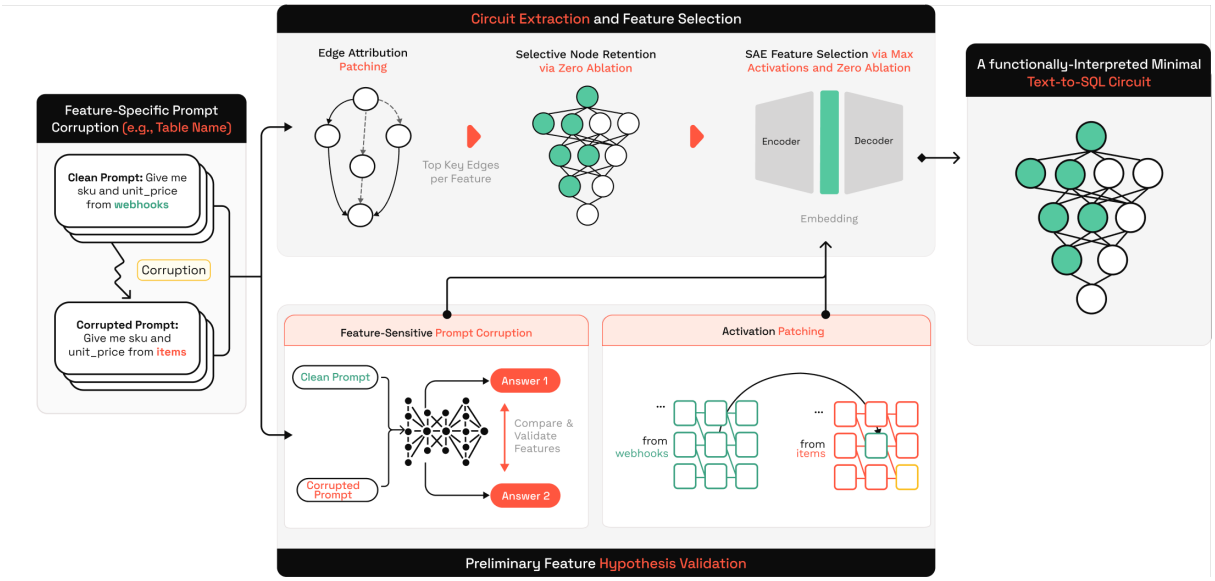


Figure 2: To extract and interpret text-to-SQL functionality, we use Edge Attribution Patching to identify key connections, Selective Node Retention to create a minimal working circuit, and SAE Feature Selection to interpret node functionality. We also use prompt corruption and activation patching to form hypotheses on how the model implements functions.

For each feature, we ran EAP four times across all combinations of semantic field and table name variations. While we obtain 10 edges per batch, we retain only the edges that appear across batches consistently, requiring 100% consistency for CS1 and CS2, and 80% for CS3 due to its increased complexity. The full results appear in Figures 14, 15, and 16 in the appendix. After running EAP for all features, we obtain a set of edges per feature. We then take the set of attention heads associated with these edges across all features, and retain them during the following ablation study.

Attention Head Ablation. To validate our findings, we ablated all attention heads except those linked to EAP-identified edges and measured Recovered Accuracy. We tested both mean- and zero-ablation (the latter being more informative). Figure 17 shows how performance varies under selective component removal.

Table 2 shows that EAP performed best on smaller BM1 models, identifying precise task-relevant heads. In larger models (BM2/BM3), top edges often involved MLPs, leading to few head selections and low recovered accuracy (5–10%). To compensate, we retained entire layers when any head was marked important, improving accuracy at the cost of granularity. In the next section, we address this limitation using more targeted SAE circuits.

4.4 Sparse Circuit Identification: Method and Results

To identify interpretable latent features relevant to SQL task performance, we train Sparse Autoencoders (SAEs) on attention head output and MLP activations from the decoder blocks of our transformer models. We focus our main analysis on the BM1 family spanning all the datasets, with results for BM2 provided in Appendix.I.

Experimental Setup We begin by collecting attention heads and MLP activations from each of $\texttt{BM}_2\texttt{\_CS}_1\texttt{\_Syn}$ and $\texttt{BM}_1\texttt{\_CS}_i\texttt{\_Syn}$ models when applied to its corresponding $\texttt{CS}_i$ datasets, for $i \in \{1, 3, 4, 5\}$. To analyze how models trained on more complex tasks generalize to simpler ones, we also create a *descending complexity* evaluation set by applying $\texttt{BM1\_CS3\_Syn}$ to $\texttt{CS1}$ and $\texttt{CS2}$ data.

We train SAEs on the collected activations using the Belrose (2024)'s package, following the top-k training regime Gao et al. (2024) , with $k = 16$, an expansion factor of 4, and a learning rate of 8×10^{-4} for 6 training epochs. Across all models and datasets, we find that the Fraction of Unexplained Variance (FVU) remains below 0.05, indicating that the SAEs faithfully reconstruct the input representations. Training each SAE takes approximately 1 hour for BM1 and 2 hours for BM2, all on a single A100 GPU.

Model	Dataset	Heads		Recovered Accuracy
		Layer 0	Layer 1	
BM1_CS1_Syn	CS1	11,3,1,8,15,14,13,7	13,3,7,14,15,11	85.4%
BM1_CS3_Syn	CS1	11,14,3,8,2,7,0,9,6,13,5,1	2,3,5,15,11,8,13	97.6%
BM1_CS3_Syn	CS2	11,3,7,14,2,8,13,1,10,4	3,2,15,5,11,8	74.6%
BM1_CS3_Syn	CS3	11,14,3,0,13,2,7,6,4,1	2,15,3,8,1,7,5,11	78.3%
		Layers		
BM2_CS1_Syn	CS1	21,8,20,9,0,1,13,16,11,14,7,18,15,23		71.6%
BM3_CS1_Syn	CS1	0,5,4,14,8,6,1,9,11,10,15,13,7,12		82%

Table 2: Minimal circuits identified for Text-to-SQL tasks, showing the attention heads found using EAP and their recovered accuracy. For BM1 models, heads are split between Layer 0 and Layer 1, while for BM2 and BM3 they are reported across all layers.

Algorithm 1 SAE-Based Circuit Identification via Similarity-Guided Feature Selection and AUC Mapping

1: **Collect Activations:** Extract residual activations $\{x^{(j)}\}_{j=1}^{N}$ at hook h and position n
2: **Compute SAE Feature Activations:** Project activations through SAE encoder to obtain sparse codes $\{z^{(j)}\}$
3: **Similarity-Guided Feature Selection:** Iterate $k = 1$ to $K_{\max}$: compute diff vector Δ_k by ablating lower-ranked features, apply Δ_k to the model, and measure output similarity. Stop when similarity $\geq$ threshold or improvement flattens. Set $k^\star = k$
4: **Map to Model Components via AUC Attribution:** Let $a_f = \|W_{\text{dec}}[f]\|$ for each selected feature $f \in F$, and $A = \sum_{f \in F} a_f$. Select smallest m such that:

$$m^\star = \underset{m}{\operatorname{argmin}} \left| \frac{\sum_{i=1}^{m} a_i}{A} - \tau \right|$$

Feature Selection via AUC Scoring. To isolate task-relevant features, we follow the four-step procedure in Algorithm 1. We evaluate each circuit's fidelity using zero-ablation and report recovered accuracy in Table 3, with additional results on other combinations in Appendix. I.

5 Identifiability and Local Minimality of Derived Circuits

5.1 Identifiability via SAE-EAP Overlap

To assess the identifiability of circuits extracted using sparse autoencoders (SAEs), we compare them against circuits identified by Edge Attribution Patching (EAP). We quantify overlap between the two methods and contrast it with a random baseline. This comparison provides a new identifiability measure for SAE circuits.

Our results in Table 4 reveal a consistently higher overlap between EAP and SAE circuits than between EAP and random circuits (for which the accuracies are presented in Appendix.K). This supports the hypothesis that SAE-selected features align with the ground truth structure identified via causal intervention. Notably, Layer 2 exhibits more variability and lower overlap across methods, which is consistent with our earlier ablation study (Fig.19) indicating its minor role in SQL task processing, often involving only a handful of heads.

5.2 Local Minimality of Circuits

We also analyze the local minimality of the circuits (i.e., the property that no smaller circuit can be obtained by removing or editing components of the current "locally minimal") circuits identified by SAEs, by measuring the percentage of the total model they utilize. This metric reflects how sparse and efficient the discovered subcircuits are in reproducing task behavior. Table 5 summarizes these results across different combinations of models and datasets.

We also explored an alternative circuit extraction strategy based on mean ablation . While this approach produced more minimal circuits, their effectiveness was comparable to random circuits of the same size, suggesting poor alignment with ground-truth task structure. Thus, we opted not to include mean-ablated circuits in our main results. Detailed examples and ablation configurations are

Model	Dataset	Heads and MLPs		Recovered Accuracy
		Layer 0	Layer 1	
BM1_CS1_Syn	CS1	13, 8, 1, 10, 15, 3, 14, 9, 11 All MLP neurons	– –	80.64%
BM1_CS3_Syn	CS1	13, 8, 1, 15, 10, 12, 14, 7, 9, 3 846 MLP neurons	– –	85.71%
BM1_CS3_Syn	CS3	13, 8, 1, 15, 10, 12, 14, 9, 7, 3 All MLP neurons	– –	87.67%

Table 3: Recovered accuracy and layer-wise head selections for circuits identified via high-AUC SAE features. Each layer column is split into two rows per model (first row for attention heads and second for MLP neurons). Full results are in Appendix. H.

Model	Dataset	Layer 0	Layer 1
BM1-CS1-Syn	CS1	87.5% 52.62%	50% –
BM1-CS3-Syn	CS1	70% 74.85%	57% –
BM1-CS3-Syn	CS2	70% 54.81%	50% –
BM1-CS3-Syn	CS3	50% 36.16%	62.5% –

Table 4: Percentage overlap between EAP circuits and other methods across different datasets and layers. For each model and dataset, the first row shows overlap between SAE and EAP circuits, and the second row shows overlap between random (Rand) and EAP circuits

Model	Dataset	Percentage of Model Used
BM2-CS1-Syn	CS1	69.94%
BM1-CS1-Syn	CS1	42.69%
BM1-CS3-Syn	CS1	37.95%
BM1-CS3-Syn	CS2	42.69%
BM1-CS3-Syn	CS3	43.75%
BM1-CS4-Syn	CS4	44.79%
BM1-CS5-Syn	CS5	34.83%

Table 5: Percentage of Model Components Used by SAE Circuits

provided in Appendix. J.

For example, in `BM1-CS3-Syn` → `CS1`, the found circuit recovered 88.63% accuracy while using only 22.5% of model components. Other cases show similar patterns, with best circuits retaining high performance using only 20–30% of the model.

Finally, on `BM2-CS1-Syn` → `CS1`, focusing only on attention heads yielded circuits using just 3.17% of the model's heads while maintaining high accuracy. However, including MLPs raised usage to 60.71%, indicating that while attention sparsity is possible, full circuit function often relies on broader model interaction.

6 Two-Phase SQL Generation: Intent First, Grounding Later

To better understand how the model constructs SQL queries, we apply **LogitLens** analysis (Nostalgebraist, 2020). We observe a clear two-phase behavior in the generation process.

In the first phase, the model focuses on identifying the **intent** of the query by emitting appropriate *SQL keywords* (e.g., `SELECT`, `WHERE`, `GROUP BY`). These tokens receive higher logit probabilities in the earlier layers of the model, suggesting an early commitment to the structural form of the SQL command.

Subsequently, the model transitions into a second phase where it determines the appropriate **table names** to include in the query. This transition is evident from the delayed rise in logit probabilities for table-related tokens, which occurs at deeper layers, indicating the model is leveraging contextual information from the prompt to ground the query in schema elements.

We provide additional visualizations demonstrating similar trends across other benchmarks and model configurations in Appendix. M and a detailed logit lens analysis illustrating the two-phase token emergence across layers in Appendix. L

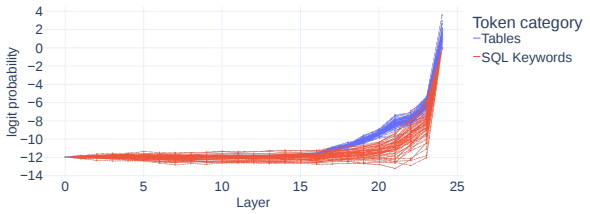


Figure 3: Emergence of SQL intent (via keywords) precedes table name resolution during generation in BM2-CS3. LogitLens reveals elevated logit probabilities for SQL keywords early, followed by table tokens at deeper layers.

7 Discussion

7.1 Ablation Validates Robustness–Granularity Tradeoff

Our ablation results reveal that zero-ablation recovers performance significantly better than random circuits, making it more reliable for identifying robust, task-relevant components. In contrast, mean-ablation yields smaller circuits but recovers performance similar to random baselines, suggesting weaker causal validity. Thus, zero-ablation is better for robustness, while mean-ablation may be useful for identifying compact, interpretable circuits when some loss in reliability is acceptable. Notably, across prompts within all classes, activation patching results vary significantly, even among prompts in the same class, highlighting the inherent variability in the circuits identified (see Appendix.F for examples).

7.2 Distributed Computation in SQL Generation

LogitLens analysis shows that SQL generation unfolds compositionally: early layers capture intent, middle layers resolve schema references, and later layers assemble the final query. However, EAP often reveals fragmented mechanisms scattered across layers, especially in larger models, highlighting the difficulty of isolating clean circuits and the limits of current interpretability tools at scale. This fragmentation is consistent with the observed variability in activation patching results, suggesting that circuit-level interpretations may differ notably even for similar inputs.

8 Conclusion

We present TinySQL, a dataset with increasingly complex SQL subsets and models of varying sizes. While not a replacement for real-world data, it supports controlled circuit discovery. Our analysis of EAP, SAE, and Logit Lens reveals both strengths and limitations, offering a testbed for advancing interpretability methods.

Acknowledgements

We thank Narmeen Oozeer for her invaluable early feedback on the project and for proposing Edge Attribution Patching as a linear approximation of the ACDC idea. We are also grateful to Shriyash Upadhyay for his thoughtful feedback and assistance with communication around the project. We appreciate Antia Garcia Casal for her help with figures and for shaping the overall style of our visualizations.

Limitations

While our approach provides meaningful insights into SQL generation mechanisms, several limitations remain. Interpretability results are sensitive to the choice of method and ablation strategy: zero-ablation offers robustness but inflates circuit size, while mean-ablation yields smaller circuits with unclear causal impact. Our EAP technique scales less effectively to larger models, and even the best-performing circuits recover only partial accuracy, suggesting that key mechanisms may remain undetected. We applied SAEs to CS4/5 to explore shared structure, but did not apply EAP due to its computational overhead and diminishing returns in deeper tasks, which require more targeted and scalable methods.

Ethical Considerations

This research primarily impacts the technical robustness of SQL understanding models. Our datasets should lead to more reliable database query systems, benefiting software developers and database administrators. As our work focuses on technical SQL syntax understanding using synthetic data, it presents minimal risks of societal harm or misuse.

References

Zeyuan Allen-Zhu and Yuanzhi Li. 2023. Physics of language models: Part 1, learning hierarchical language structures. *arXiv preprint arXiv:2305.13673*.

b mc2. 2023. sql-create-context dataset. This dataset was created by modifying data from the following sources: (Zhong et al., 2017; Yu et al., 2019b).

Nora Belrose. 2024. Sparsify. https://github.com/EleutherAI/sparsify. Accessed: 2025-02-11.

Cheng-Han Chiang and Hung-yi Lee. 2022. On the transferability of pre-trained language models: A study from artificial datasets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10518–10525.

Arthur Conmy, Augustine Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. 2023. Towards automated circuit discovery for mechanistic interpretability. *Advances in Neural Information Processing Systems*, 36:16318–16352.

Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2020. Structure-grounded pretraining for text-to-sql. *arXiv preprint arXiv:2010.12773*.

Ronen Eldan and Yuanzhi Li. 2023. Tinystories: How small can language models be and still speak coherent english? *Preprint*, arXiv:2305.07759.

Rudolf Franz Flesch and Alan J Gould. 1949. The art of readable writing. *(No Title)*.

Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R Woodward, Jinxia Xie, and Pengsheng Huang. 2021. Towards robustness of text-to-sql models against synonym substitution. *arXiv preprint arXiv:2106.01065*.

Leo Gao, Tom Dupré la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. 2024. Scaling and evaluating sparse autoencoders. *arXiv preprint arXiv:2406.04093*.

Aaron et al. Grattafiori. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783. Full author list available at https://arxiv.org/abs/2407.21783.

Robert Huben, Hoagy Cunningham, Logan Riggs Smith, Aidan Ewart, and Lee Sharkey. 2023. Sparse autoencoders find highly interpretable features in language models. In *The Twelfth International Conference on Learning Representations*.

Victoria Johansson. 2008. Lexical diversity and lexical density in speech and writing: A developmental perspective. *Working papers/Lund University, Department of Linguistics and Phonetics*, 53:61–79.

Samuel Marks, Can Rager, Eric J Michaud, Yonatan Belinkov, David Bau, and Aaron Mueller. 2024. Sparse feature circuits: Discovering and editing interpretable causal graphs in language models. *arXiv preprint arXiv:2403.19647*.

Yev Meyer, Marjan Emadi, Dhruv Nathawani, Lipika Ramaswamy, Kendrick Boyd, Maarten Van Segbroeck, Matthew Grossman, Piotr Mlocek, and Drew Newberry. 2024. Synthetic-Text-To-SQL: A synthetic dataset for training language models to generate sql queries from natural language prompts.

Anna Mitsopoulou and Georgia Koutrika. 2025. Analysis of text-to-sql benchmarks: Limitations, challenges and opportunities.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. 2023a. Progress measures for grokking via mechanistic interpretability. In *The Eleventh International Conference on Learning Representations*.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. 2023b. Progress measures for grokking via mechanistic interpretability. *arXiv preprint arXiv:2301.05217*.

Nostalgebraist. 2020. Interpreting gpt: The logit lens. https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens. Accessed: 2025-05-19.

Xinyu Pi, Bing Wang, Yan Gao, Jiaqi Guo, Zhoujun Li, and Jian-Guang Lou. 2022. Towards robustness of text-to-sql models against natural and realistic adversarial table perturbation. *arXiv preprint arXiv:2212.09994*.

Philip Quirke and Fazl Barez. 2024. Understanding addition in transformers. In *The Twelfth International Conference on Learning Representations*.

John Read. 2000. *Assessing Vocabulary*. Cambridge University Press.

Jett Janiak Stefaan Hex. 2023. A circuit for python docstrings in a 4-layer attention-only transformer.

Aaquib Syed, Can Rager, and Arthur Conmy. 2023. Attribution patching outperforms automated circuit discovery. In *Advances in Neural Information Processing Systems*.

Qwen Team. 2024. Qwen2.5: A party of foundation models.

Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, Hoagy Cunningham, Nicholas L Turner, Callum McDougall, Monte MacDiarmid, C. Daniel Freeman, Theodore R. Sumers, Edward Rees, Joshua Batson, Adam Jermyn, Shan Carter, Chris Olah, and Tom Henighan. 2024. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. *Transformer Circuits Thread*.

Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. 2023. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. In *The Eleventh International Conference on Learning Representations*.

Ping Wang, Tian Shi, and Chandan K Reddy. 2020. Text-to-sql generation for question answering on electronic medical records. In *Proceedings of The Web Conference 2020*, pages 350–361.

Zhengxuan Wu, Atticus Geiger, Thomas Icard, Christopher Potts, and Noah D. Goodman. 2024. Interpretability at scale: Identifying causal mechanisms in alpaca. *Preprint*, arXiv:2305.08809.

Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, et al. 2019a. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1962–1979.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2019b. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *Preprint*, arXiv:1809.08887.

Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Heyang Er, Irene Li, Bo Pang, Tao Chen, et al. 2019c. Sparc: Cross-domain semantic parsing in context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4511–4523.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.

A Note on AI assistance

AI assistance was used for code development and improving the phrasing of the manuscript, while all analyses and conclusions were independently derived by the authors.

B Working Note on Over-training

An LLM trained *from scratch* on our CSn training data should predict SQL very accurately as it can assume the answer is always "SELECT some fields FROM some table ORDER BY some fields WHERE some clauses". To investigate whether we have over-trained the LLM to be SQL specific, we measure the model performance, before and after refinement training, on general tasks. In the TinyStories case, we use story telling tasks and evaluations as per the original paper, as shown in Table6

	BM1		BM2		BM3	
	SQL	Story	SQL	Gen	SQL	Gen
Base	2%	95%	24%	92%	24%	91%
CS1	93%	92%	92%	87%	90%	88%
CS2	83%	88%	81%	82%	89%	86%
CS3	63%	85%	75%	79%	88%	84%

Table 6: The base (unrefined) models show limited ability to perform text-to-SQL tasks (columns "SQL"). For the command sets CS1, CS2 and CS3, the refined models, especially the LLMs, perform much better, while retaining most of their general capability (columns "Story" and "Gen").

C Dataset details

C.1 Dataset Generation Methodology: CSx, CSx_Syn

We developed an automated pipeline to generate SQL queries with increasing complexity while maintaining consistent patterns and structures for systematic progression and controlled experimentation.

Each dataset variant contains 100,000 examples, split into training (76.5%), validation (13.5%), and test (10%) sets. While patterns are consistent across splits, we ensure no direct overlap of specific examples.

Each example consists of three components, the natural language instruction, the table schema context, and the target SQL query and is generated as follows:

Step 1: Schema Generation. Each example features a single table with a randomly chosen name from a set of 200 common options (e.g., employees, products). Columns are drawn from a larger pool of 300 database fields (e.g., user_id, timestamp), paired with appropriate SQL types (e.g., `INTEGER`, `VARCHAR`). Tables contain 2-12 columns, with certain fields restricted to relevant tables (e.g., salary only in employment-related tables) for semantic coherence.

Step 2: Target Query Generation. Based on command set level, we generate SQL queries with increasing complexity. CS1 creates basic `SELECT-FROM` statements, selecting random schema columns. CS2 adds `ORDER BY` clauses (90% probability), randomly choosing ascending or descending order. CS3 introduces aggregate functions (`COUNT`, `SUM`, `AVG`, `MIN`, `MAX`), applied to numeric columns with an 85% probability, ensuring compatibility with data types (e.g., `SUM` and `AVG` for numeric fields). CS4 adds `WHERE` clauses (80% probability) with 1–3 type-consistent filters per query. CS5 introduces a second table with a `JOIN` clause on a shared field when schema compatibility allows.

Step 3: Instruction Generation. We translate SQL queries into natural language using a set of 50 templated patterns. These patterns range from

direct commands ("Show me X from Y") to questions ("What are the X in Y?") and complex constructions ("From Y, get me X ordered by Z"). For base variants (CS1-3), we use exact table and column names. For synonym variants (CSx_Syn), we replace 80% of table names and 50% of column names with synonyms. The templates maintain consistent structure while varying surface form. Aggregate functions have their own set of 20 dedicated phrase patterns (e.g., "average of X" → "`AVG(X)`").

Step 4: Quality Checks. We implement an automated scoring system that evaluates generated SQL along multiple dimensions. It considers structural correctness (proper clause placement), semantic validity (field name matching), and implementation accuracy (correct aggregation and ordering). The system assigns partial credit based on component-wise correctness, allowing for fine-grained evaluation of model outputs.

C.2 Dataset Generation Methodology: CSx_Nat

We additionally explore a synthetic dataset generation method using Gretel AI's Data Designer. By defining two *categorical seed columns* (`table_name` and `instruction_phrase`) and several *generation columns* (e.g., `column_names`, `selected_columns`), we prompt Gretel's language models to produce varied table names, column sets, and natural instructions. We then assemble them into structured SQL queries, applying post-processing rules for syntactic correctness. This approach further diversifies our text-to-SQL dataset by introducing additional schema and instruction variety.

We use Gretel AI's Data Designer, which employs a compound AI pipeline controlled by a YAML specification, to create a synthetic dataset we call **CS1_Nat**. Our configuration includes:

- **Categorical seed columns:** We define two main seed columns, `table_name` and `instruction_phrase`, each populated with a diverse set of possible values (e.g., 100 table names, 60 instruction patterns). These serve as anchors, ensuring rich contextual variety in the generated rows.
- **Generated data columns:** We specify prompts for each generated column:
  - `column_names`: Produces domain-relevant, snake_case field names.
  - `selected_columns`: Selects a subset of those field names for the final SQL query.
  - `column_data_types`: Pairs each chosen field name with an appropriate SQL type (e.g., `INT`, `VARCHAR`).
  - `sql_prompt`: Reformulates table and column references into more natural instruction phrases, injecting synonym variety.
  - `sql_context`: Composes a `CREATE TABLE` statement by combining field names and data types into a cohesive schema definition.
  - `sql`: Produces a full `SELECT` query matching the generated schema and context.
- **Post-processors:** We apply validation checks to confirm syntactic correctness and logical coherence, ensuring the generated `SELECT` query aligns with the declared table schema and the instruction phrase.

By adjusting prompts and carefully selecting seed column values, we ensure each generated instruction and corresponding SQL query remains unique and contextually consistent. The resulting dataset is then partitioned into training, validation, and test splits for further use in text-to-SQL modeling and MI studies. All configuration details and additional examples are available upon request, enabling reproducibility and further exploration of prompt-based synthetic data generation.

Below is a simplified sample of the configuration file used to generate CS1-Nat using Data Designer:

```
model_suite: apache-2.0

special_system_instructions: >-
  You are a SQL expert. Your role is
  to write SQL prompts, SELECT,
  and CREATE TABLE statements.

categorical_seed_columns:
  - name: table_name
    values:
      - users
      - user_profiles
      - ...
  - name: instruction_phrase
    values:
```

https://gretel.ai/navigator/data-designer
https://gretel.ai/navigator/data-designer

```
      - ``Construct an SQL query to''
      - ``Retrieve the''
      - ...

generated_data_columns:
  - name: sql_prompt
    generation_prompt: >-
      Generate a concise natural
      language instruction for ..
  - name: sql_context
    generation_prompt: >-
      Generate the SQL CREATE TABLE
      statement for a table
      named '{table_name}'...
  - name: sql
    generation_prompt: >-
      Generate a SQL statement ...

post_processors:
  - validator: code
```

This systematic approach ensures reproducible dataset generation while maintaining controlled progression in task complexity. The explicit probabilities and controlled vocabularies enable consistent generation across different implementations.

C.3 Dataset Examples

Table 7 shows representative examples from each category (CS1–CS5) in the TinySQL dataset. Each example includes a natural language question, a SQL statement and the corresponding SQL query.

D Trained Models

Base Model Licenses. The base model for BM1, `TinyStories-33M` (Eldan and Li, 2023),is released under the MIT license. The base model for BM2, `Qwen2.5-0.5B-Instruct` (Team, 2024), is released under the Apache 2.0 license. The base model for BM3, `Llama-3.2-1B-Instruct` (Grattafiori, 2024), is released under the Llama 3.2 Community License.

Model Architectures. The three models vary in size and architecture. BM1 (`TinyStories-33M`) has 2 transformer layers and 16 attention heads. BM2 (`Qwen2.5-0.5B-Instruct`) is composed of 24 layers with 14 attention heads. BM3 (`Llama-3.2-1B-Instruct`) consists of 16 layers and 32 attention heads. These configurations reflect each model's respective scale and capacity.

Training Details. We use the `transformers` library and the `trl` extension for instruction fine-tuning using the standard alpaca template. Models are trained using an *AdamW* optimizer with weight decay of 0.01. The learning rate used is 1×10^{-5}, a value chosen based on preliminary experiments showing that higher rates led to larger validation oscillations and less stable convergence. We employ 100 warmup steps at the beginning of training. Each model is trained for a single epoch over `76,500` training examples, with `13,500` validation and `10,000` test examples in each dataset split. We use a maximum sequence length of 512 tokens. The effective batch size is 32 per update step, achieved by running on four GPUs with a per-device batch size of 8 and a gradient accumulation step of 1. For BM2 and BM3, we enable `flash-attention v2` for efficient attention computation and reduced memory overhead. The padding side is configured based on each model's tokenizer requirements, and new tokens (e.g. a dedicated <pad> token) are added or resized if needed to accommodate smaller or older base models (e.g. TinyStories). Training took about 1 hour / epoch for TinyStories and about 2 hours / epoch on Llama / Qwen models on 2 x A100 GPUs.

We trained basic and semantic models as shown in Table 8.

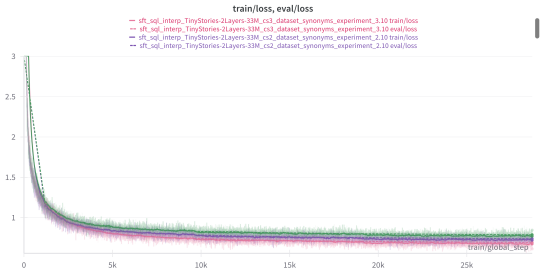


Figure 4: Training and Validation loss curves for instruction tuning BM1 (TinyStories-Instruct-2Layers-33M) on CS1_Syn, CS2_Syn, CS3_Syn

E Data set entry characteristics

For brevity, the main text of this paper uses short examples. The dataset entries are considerably longer. These are some characteristics of the generated datasets:

- All CREATE TABLE clause contains (randomly) 2 to 12 columns.
- All SELECT clause contains (randomly) 1 to the number of table columns.

Dataset	Create Statement	English Prompt	SQL Statement
CS1	CREATE TABLE surveys (file_type VARCHAR(100), expires_at DATETIME, comment TEXT, message TEXT, birthday DATE)	Get a readout of expires_at, birthday, comment, message and file_type from surveys	SELECT expires_at, birthday, comment, message, file_type FROM surveys
CS1-Syn	CREATE TABLE search_filters (downloads BIGINT, passed BOOLEAN, website VARCHAR(255), discount DECIMAL(10,2), birthday DATE, is_active BOOLEAN, sequence SMALLINT, last_message TEXT, meta_keywords VARCHAR(500), response_id INTEGER, reference TEXT)	Can you get me recent message, date of birth, reply id, meta_keywords, url, downloads and referral from refinement options?	SELECT last_message, birthday, response_id, meta_keywords, website, downloads, reference FROM search_filters
CS2	CREATE TABLE terms (vote_id INTEGER, start_date DATE, mac_address VARCHAR(17), last_message TEXT, signature BLOB, middle_name VARCHAR(100), coordinates POINT, address VARCHAR(255), meta_description TEXT)	Can you get me coordinates, signature, address, start_date and mac_address from terms? top coordinates, best meta_description, with the highest start_date, starting with the highest last_message, starting with the highest mac_address	SELECT coordinates, signature, address, start_date, mac_address FROM terms ORDER BY coordinates DESC, meta_description DESC, start_date DESC, last_message DESC, mac_address DESC
CS2-Syn	CREATE TABLE configurations (subject TEXT, image TEXT, response TEXT, major VARCHAR(100), company VARCHAR(255), session_id VARCHAR(100), city VARCHAR(100), min_score INTEGER, actual_time INTEGER, manufacturer VARCHAR(255))	Looking in setup details, show me major, real duration, title, city, maker, session_id, photo, minimum points and organization worst company	SELECT major, actual_time, subject, city, manufacturer, session_id, image, min_score, company FROM configurations ORDER BY company ASC
CS3	CREATE TABLE conversions (total_price DECIMAL(10,2), approved_at TIMESTAMP, content LONGTEXT, estimated_time SMALLINT, permissions TEXT, first_name VARCHAR(100), salt VARCHAR(32), full_name VARCHAR(150))	Read out total count total_price, how many full_name, times approved_at, tally salt, content, instances of permissions, first_name and how many estimated_time from conversions arranged by first_name, oldest permissions, structured by estimated_time	SELECT COUNT(total_price) AS COUNT_total_price, COUNT(full_name) AS COUNT_full_name, COUNT(approved_at) AS COUNT_approved_at, COUNT(salt) AS COUNT_salt, content, COUNT(permissions) AS COUNT_permissions, first_name, COUNT(estimated_time) AS COUNT_estimated_time FROM conversions ORDER BY first_name ASC, permissions ASC, estimated_time ASC
CS3-Syn	CREATE TABLE product_versions (skills TEXT, location POINT)	Can you get me latest skills and location from releases? sorted alphabetically by location	SELECT MAX(skills) AS MAX_skills, location FROM product_versions ORDER BY location ASC
CS4	CREATE TABLE data (max_score INTEGER, gender CHAR(1), region GEOMETRY, location GEOMETRY, middle_name VARCHAR(100), salt VARCHAR(32), last_message TEXT)	Bring up middle_name, last_message, location, salt, region, max_score and gender from information where salt is containing '%z%' priority ordered by max_score, in reverse alphabetical order of location	SELECT middle_name, last_message, location, salt, region, max_score, gender FROM data WHERE salt LIKE '%z%' ORDER BY max_score DESC, location DESC
CS5	CREATE TABLE workflow_templates (like_id INTEGER, heading DECIMAL(5,2), is_verified BOOLEAN, share_id BIGINT, average_score FLOAT, rating DECIMAL(3,2), time TIME, certification VARCHAR(255), name VARCHAR(100), city VARCHAR(100))	I need a list of verified status, like_id, time, average_score, designation, city, heading, rating, certification and share_id from procedure patterns join with pairs on like_id equals user_id ordered numerically by average_score, ordered by date of is_verified, alphabetically by certification, grouped by rating	SELECT is_verified, like_id, time, average_score, name, city, heading, rating, certification, share_id FROM workflow_templates JOIN pairs ON workflow_templates.like_id = pairs.user_id ORDER BY average_score ASC, is_verified ASC, certification ASC, rating ASC

Table 7: Examples from TinySQL dataset across five compositional subsets (CS1–CS5) and CS1-Syn, CS2-Syn, CS3-Syn data.

Abbrev.	Model	CSn	Type	Exact-match Accuracy
BM1_CS0	TinyStories	N/A	N/A	0%**
BM2_CS0	Qwen 2.5	N/A	N/A	10%**
BM3_CS0	Llama 3.2	N/A	N/A	10%**
BM1_CS1_1.8	TinyStories	CS1	Basic	98.79%
BM1_CS1_1.10	TinyStories	CS1_Syn	Semantic	92.97%
BM1_CS2_2.8	TinyStories	CS2	Basic	97.62%
BM1_CS2_2.10	TinyStories	CS2_Syn	Semantic	92.02%
BM1_CS3_3.8	TinyStories	CS3	Basic	94.31%
BM1_CS3_3.10	TinyStories	CS3_Syn	Semantic	85.63%
BM2_CS1_4.2	Qwen 2.5	CS1	Basic	100.0%
BM2_CS1_4.3	Qwen 2.5	CS1_Syn	Semantic	99.52%
BM2_CS2_5.2	Qwen 2.5	CS2	Basic	100.0%
BM2_CS2_5.3	Qwen 2.5	CS2_Syn	Semantic	99.55%
BM2_CS3_6.2	Qwen 2.5	CS3	Basic	99.82%
BM2_CS3_6.3	Qwen 2.5	CS3_Syn	Semantic	98.88%
BM3_CS1_7.2	Llama 3.2	CS1	Basic	100.0%
BM3_CS1_7.3	Llama 3.2	CS1_Syn	Semantic	99.67%
BM3_CS2_8.2	Llama 3.2	CS2	Basic	100.0%
BM3_CS2_8.3	Llama 3.2	CS2_Syn	Semantic	99.63%
BM3_CS3_9.2	Llama 3.2	CS3	Basic	99.94%
BM3_CS3_9.3	Llama 3.2	CS3_Syn	Semantic	99.43%

Table 8: The base models (TinyStories-Instruct-2Layers-33M, Qwen2.5-0.5B-Instruct and Llama-3.2-1B-Instruct show limited ability to perform text to SQL tasks. The trained models, especially the LLMs, perform much better, while retaining most of their general capability. ** The evaluation is strict as it requires the model provide just the answer without any additional preface or text, explaining the low zero-shot scores here for Llama and Qwen

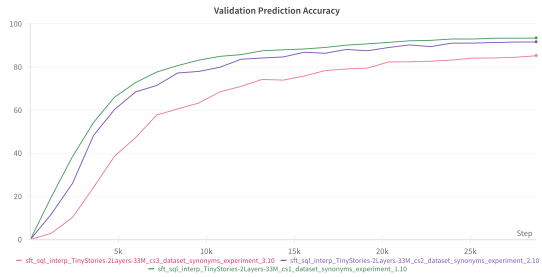


Figure 5: Accuracy curves for instruction tuning BM1 (TinyStories-Instruct-2Layers-33M) on CS1_Syn, CS2_Syn, CS3_Syn

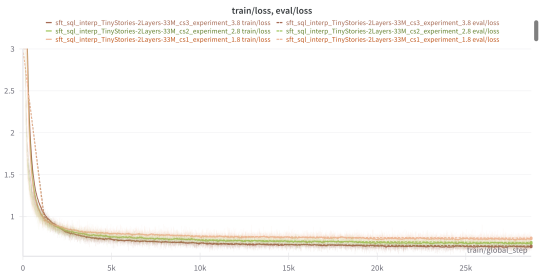


Figure 6: Training and Validation loss curves for instruction tuning BM1 (TinyStories-Instruct-2Layers-33M) on CS1, CS2, CS3

- In data set CS2, the ORDER BY clause is added and contains (randomly) 1 to the number of table columns. The ASC or DESC ordering is chosen randomly. An ORDER BY clause is added to 90% of entries.

- In data set CS3, each SELECT column is randomly assigned a valid (depending on the column data type) aggregate from the list: SUM, AVG, MIN, MAX, COUNT and "". That is, ~20% of columns have no aggregate.

- In data set CS4, we extend CS3 by adding a type-aware WHERE clause to 80% of queries, selecting 1–3 distinct columns and type appropriate operators/values.

- In data set CS5, we introduce a second table and a JOIN clause that matches columns with the same base type. If there are no columns in the second table that share a common type with the table already in the statement, we do not add a JOIN clause. This results in 73.0%

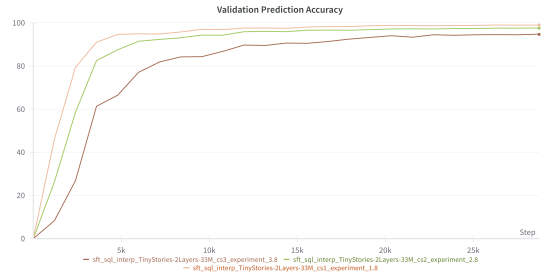


Figure 7: Accuracy curves for instruction tuning BM1 (TinyStories-Instruct-2Layers-33M) on CS1, CS2, CS3

of the train split for CS5 containing a JOIN.

Table 9 shows the maximum length of entries in each dataset.

F Activation Patching Results

Across prompts across all classes, we find that the activation patching results differ quite significantly from each other, even for prompts in the same class. We provide a sample of 4 examples in Figure 8, and we note that this variation persists throughout all examples.

G EAP Circuits Identification

G.1 Edge Attribution Patching (EAP) Results

We run the Edge Attribution Patching (EAP) experiment as described in Sec. 4.3 on a set of clean and corrupted prompt pairs. For each model, we determine which features to corrupt based on the complexity of the command set being evaluated. Typically, we corrupt one token at a time, then input the clean and corrupted prompt pairs into the EAP algorithm to identify the set of important edges for each feature.

To ensure reliability, we run EAP on 15 batches of 100 prompts each, recording only the edges that appear across all batches above a chosen threshold.

G.1.1 Features used per dataset

For the CS1 command set, where SQL statements follow this structure:

Example from CS1

Instruction: show me the type and date from the orders table ### Context: CREATE TABLE orders (type CHAR, date INT) ### Response: SELECT type, date FROM orders

We identify the following key features for correct response prediction:

- **EngTableName**: Table name in the ### Instruction
- **EngFieldName**: Field name in the ### Instruction
- **DefTableName**: Table name in the ### Context
- **DefFieldName**: Field name in the ### Context

For each feature, we corrupt prompts by replacing only the feature with a random word while keeping everything else unchanged. We then run EAP and record the results.

For the CS2 command set, where SQL statements follow this structure:

Example from CS2

Instruction: show me the category and value from the links table ordered by value in ascending order ### Context: CREATE TABLE links (category CHAR, value TEXT) ### Response: SELECT category, value FROM links ORDER BY value ASC

We use the same features as in CS1 and add:

- **OrderByField**: Field used for ordering in the ### Instruction (*value* in the example)
- **OrderByDirection**: Ordering direction in the ### Instruction (*ascending* in the example)

For the CS3 command set, where SQL statements follow this structure:

Example from CS3

Instruction: From orders, get me least recent code with the highest code ### Context: CREATE TABLE orders (weight CHAR, code TEXT) ### Response: SELECT MIN(code) AS MIN_code FROM orders ORDER BY code DESC

In addition to the CS2 features, we introduce:

- **AggregateField**: Field used for aggregation in the ### instruction (code in the example)
- **AggregateFunction**: Term indicating the aggregation function in the ### Instruction (highest in the example)

Dataset	English Prompt	Create statement	SQL statement
CS1	19 words, 169 chars	29 words, 299 chars	15 words, 159 chars
CS2	63 words, 476 chars	29 words, 308 chars	40 words, 325 chars
CS3	78 words, 579 chars	29 words, 296 chars	55 words, 564 chars

Table 9: Lengths of the longest entries in words and characters per dataset. For the synonym datasets, the English Prompts are ~10% longer.

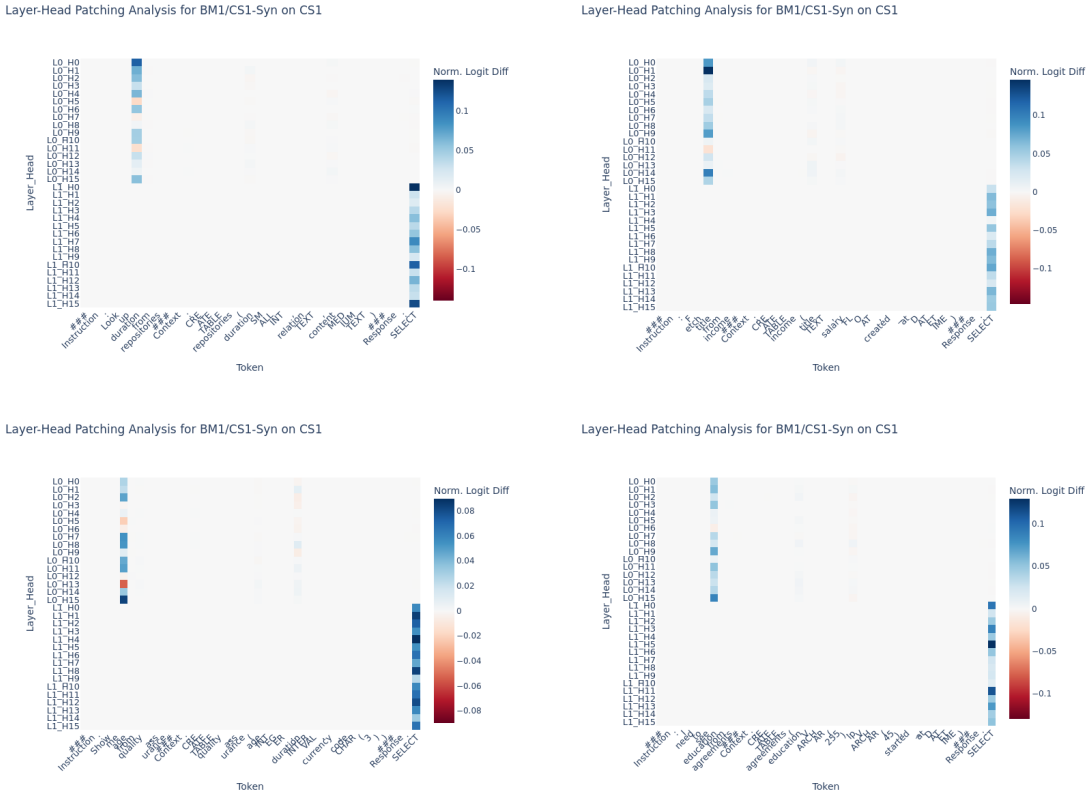


Figure 8: Activation patching results for 4 prompts, where the table contains three fields and the user queries for only the first field of the table. Even when controlling for some variables, the results differ from prompt to prompt.

G.1.2 EAP results

Once we generate our set of batches for each feature, we run the EAP algorithm and record the most important edges.

For each run, we select the top 10 edges per batch and retain only the edges that appeared at least T% of the time across batches. For CS1 and CS2, we set T = 100, while for CS3, we set T = 80.

For each feature, we run EAP four times, covering all possible truth-value combinations of use_synonyms_field and use_synonyms_table. This allows us to identify the set of edges responsible for each feature in both semantic and non-semantic cases.

After running EAP for all features in both cases, we obtain a set of edges per feature. We then aggregate these edges, extract their corresponding nodes, and use them later in the Ablation Study. Results of EAP are shown in Fig. 14 for CS1, Fig. 15 for CS2 and Fig. 16 for CS3. We only show results where use_synonyms_field and use_synonyms_table are both set to False.

G.2 Ablation Study Results

After selecting the set of attention head nodes using EAP, we ablate all attention heads that are not part of this set. We apply both zero ablation and mean ablation, then compare the results, as shown in Fig. 17. Our conclusions are presented in Sec. 4.3.

G.3 Multi-Layer Perceptron (MLP) Ablations

The next step was to keep the attention heads intact and instead ablate MLP outputs to determine whether mean ablation produces the same results and whether the model maintains its performance when using average activations instead of actual output activations. To compute the average activations, we use a batch of 200 examples.

Our conclusions were presented in Sec. 4.3. However, the figures in Fig. 18 show that the first layer plays a major role in all models. Additionally, when a complex dataset is used (CS3), both layers become important, and mean ablation no longer preserves performance. This trend is consistently observed across all cases of MLP ablations.

G.4 Full-Layer Ablations

In the same manner, we now ablate all the outputs of a layer. Once again, we observe as shown in Fig. 19 that layer 1 plays a more significant role in generation. However, this trend shifts with more complex command sets. When ablating entire layers, the model loses performance under both mean ablation and zero ablation across all layers.

For larger models like Qwen, accuracy drops to 20% regardless of the number of layers ablated.

Model	Dataset	Layer 0	Layer 1
BM1_CS1_Syn	CS1	[1, 8, 10, 7]	[13, 4, 2, 1]
BM1_CS1_Syn	CS1_Syn	[1, 8, 10, 7]	[13, 4, 2, 8]
BM1_CS2_Syn	CS2	[1, 10, 8, 14]	[13, 8, 4, 1]
BM1_CS2_Syn	CS2_Syn	[1, 10, 8, 14]	[13, 8, 4, 1]
BM1_CS3_Syn	CS3	[1, 10, 8, 14]	[6, 2, 1, 9]
BM1_CS3_Syn	CS3_Syn	[1, 10, 8, 14]	[6, 2, 1, 9]

Table 10: From SAE feature analysis, important attention heads for processing table name tokens.

Model	Dataset	Layer 0	Layer 1
BM1_CS1_Syn	CS1	[1, 10, 8, 7]	[13, 4, 1, 12]
BM1_CS1_Syn	CS1_Syn	[1, 10, 8, 7]	[13, 4, 12, 1]
BM1_CS2_Syn	CS2	[1, 10, 8, 14]	[13, 8, 4, 1]
BM1_CS2_Syn	CS2_Syn	[1, 10, 8, 14]	[13, 8, 4, 1]
BM1_CS3_Syn	CS3	[1, 10, 8, 14]	[6, 2, 1, 13]
BM1_CS3_Syn	CS3_Syn	[1, 10, 8 , 14]	[6, 2, 1, 9]

Table 11: From SAE feature analysis, important attention heads for generating table name tokens.

H SAE analysis

We show the important heads in BM1 for processing table tokens next in Table 10. We also consider the important SAE features for generating the field tokens used for "ORDER BY" or "GROUP BY" type queries in Table 12.

Note that for response table names, Head 1 in Layer 0 is consistently the most important. Whereas we see in Layer 1, that BM1_CS3_* models consistently use Head 6 as opposed to BM1 and BM2 which use Layer 13.

We next look at important heads for generating table names, as determined by being immediately before the response table name, in Table 11.

Model	Dataset	Layer 0	Layer 1
BM1_CS2_Syn	CS2	[1, 10, 8, 14]	[13, 8, 1, 4]
BM1_CS2_Syn	CS2_Syn	[1, 10, 8, 14]	[13, 8, 4, 1]
BM1_CS3_Syn	CS3	[1, 10, 8, 14]	[6 ,2 ,1 , 13]
BM1_CS3_Syn	CS3_Syn	[1, 10, 8, 14]	[6, 2, 1, 13]

Table 12: From SAE feature analysis, important attention heads for generating order by and sort by tokens accurately.

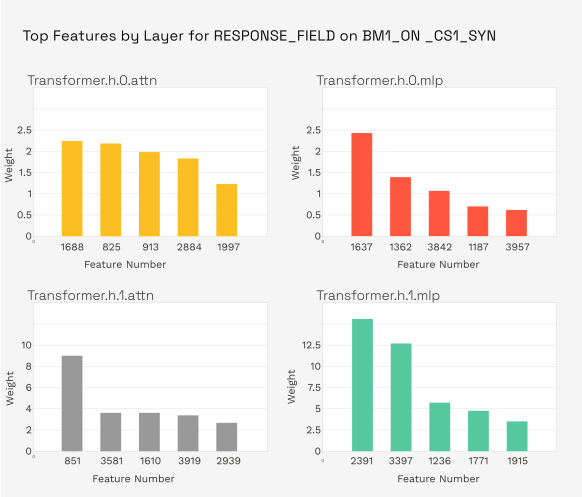


Figure 9: Average SAE activations for BM1_CS1_Syn on CS1_Syn table fields.

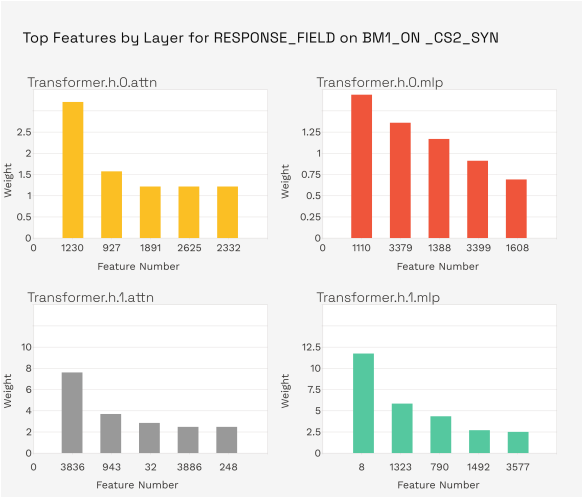


Figure 10: Average SAE activations for BM1_CS2_Syn on CS2_Syn table fields.

The high degree of overlap between the heads used for processing the table names, and the field tokens, suggests that either these heads are polysemantic, or that the model uses a similar mechanism for selecting parts of the prompt to be used in the final response.

We show now the SAE features most involved in filling out the table name tokens, in particular for BM1_CS1_Syn, BM1_CS2_Syn and BM1_CS3_Syn. See Figures 9, 10 and 11 for the top activating SAE features by average magnitude.

We can map these to the actual attention outputs, see for instance Figures 12a, 12b and 12c. We average over these to determine the influential attention heads.

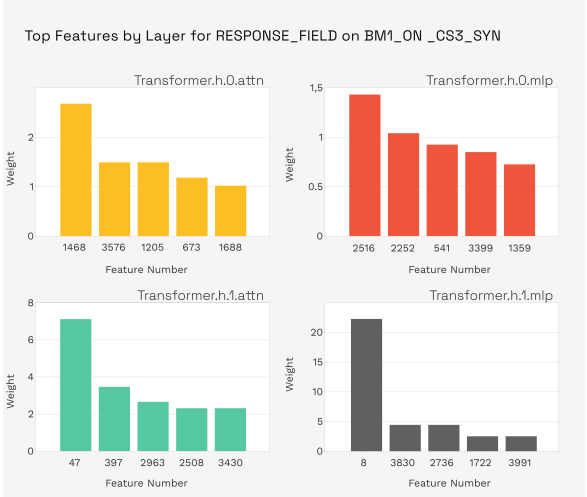


Figure 11: Average SAE activations for BM1_CS3_Syn on CS3_Syn table fields.

I SAE Circuits Results

We present additional results from our Synthetic Activation Experiments (SAEs) highlighting the identified circuit components and their recovered accuracy after zero-ablation across different model and dataset configurations (Table 13).

tabularx

Table.14 details the layer-wise circuits identified by Synthetic Activation Experiments (SAEs) for the BM2_CS1_Syn model, demonstrating more granular selection of individual attention heads compared to the broader patterns found by EAP.

J Mean-Ablation Circuits

Circuits in Small 2-Layer Models. We begin by analyzing models with only two transformer layers (e.g., BM1 and BM3 variants). The mean ablation results for these models reveal small, interpretable sets of attention heads responsible for accurate task performance. Table 15 summarizes the critical heads involved in each model-task pair and their corresponding recovered accuracy after ablation.

Circuits in a Deeper Model: BM2-CS1. To investigate whether similar sparse circuits exist in deeper models, we perform a layer-wise mean ablation analysis on BM2-CS1, which has 24 layers. For each layer, we identify key heads contributing to recovery and report both the area under the curve (AUC) and the recovered accuracy. Table 16 presents these results, showing how different layers contribute to robust model behavior via distributed

Model	Dataset	Heads and MLPs		Recovered Accuracy
		Layer 0	Layer 1	
BM1_CS3_Syn	CS2	13, 1, 8, 15, 10, 12, 14, 3, 7 All MLP neurons	– All MLP neurons	81.41%
BM1_CS4_Syn	CS4	8, 1, 10, 13, 3, 6, 12, 9, 2, 15 750 MLP neurons	– –	82.19%
BM1_CS5_Syn	CS5	8, 10, 1, 13, 3, 12, 6, 9, 2, 7, 4 All MLP neurons	– –	70.00%

Table 13: Additional SAE-based circuit results for BM1-CS3→CS2, CS4, and CS5 configurations. Recovered accuracy reflects performance after zero-ablation of all model components not selected by high-AUC SAE features.

yet sparse mechanisms.

Efficiency of Model Component Usage. Beyond identifying which layers and heads are important, we quantify how much of the model is actually needed for successful task completion. Table 17 reports the percentage of total components (e.g., attention heads or MLP neurons) that were retained in each ablation configuration while still achieving high performance. These results indicate that high accuracy can often be recovered using only a small fraction of the model's total capacity.

K Comparison of EAP-Selected Circuits and Random Circuits

Table 18 presents a comparison between the recovered accuracies of circuits selected by Edge Attribution Patching (EAP) and the average accuracies of randomly selected circuits of comparable size across different model and dataset configurations.

L Logit Lens Analysis Case Study

We performed a comprehensive logit lens analysis on three representative examples from CS2 and CS3 using the BM2 model (Qwen 0.5B, 24 layers) to better understand how SQL generation emerges across transformer layers.

Progressive Token Emergence Pattern: SQL keywords (e.g., `SELECT`, `FROM`, `WHERE`) achieve high confidence (top ranks) in earlier layers (17–21). In contrast, table and column names (e.g., `orders`, `order_id`, `links`) only emerge with high confidence in the final layer (24). *Example:* In Table 19, across all cases, `FROM` tokens reach rank 1–2 by layers 21–22, while table names like `orders` and `links` remain low until jumping to rank 1 at layer 24.

Logit Trajectory Analysis: A systematic reversal in relative logit strengths is observed. Early layers (0–16): SQL keywords maintain higher logits than table/column names. Later layers: Table/column names surpass SQL keywords in logit values. *Example:* In Example 1, `SELECT` has logits around 0.4–0.5 through layers 8–9, while `orders` starts near 0.05 and only reaches comparable logits (> 0.3) in layer 24.

Token-Type Stratification: Three distinct emergence phases were found:

- Basic SQL tokens (`SELECT`, `FROM`) stabilize by layers 17–21
- Functional tokens (`COUNT`, `ORDER BY`) emerge in mid-layers
- Context-specific identifiers (table/column names) resolve in the final layer

This supports a hierarchical decoding pattern: from general syntax to specific schema grounding.

Cross-Example Consistency: The same patterns hold across diverse SQL operations: `COUNT` aggregation (Example 1), `MIN` functions (Example 2), and `ORDER BY` clauses (Example 3). Schema-specific tokens always emerge last (layer 24), while SQL structure stabilizes earlier.

Examples Analyzed

- **Example 1**
  Instruction: In `orders`, list total count amount and instances of duration
  Context: `CREATE TABLE orders (duration TIME, amount INT)`
  Response: `SELECT COUNT(amount) AS COUNT_amount, COUNT(duration) AS COUNT_duration FROM orders`
- **Example 2**
  Instruction: Search for least duration in `links` beginning with the most duration

Model	Layer	Selected Heads	Recovered Accuracy
BM2_CS1_Syn	L0	12, 7, 6, 0, 10, 2, 1, 9, 4	88.1%
	L1	All Heads	
	L2	0, 11, 4, 5, 9, 8, 13, 1, 2	
	L3	0, 11, 10, 12, 3, 1, 9, 4	
	L4	All Heads	
	L5	All Heads	
	L6	5, 0, 2, 13, 6, 8, 4, 12, 7, 9	
	L7	9, 1, 6, 0, 2, 4, 11, 8, 3	
	L8	All Heads	
	L9	0, 3, 8, 4, 13, 9, 7, 10, 1	
	L10	All Heads	
	L11	All Heads	
	L12	0, 8, 7, 11, 3, 1, 5, 13	
	L13	0, 3, 11, 7, 1, 2, 4, 10, 9	
	L14	All Heads	
	L15	9, 7, 10, 1, 4, 3, 8, 13, 5, 0	
	L16	All Heads	
	L17	13, 5, 4, 10, 7, 0, 8, 11	
	L18	3, 0, 5, 13, 11, 8, 4, 9, 1, 6	
	L19	3, 0, 7, 4, 11, 10, 13, 1, 5, 12	
	L20	0, 3, 7, 2, 1, 13, 10	
	L21	0, 3, 7, 1, 10, 6, 11, 9	
	L22	0, 3, 7, 1, 2, 11, 10, 6	
	L23	12, 1, 9, 13, 5, 2, 11	

Table 14: Layer-wise SAE-selected circuits for *BM2_CS1_Syn*. Compared to EAP, the circuits identified by SAEs exhibit finer granularity, isolating individual attention heads rather than entire layers.

Model	Dataset	Important Heads (Layer0)	Recovered Accuracy	Component %
BM1-CS1	CS1	11, 3, 1, 8	93.68%	12.5%
BM1-CS3	CS1	11, 14, 3, 8, 2	93.8%	15.62%
BM1-CS3	CS2	11	99.0%	3.12%
BM1-CS3	CS3	11, 14, 3, 0, 13	73.43%	15.62%
BM3-CS1	CS1	0, 5	99.2%	8.3%

Table 15: Mean Ablation Results for Attention Circuits in BM1 and BM3 Models

Layer	Key Heads	Recovered Accuracy
0	12	98.6%
1	7	98.4%
2	0	98.2%
3	0, 11, 10, 12, 3, 1, 9, 4, 13	98.3%
4	2	98.3%
5	0	98.4%
6	5	97.9%
7	9	98.2%
8	11	98.2%
9	3	98.39%
10	0	98.2%
11	3	97.8%
12	7	98.2%
13	0	98.2%
14	0	97.9%
15	10, 7, 4, 13, 1, 9, 3, 8	–
16	0, 7, 13, 10, 3, 11, 2, 12, 1	97.9%
17	13, 0, 5, 4, 1, 3, 10	98.2%
18	3, 4, 5, 11, 0, 13, 12, 9, 6, 8	98.2%
19	3, 0, 13, 10, 7, 12, 5, 4, 1	97.9%
20	0, 3, 7, 2, 6, 4, 10, 1, 8	97.9%
21	7, 13, 1, 6, 3, 8, 10, 9, 0	98.2%
22	0, 3, 1, 6, 7, 8, 9, 12, 10	97.9%
23	12, 13, 1, 6, 9, 11, 5, 0, 2, 7	98.8%

Table 16: Mean Ablation Results Across Layers for BM2-CS1

Model	MI Method	Component Type	Percentage of Model Components
BM1-CS1-Syn	CS1	EAP	12.5%
BM1-CS3-Syn	CS1	EAP	15.62%
BM1-CS3-Syn	CS2	EAP	3.12%
BM1-CS3-Syn	CS3	EAP	15.62%
BM3-CS1-Syn	CS1	EAP	8.3%
BM2-CS1-Syn	CS1	EAP	12.5%
BM1-CS1-Syn	CS1	SAE	17.26%
BM1-CS3-Syn	CS1	SAE	22.5%
BM1-CS3-Syn	CS2	SAE	18.75%
BM1-CS3-Syn	CS3	SAE	28.97%
BM2-CS1-Syn	CS1	SAE	60.71%

Table 17: Percentage of Model Components Used in Successful Recovery. In EAP, we only consider the percentage of Attention Heads as we only had access to those.

Model	EAP Circuit Accuracy	Average Random Circuit Accuracy
BM1-CS1-Syn → CS1	85.4%	54.55%
BM1-CS3-Syn → CS1	97.6%	82.76%
BM1-CS3-Syn → CS2	74.6%	57.83%
BM1-CS3-Syn → CS3	78.3%	66.66%
BM2-CS1-Syn → CS1	71.6%	30.0%

Table 18: Recovered accuracy comparison between EAP-selected circuits and randomly selected circuits of similar size.

Context: `CREATE TABLE links (price INT, duration TIME)`
Response: `SELECT MIN(duration) AS MIN_duration FROM links`

- **Example 3**
  Instruction: Show me the duration and size from the `orders` table ordered by duration in ascending order
  Context: `CREATE TABLE orders (duration TIME, size INT)`
  Response: `SELECT duration, size FROM orders ORDER BY duration ASC`

The results are summarized in Table 19

Summary Table: Layer Where Token First Achieves Top Rank

Logit Progression Pattern

- Early Layers (0–16): SQL keywords dominate with a positive logit advantage
- Mid Layers (17–21): SQL structure tokens reach high ranks, schema tokens remain low
- Final Layer (24): Schema-specific tokens (e.g., table/column names) reach top logit values

This progression supports the hypothesis that transformer layers follow a hierarchical reasoning process, from abstract SQL syntax to grounded schema resolution.

We also present in Table 20, a detailed analysis of the SQL Keyword `COUNT` ranks across layers compared to the table name `orders` ,

M Additional LogitLens Visualizations

To further support our interpretation of two-phase SQL query generation, we include additional LogitLens visualizations across other benchmarks and model configurations.

In each case, we consistently observe the same dynamics: an early spike in the logit probabilities of SQL-specific keywords, followed by a later emergence of table name tokens, indicating a transition from syntactic intent planning to context-aware content grounding.

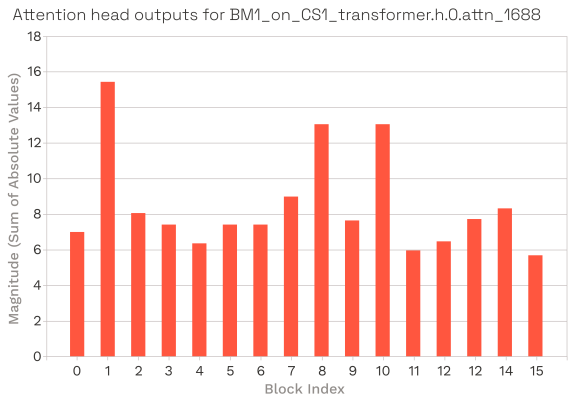
These findings generalize across schema types and query contexts, reinforcing the idea that language models internally decompose structured generation tasks into distinct stages aligned with human-interpretable reasoning processes.

Token Type	**Example 1**	**Example 2**	**Example 3**
FROM	Layer 21	Layer 21	Layer 22
Aggregation (MIN, COUNT)	Layer 17	Layer 21	N/A
AS	Layer 24	Layer 24	N/A
GROUP BY	N/A	N/A	Layer 17
Table Name (links, orders)	Layer 24	Layer 24	Layer 24
Column Name	Layer 24	Layer 24	Layer 24

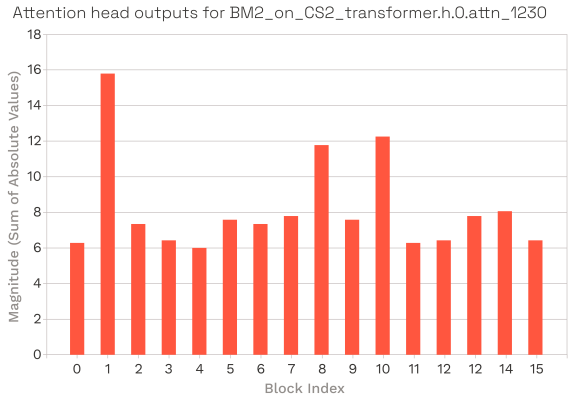
Table 19: Layer where each token type first achieves top rank.

Layer	COUNT Rank	orders Rank
1	392	43497
2	832	71069
3	1032	67813
4	5652	83330
5	7892	118378
6	4532	85341
7	2209	112749
8	3505	136323
9	2101	144139
10	2065	136692
11	4047	136514
12	2676	132739
13	384	127971
14	92	108117
15	124	76418
16	1	79962
17	1	33234
18	1	37628
19	1	25178
20	1	24058
21	1	7664
22	2	2430
23	2	166
24	1	1

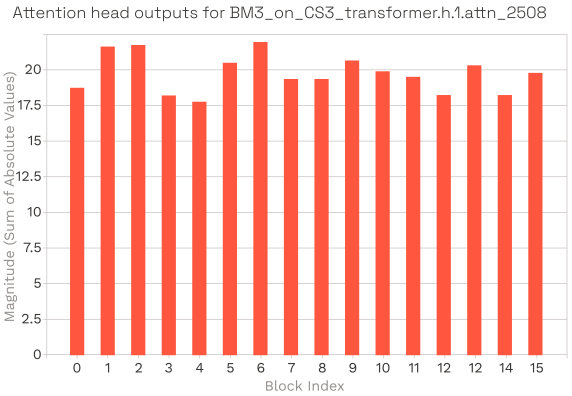
Table 20: Token rank comparison between `COUNT` and `orders` across layers for Example 1.



(a) BM1_CS1 Attention outputs by head for SAE feature 1688 on Layer 0.

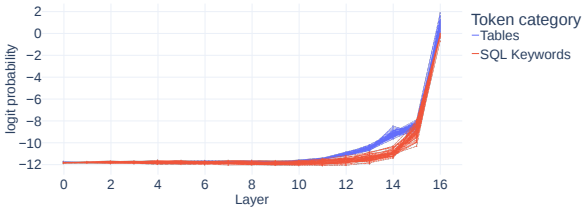


(b) BM1_CS2 Attention outputs by head for SAE feature 1230 on Layer 0.

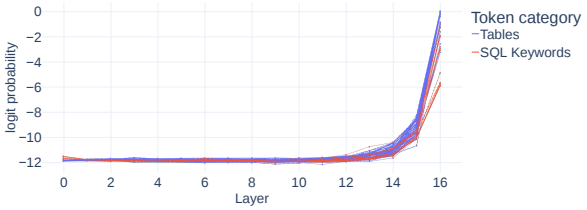


(c) BM3_CS3 Attention outputs by head for SAE feature 2508 on Layer 1.

Figure 12: Attention outputs by head for selected SAE features across models and layers. Top row: BM1 with two different contexts; Bottom: BM3 highlighting deeper-layer behavior.



(a) BM3-CS3



(b) BM3-CS1

Figure 13: Additional LogitLens visualizations across six benchmarks, illustrating consistent two-phase behavior: early SQL intent formation followed by table resolution.

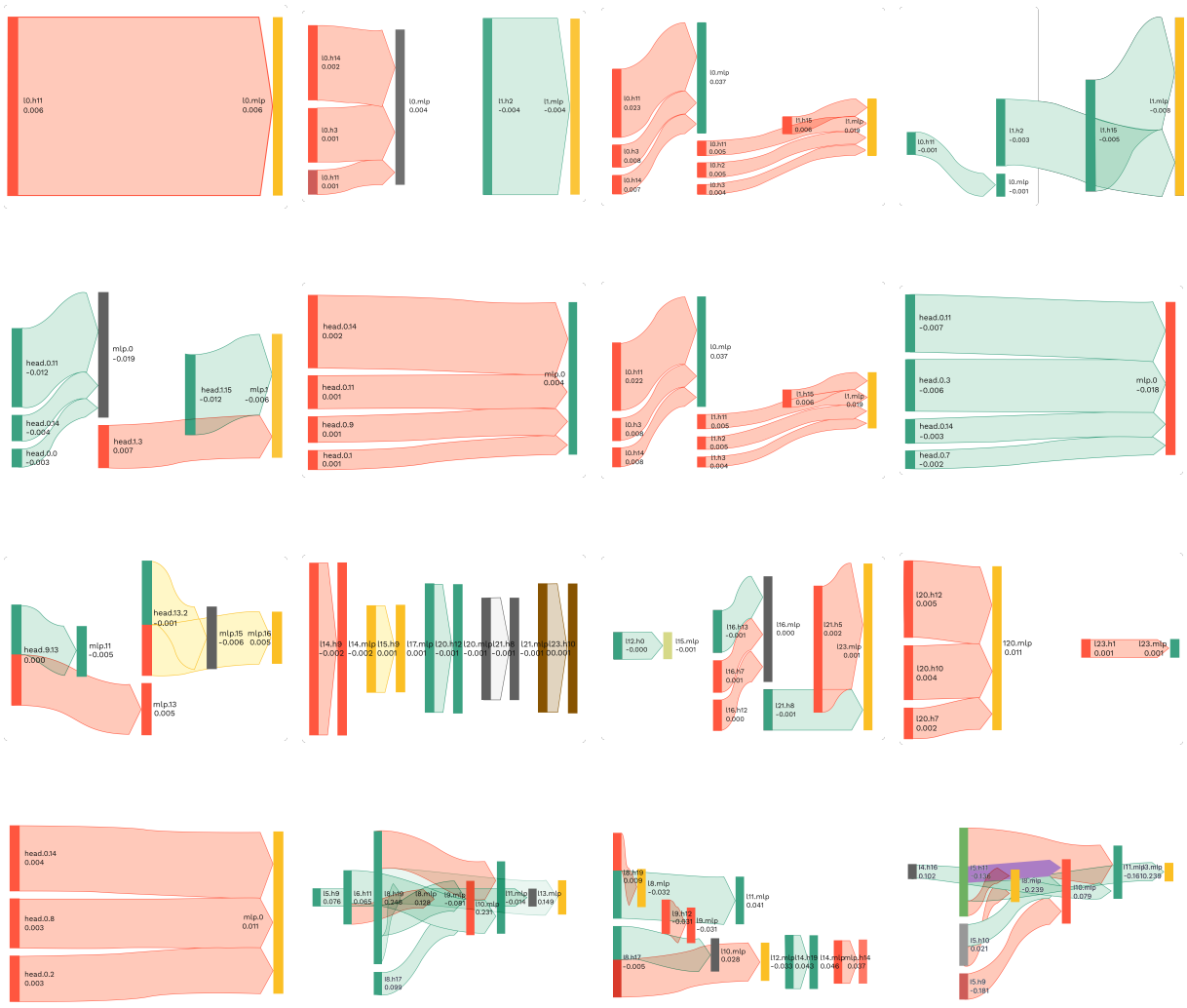


Figure 14: EAP results for different models and command set CS1: BM1_CS1_Syn (row 1), BM1_CS3_Syn (row 2), BM2_CS1_Syn (row 3) and BM3_CS1_Syn (row 4). Each row presents results for EngTableName, EngFieldName, DefTableName, and DefFieldName in that order.

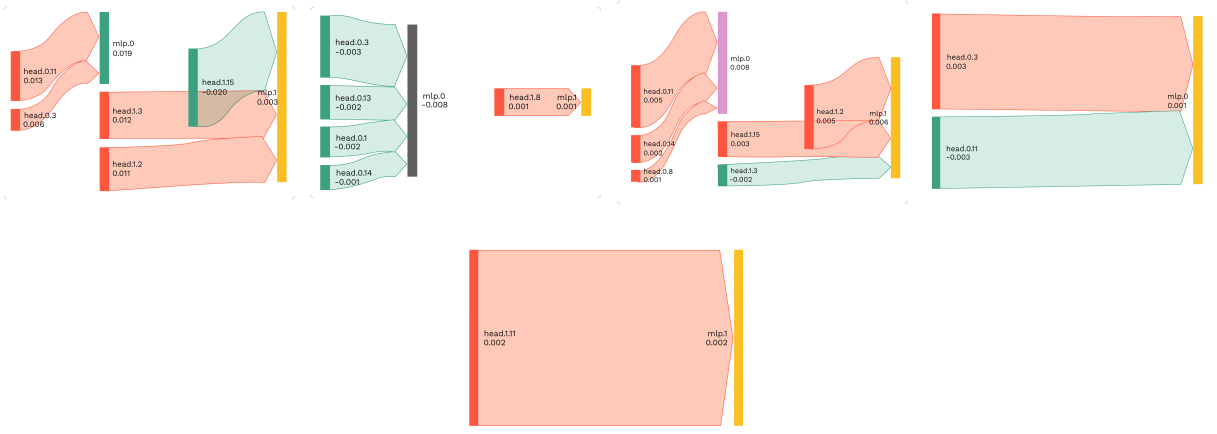


Figure 15: EAP results for model BM1_CS3_Syn on command set CS2: Results are for EngTableName, EngFieldName, DefTableName, DefFieldName and OrderBy in that order.

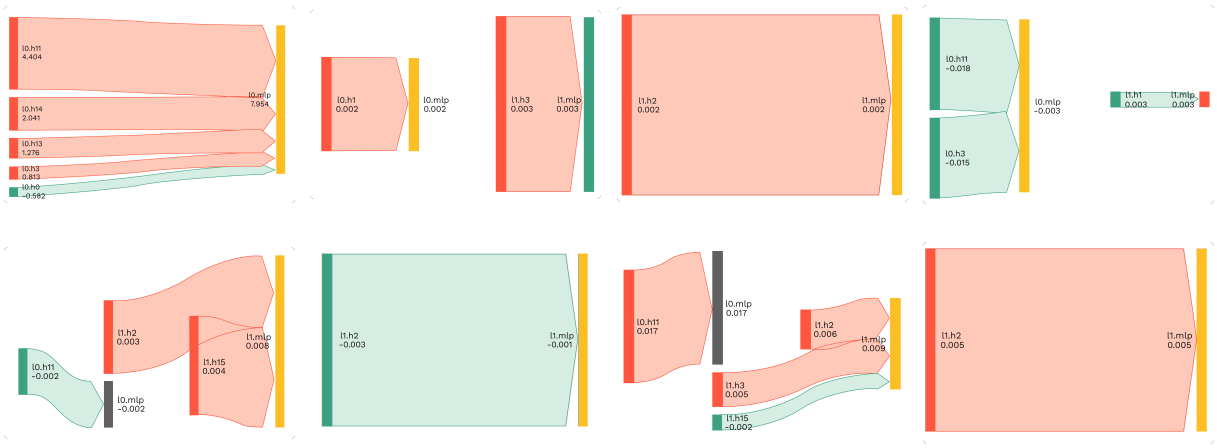


Figure 16: EAP results for model BM1_CS3_Syn on command set CS3: Results are for EngTableName, EngFieldName, DefTableName, DefFieldName, OrderByField, OrderByDirection, AggegrateField and AggregateFunction in that order.

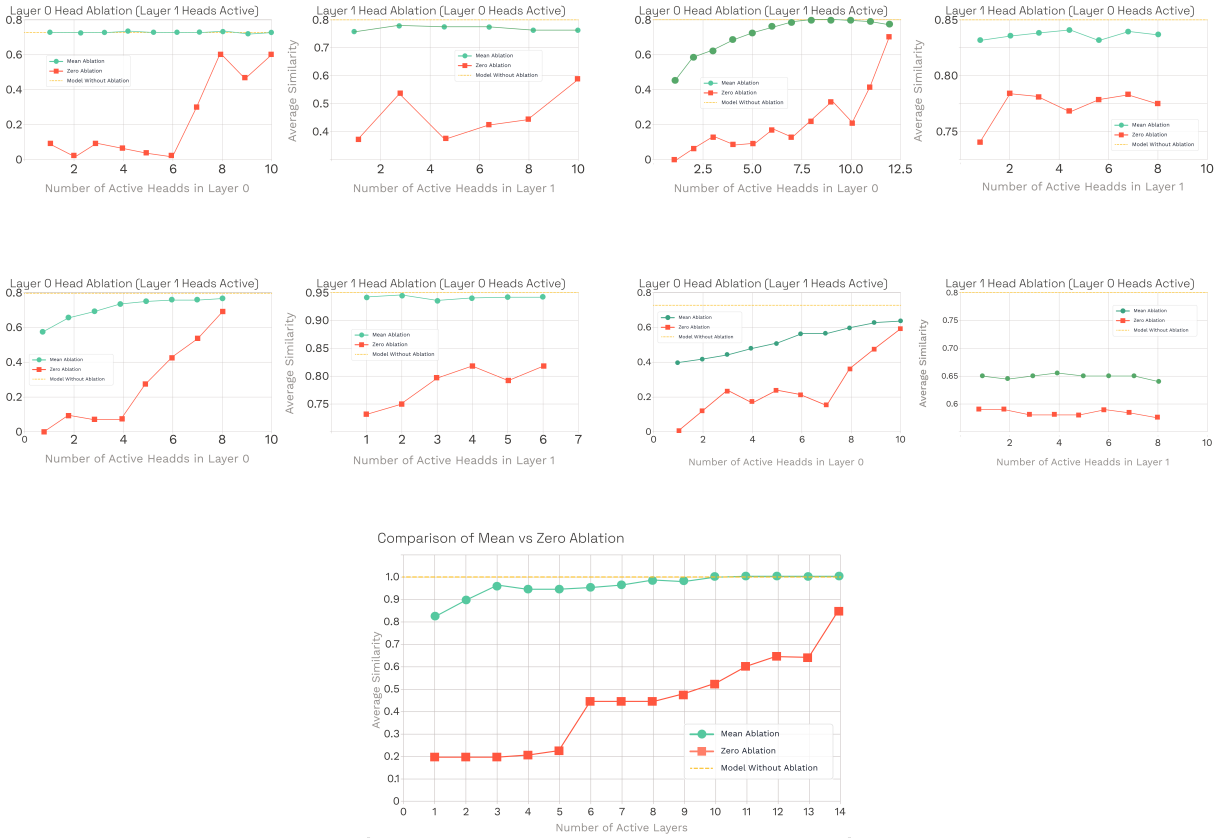
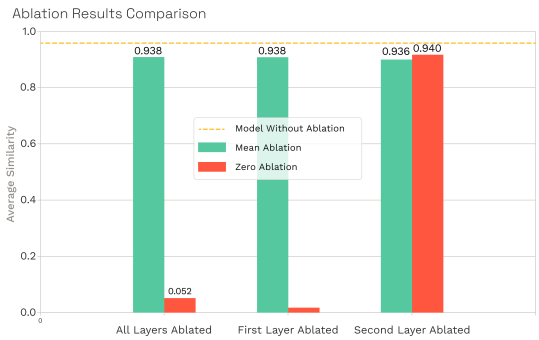
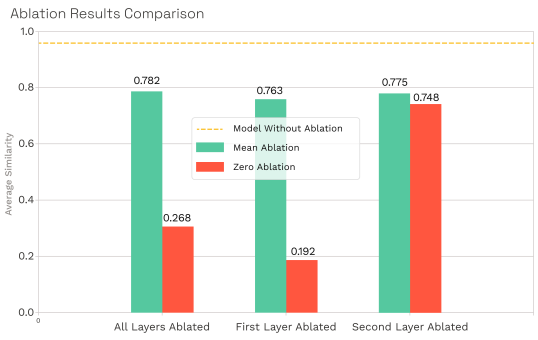


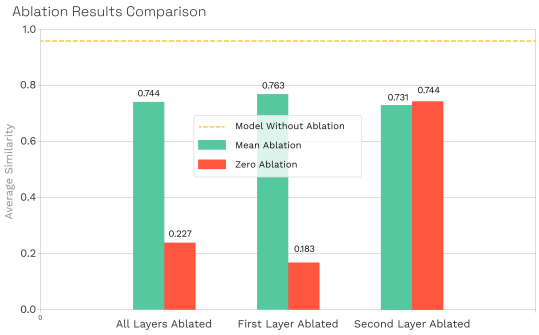
Figure 17: Ablation results showing accuracy trends as more attention heads remain unablated. (Top-left) BM1_CS1_Syn with CS1 data, (Top-right) BM1_CS3_Syn with CS1 data, (Middle-left) BM1_CS3_Syn with CS2 data, (Middle-right) BM1_CS3_Syn with CS3 data, (Bottom) BM2_CS1_Syn with CS1 data.



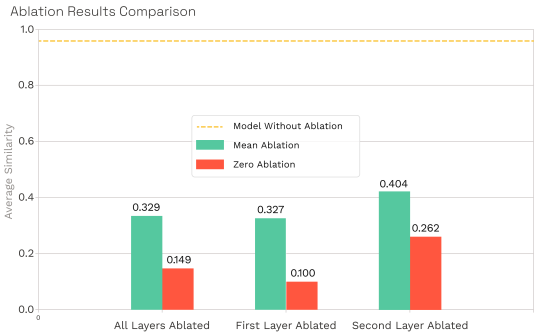
(a) BM1_CS1 with CS1 data



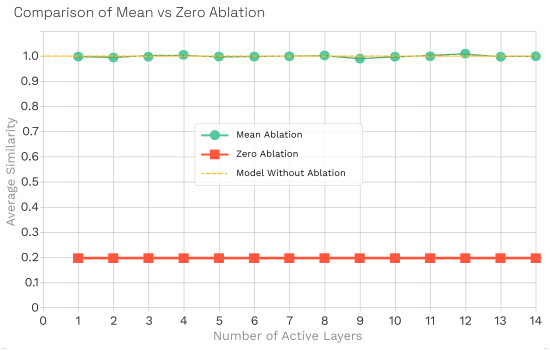
(b) BM1_CS3 with CS1 data



(c) BM1_CS3 with CS2 data

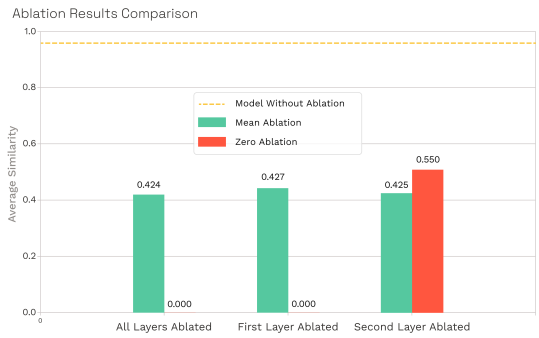


(d) BM1_CS3 with CS3 data



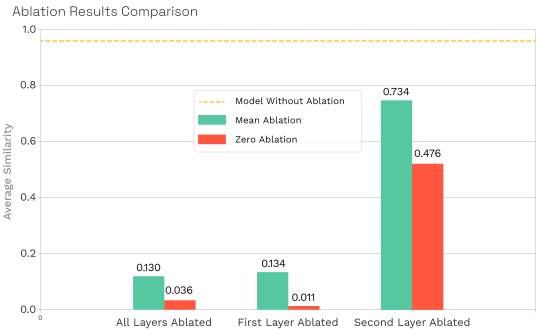
(e) BM2_CS1 data

Figure 18: MLP ablation results across different models and datasets. Each subfigure presents accuracy changes when ablating MLP outputs for specific configurations.

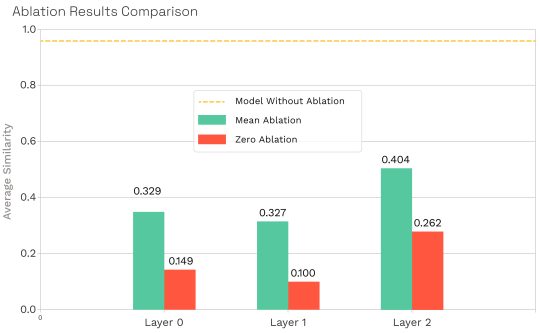


(a) BM1_CS1 with CS1 data

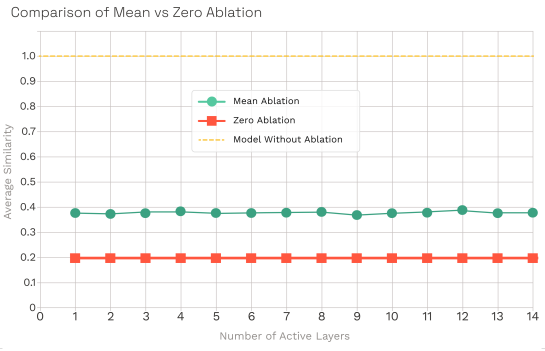
(b) BM1_CS3 with CS1 data



(c) BM1_CS3 with CS2 data



(d) BM1_CS3 with CS3 data



(e) BM2_CS1 data

Figure 19: Results of ablation across all outputs for different models and datasets. Each subfigure illustrates the impact of ablating all outputs in the specified configuration.