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

Anonymous ACL submission

## 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. Our analysis reveals both the potential and limitations of current interpretability methods, showing how circuits can vary even across similar queries. Lastly, we demonstrate how mechanistic interpretability can identify flawed heuristics in models and improve synthetic dataset design. Our work provides a comprehensive framework for evaluating and advancing interpretability techniques while establishing clear boundaries for their reliable application.

## 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., 2019) 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 activation patching experiments reveal the challenges of identifying consistent mechanisms even across similar queries, providing insights into the limitations of current methods.

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 task complexity along multiple dimensions, as well as releasing models of different sizes trained on different subsets of TinySQL, we provide a robust testbed for mechanistic interpretability research.

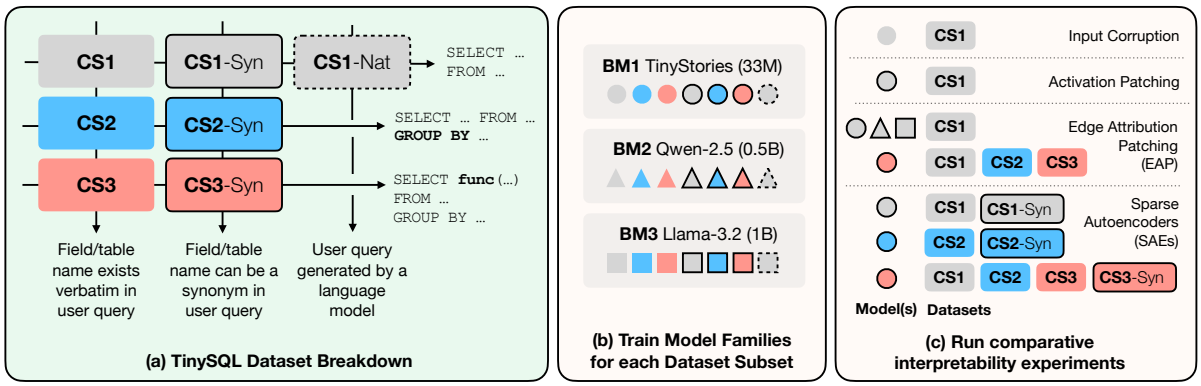



Figure 1: (a) TinySQL is broken down into 7 subsets of varying complexities, across both SQL query and user query axes. (b) We train and release a comprehensive set of models on each dataset subset.(c) We apply MI techniques across various configurations to understand model behavior and compare results.

2. Through systematic application of multiple interpretability methods (EAP, SAE, activation patching), we demonstrate both the strengths and limitations of current techniques for circuit identification. Our analysis reveals how circuits can vary even across similar queries, providing important insights into the reliability of current interpretability methods.

3. We show how mechanistic interpretability can be used to identify and remove unintended shortcuts in synthetic datasets. By tracing model behavior back to specific dataset characteristics, we demonstrate a practical approach for improving synthetic dataset design through interpretability findings.

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.

Recent work with SAEs (Huben et al., 2023; Templeton et al., 2024) has enabled large-scale model behavior analysis, with Marks et al. (2024) using SAE features to uncover interpretable circuits for tasks like subject-verb agreement. However, a key challenge remains: once we understand a model's behavior in one context, the next meaningful step is unclear, hindering systematic research and clear progress in model mechanism understanding.

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.

### 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

Spider (Yu et al., 2019) 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.

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. More recent efforts like Gretel synthetic SQL (Meyer et al., 2024) and SQL-create-context (b mc2, 2023) have balanced synthetic data generation with authentic query patterns, reflecting the community's growing emphasis on combining controlled generation with 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 gretelai/synthetic_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 three 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

**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.

The complete dataset consists of seven task variants (CS1/2/3, CS1/2/3-Syn, and CS1-Nat), each containing 100,000 examples generated through a systematic data creation pipeline that ensures consistency with our design principles.

### 3.2 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).

**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.

### 3.3 Dataset Generation Methodology: CSx_Nat

We additionally explore a synthetic dataset generation method using Gretel AI's Data Designer[1]. 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. More details in App. 3.3.

### 3.4 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.

Each model is fine-tuned on three SQL complexity levels (`CS1`, `CS2`, `CS3`), combined with either base or synonym versions, resulting in six variations. 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. This produces 18 fine-tuned checkpoints (three base models × six datasets). Refer App.D for training details, and App.F.1.1 for examples of the model input format.

**Performance Overview.** All three base models reliably converge on all six dataset variants. For example, fine-tuned `BM1` achieves over 85% exact-match accuracy on CS3_Syn, the most challenging

[1] https://gretel.ai/navigator/data-designer

variant, while BM2 and BM3 exceed 98% on most tasks. See App.D and Tab.4 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 included understanding how a model trained on a different set of tasks performs on one specific task.

### 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).

**Activation Patching.** Given a clean input $x_c$ and corrupted input $x_d$, activation patching copies an activation on the corrupted input $h_i(x_d)$ from node $i$ into the clean run. The causal effect of node $i$ is measured by comparing model outputs with and without patching: $f(x_c; h_i \leftarrow h_i(x_d)) - f(x_c)$, where $f$ is some metric such as the logit difference between two tokens in the output distribution. This identifies which layers are crucial for the target computation (Meng et al., 2022).

**Attribution patching.** Attribution patching is a linear approximation to activation patching around the corrupted input. For node $i$, instead of running the model with the patched activation, we approximate the causal effect as $(h_i(x_c) - h_i(x_d)) \cdot \nabla_{h_i} f(x_d)$, where $\nabla_{h_i}$ is the gradient of the model output with respect to activation $h_i$. This allows computing all patches with just two forward passes and one backward pass (Nanda, 2023).

**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 understand how BM1_CS1 generates field and table names in the CS1 task, we corrupt either the instruction or schema context by replacing names with alternatives (e.g. using "table_a" in the instruction but "table_b" in the CREATE TABLE statement, and similarly for field names). If the corruption does not hurt performance, it implies that the model does not significantly depend on that part of the context.

We find that the model consistently uses table names from the schema context while taking field names from the instruction, ignoring conflicting names in the other source. This strategy works for models trained on CS1 since field names appear identically in both locations. Hence, we created our synonym variants (CSx_Syn), which require models to develop more sophisticated mechanisms for mapping between equivalent terms rather than relying on exact matches. Given these results, we now will focus on models trained on the synonym variants to study more realistic text-to-SQL behaviors.

**Activation Patching.** To minimise confounding variables, we take only the queries in CS1 where the table only has three fields. We analyze BM1_CS1-Syn's field name prediction across four prompt classes in CS1: (1-3) selecting a single field from positions 1, 2, and 3, and (4) predicting the second field after correctly generating the first in a two-field query. For each prompt, we create two corrupted versions - one with an alternative table field and another with a neutral token. We ensure that the clean and corrupted field name is always a single token. We then patch in clean activations on a corrupted run, and measure patch effectiveness

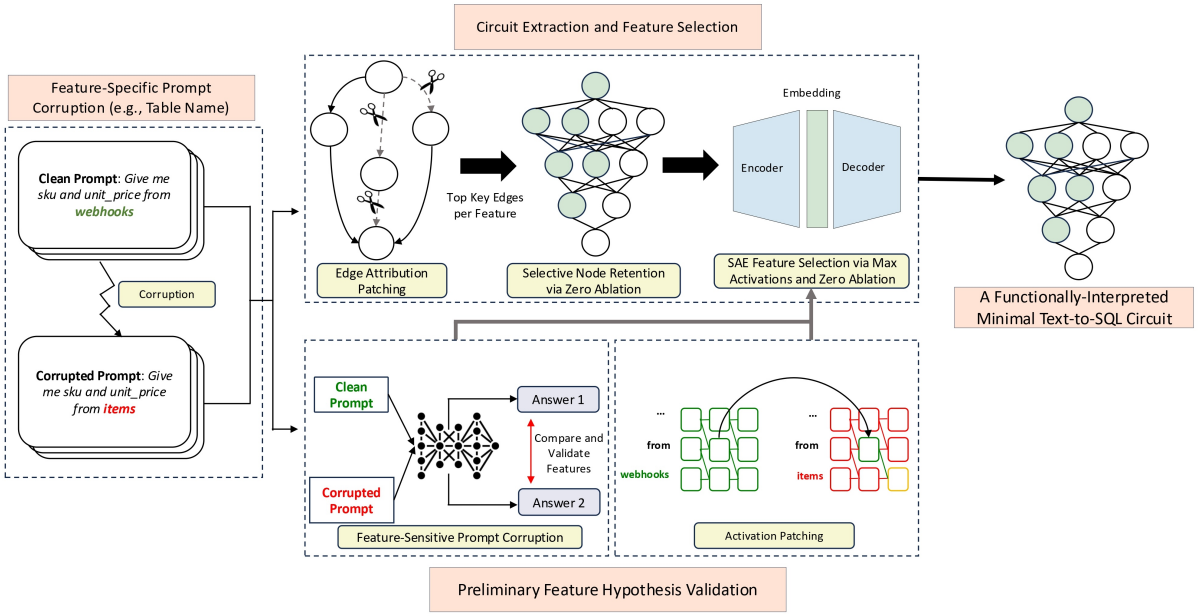



Figure 2: To extract and interpret text-to-SQL functionality, we use Edge Attribution Patching to identify key connections, Selective Node Retention to create a minimal working circuit, and SAE Feature Selection to interpret node functionality. We also use prompt corruption and activation patching to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

using:

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

where $\ell(\cdot)$ denotes the logit difference between the correct and incorrect tokens.

Surprisingly, we observe high variability in important attention heads, even between similar prompts within categories. A set of examples in App. E. This is especially notable given that we have already tried to reduce the variance in tasks by investigating specific prompt classes. This variability motivates our use of complementary techniques like Edge Attribution Patching and sparse autoencoders for more robust analysis.

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

To idenfity the 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. F.1.1). We then performed EAP, selecting the 10 most important edges ranked by the task-specific metric $L$ (Eq. 1), similar to the one used for activation patching, which produces scores near 1 for essential edges and near 0 for less relevant ones.

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 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 these findings, we conducted systematic ablation experiments across multiple model components. We first ablated all the attention heads of the model, except those associated with the edges that EAP identified. We then evaluated the Recovered Accuracy by only using the set of attention heads identified as important. We tested both mean-ablation and zero-ablation techniques, though mean-ablation proved less informative for our analysis. Figures 3, 4, and 17 demonstrate how model performance changes when selectively removing components.

The ablation results are in Table 1. EAP performed better on smaller BM1 models than on larger BM2 and BM3 models. In BM1, it identified precise attention heads critical for the task,

| Model | Dataset | Heads | | Recovered Accuracy (%) |
|---|---|---|---|---|
| | | Layer 0 | Layer 1 | |
| BM1_CS1_Syn | CS1 | 1,3,7,8,11,13,14,15 | 3,7,11,13,14,15 | 85.4 |
| BM1_CS3_Syn | CS1 | 0,1,2,3,5,6,7,8,9,11,13,14 | 2,3,5,8,11,13,15 | 97.6 |
| BM1_CS3_Syn | CS2 | 1,2,3,4,7,8,10,11,13,14 | 2,3,5,8,11,15 | 74.6 |
| BM1_CS3_Syn | CS3 | 0,1,2,3,4,6,7,11,13,14 | 1,2,3,5,7,8,11,15 | 78.3 |
| | | Layers | | |
| BM2_CS1_Syn | CS1 | 1,2,3,5,7,8,11,13,14,15 | | 88.7 |
| BM3_CS1_Syn | CS1 | 1,4,5,6,7,8,9,10,11,12,13,14,15 | | 99.1 |

Table 1: 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.

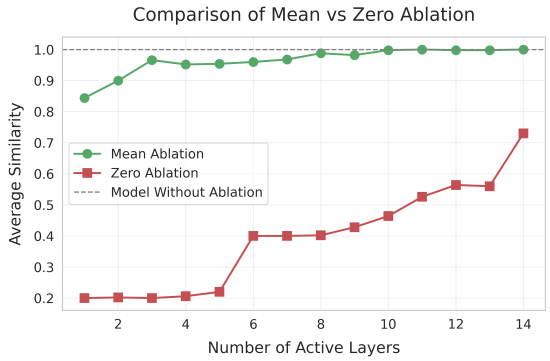



Figure 3: For model BM2_CS1_Syn with CS1 data, accuracy improves with more unablated layers, considering only the attention heads in those layers.

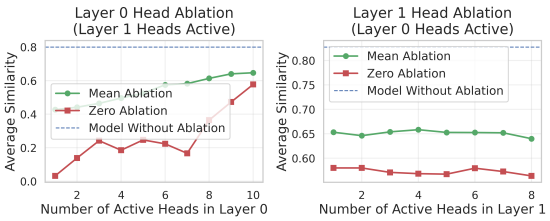



Figure 4: For model BM1_CS3_Syn with CS3 data, accuracy increases with more unablated attention heads.

allowing fine-grained selection. However, in larger models, too few heads were found as the top edges were often between MLP components, resulting in low recovered accuracy (5–10%). To address this, we retained entire layers whenever at least one head was marked important by EAP. This adjustment significantly improved accuracy at the cost of including additional heads.

**MLP and Layer Ablation.** We extended our analysis with systematic ablation of Multi-Layer Perceptrons (MLPs) across different layers, measuring their impact on model performance. Additionally, we performed zero and mean ablation on full layer outputs (including attention heads and MLPs).

We find that removing the first layer consistently degraded performance across all configurations, with accuracy dropping substantially as shown in Figs.19. Removing entire layers (both attention and MLPs) caused greater performance drops than ablating individual components, as seen when compared to MLP-specific ablations (Figs. 18).

### 4.4 SAE Feature Analysis for determining influential features

To understand how models encode schema information, we use SAEs to identify interpretable latent features.

**Experimental Setup.** We first collect activation data from each BMi_CSi_Syn model when applied to its corresponding CSi and CSi_Syn datasets for $i \in 1, 2$. To understand how models trained on complex tasks handle simpler queries, we also create a "descending complexity" dataset by applying BM1_CS2_Syn and BM1_CS3_Syn on CS1 data.

For training the SAEs, we use Belrose (2024)'s package and follow the top-k regime from Gao et al. (2024) with $k = 16$, an expansion factor of 4, and 6 training epochs with a learning rate of $8e - 4$. We find that the Fraction of Unexplained Variance (FVU) remains below 0.05 across all models and datasets. Training took approximately 1 hour for BM1, 2 hours for BM2, and 6 hours for BM3, all on a single A100 GPU. While our analysis focuses on the BM1 SAEs, we also release the suite of

| Model | Dataset | Layer 0 | Layer 1 |
|---|---|---|---|
| BM1_CS1_Syn | CS1 | 1, 8, 10, 7 | 13, 4, 2, 1 |
| BM1_CS1_Syn | CS1_Syn | 1, 10, 8, 7 | 13, 14, 6, 4 |
| BM1_CS2_Syn | CS2 | 1, 10, 8, 14 | 13, 8, 4, 1 |
| BM1_CS2_Syn | CS2_Syn | 1, 10, 8, 2 | 10, 8, 1, 13 |
| BM1_CS3_Syn | CS1 | 1, 8, 10, 14 | 6, 1, 2, 5 |
| BM1_CS3_Syn | CS2 | 1, 8, 10, 14 | 6, 1, 2, 5 |
| BM1_CS3_Syn | CS3 | 1, 10, 8, 14 | 6, 2, 1, 13 |
| BM1_CS3_Syn | CS3_Syn | 1, 10, 8, 7 | 2, 6, 1, 13 |

Table 2: From SAE feature analysis, important attention heads for generating table field tokens accurately. Note the similarity in Layer 0 heads across the three models.

SAEs for BM2 and BM3.

Next, we identify maximally activating SAE features over key input elements and map them to attention head outputs. For instance, we examine SAE features activating over table field tokens in generated SQL, averaged over 200 examples. For each of the top five activating SAE features per layer, we decode them back into original attention-head output, average over these outputs, and then pick the top four heads with largest magnitude activations.

**Results.** Table 2 presents these influential attention heads for BM1's table field processing. Across all three command datasets, BM1_CS*_Syn consistently uses similar attention heads in Layer 0 when processing table field tokens. This consistency suggests that attention mechanisms converge similarly across different trainings and datasets.

Most notably, the BM1_CS3_Syn model employs the same heads across CS1, CS2, CS3, and CS3_Syn datasets, suggesting that the mechanism for processing table field tokens remains largely unchanged despite increasing query complexity. This finding aligns with our understanding that the main differences between these datasets lie in the choice of SQL keywords (such as `GROUP BY`, `SUM`, etc.).

For additional analysis on how the model process table names and `ORDER BY` or `GROUP BY` tokens, see App. G, particularly Table 5 and Table 6.

# 5 Discussion

## 5.1 Differences in MI techniques

Our research demonstrated that different mechanistic interpretability methods can yield varying results. We observed these differences not only between different approaches like SAEs and EAP, but also within individual methods. For instance, our activation patching experiments on a specific subset CS1 prompts (Section 4.2) produced different results, even when the prompts had largely similar structures. Furthermore, researchers must navigate many nuanced decisions that could change results, such as choosing between zero or mean ablation (Miller et al., 2024), or whether to patch a corrupted output into a clean run or vice versa (Heimersheim and Nanda, 2024).

This variability in results represents a fundamental challenge in mechanistic interpretability research. Hence, we believe that TinySQL's controlled progression of tasks and systematic structure makes it an ideal platform for investigating these methodological differences and developing more reliable interpretability techniques.

## 5.2 Improving Synthetic Datasets through MI Analysis

Our initial dataset used newlines to make the SQL query easier to read, but the trained models ended up relying on these newlines to process the field names, leading to poor generalization on differently formatted queries. By removing these formatting artifacts, we stopped the model from exploiting unintended signals and created a cleaner dataset that better tested SQL understanding.

Furthermore, we found that models trained on CS1 to CS3 were predicting the field name based solely on the user query, and not the table schema. This enabled an iterative refinement process where we could identify problematic patterns in model behavior, trace them back to specific dataset characteristics, and make targeted improvements to the synthetic data. This approach helped ensure our datasets actually test the capabilities we intend to measure, rather than allowing models to succeed through undesired shortcuts. Hence, we believe that MI could help improve synthetic datasets on programmatic tasks in this manner.

# 6 Conclusion

We present TinySQL, a dataset with increasingly complex SQL command subsets, as well as trained models of differing sizes. Our analysis of multiple interpretability methods (EAP, SAE, and activation patching) reveals both consistent patterns and limitations in interpreting models, highlighting areas where current interpretability techniques need improvement while providing a testbed for improving the robustness of interpretability techniques.

## Limitations

Time constraints meant we did not include planned more-complex SQL subsets including SQL joins, where clauses and multiple tables. Our findings may not generalize to all SQL dialects or database systems. Also, our mechanistic understanding of the model algorithm is incomplete, and so also is our ability to compare algorithms between models.

While we were to leverage our MI results to improve our training data, we were not able to use our MI results to improve model performance post-training. Given time constraints, we did not run all our analyses on all models, e.g. while we trained SAEs on all of BM1 through BM3, we only studied these on BM1.

## 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


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., 2019).

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

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.

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.

Stefan Heimersheim and Neel Nanda. 2024. How to use and interpret activation patching. *arXiv preprint arXiv:2404.15255*.

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*.

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*.

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. ArXiv:2202.05262.

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.

Joseph Miller, Bilal Chughtai, and William Saunders. 2024. Transformer circuit evaluation metrics are not robust. In *First Conference on Language Modeling*.

Neel Nanda. 2023. Attribution patching: Activation patching at industrial scale.

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*.

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*.

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*.

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

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 Table3

| | 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 3: 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 Statistics for CSx_Syn

The full dataset contains 300,000 queries across all variants split into train, validation, and test sets, with 100,000 queries per command set. The average query length using the `meta-llama/Llama-3.2-1B-Instruct` tokenizer ranges from 12.91 tokens for CS1, 26.52 tokens for CS2, and 41.76 tokens for CS3.

## C.2 Dataset Generation Methodology: CS1_Nat

We use Gretel AI's Data Designer[2], 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`).

[2] https://gretel.ai/navigator/data-designer

- 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:
      - ``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.

## 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.

**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 $1 \times 10^{-5}$, 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 4.

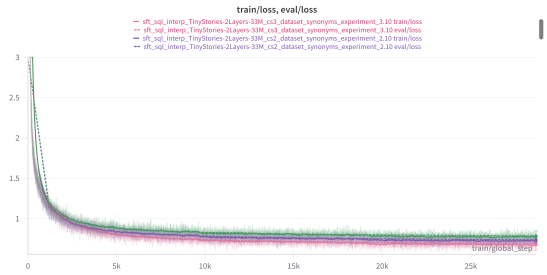



Figure 5: Training and Validation loss curves for instruction tuning BM1 (TinyStories-Instruct-2Layers-33M) on CS1_Syn, CS2_Syn, CS3_Syn

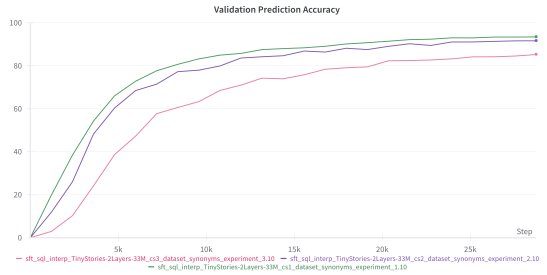



Figure 6: Accuracy curves for instruction tuning BM1 (TinyStories-Instruct-2Layers-33M) on CS1_Syn, CS2_Syn, CS3_Syn

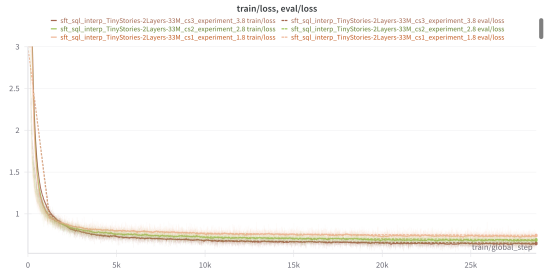



Figure 7: Training and Validation loss curves for instruction tuning BM1 (TinyStories-Instruct-2Layers-33M) on CS1, CS2, CS3

# E 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 9 in App.E, and we note that this variation persists throughout all examples.

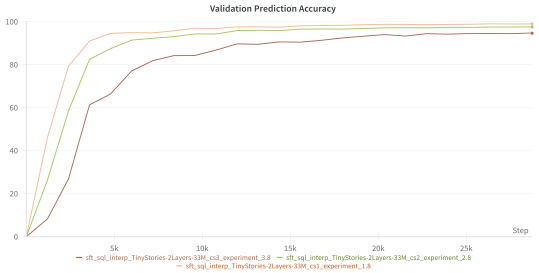



Figure 8: Accuracy curves for instruction tuning BM1 (TinyStories-Instruct-2Layers-33M) on CS1, CS2, CS3

# F Circuit Identification

## F.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.

### F.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

| 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 4: 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

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)

#### F.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.

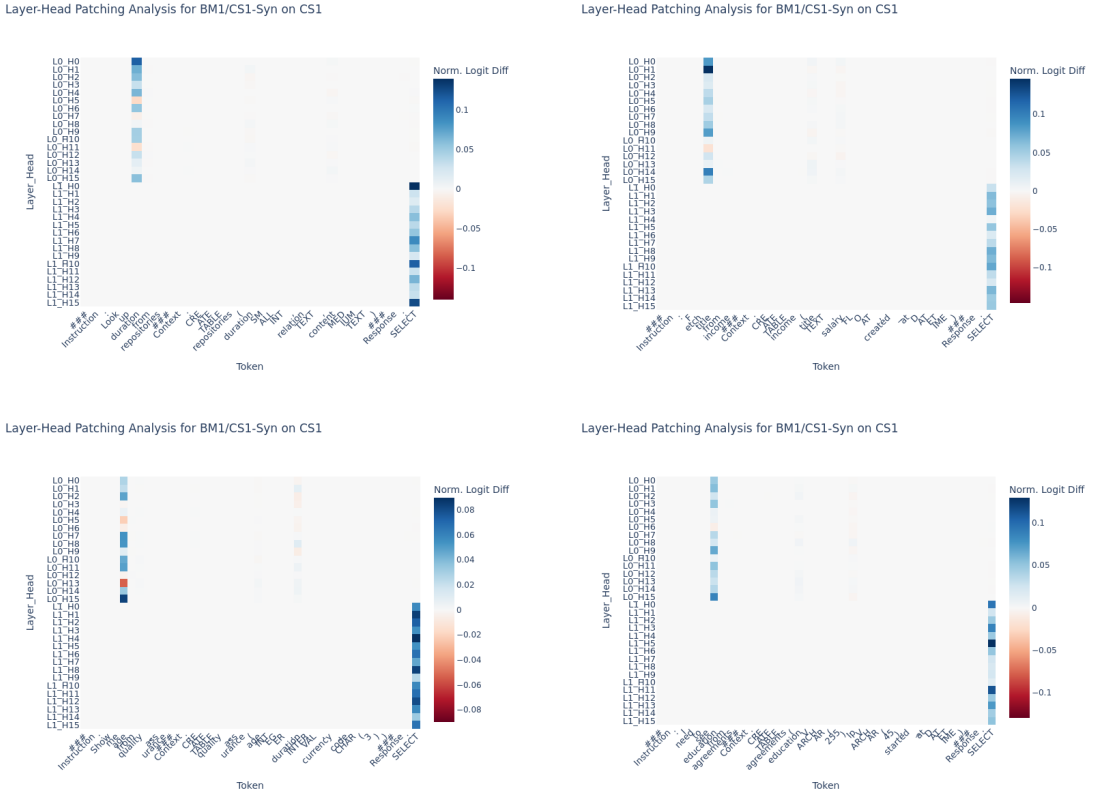



Figure 9: Activation patching results for 4 prompts, where the table contains three fields and the user queries for only the first field of the table. Even when controlling for some variables, the results differ from prompt to prompt.

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.

### F.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.

### F.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.

### F.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 5: From SAE feature analysis, important attention heads for generating table name tokens accurately.

| 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 6: From SAE feature analysis, important attention heads for generating order by and sort by tokens accurately.

## G SAE analysis

We show the important heads in BM1 for generating table tokens next in Table 5. We also consider the important SAE features for generating the field tokens used for "ORDER BY" or "GROUP BY" type queries in Table 6.

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.

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 10, 11 and 12 for the top activating SAE features by average magnitude.

We can map these to the actual attention outputs, see for instance Figures 13a, 13b and 13c. We average over these to determine the influential attention heads.

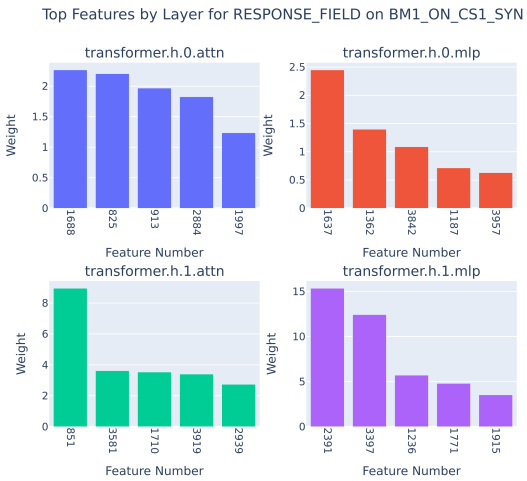



Figure 10: Average SAE activations for BM1_CS1_Syn on CS1_Syn table fields.

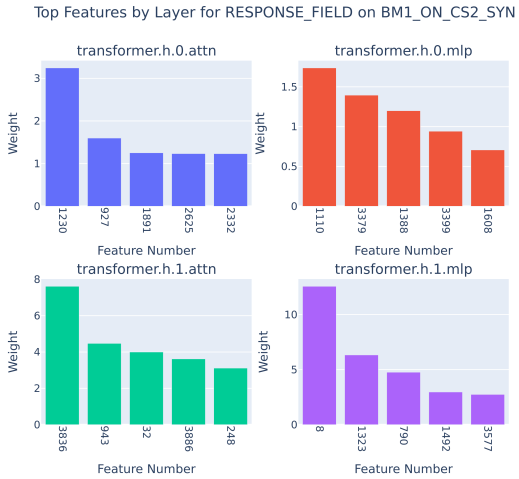
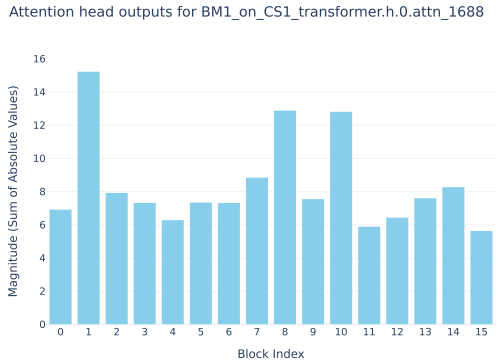
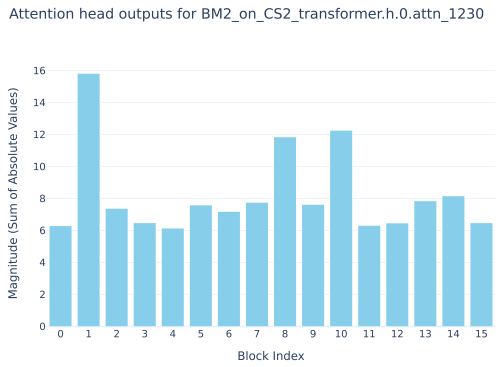



Figure 11: Average SAE activations for BM1_CS2_Syn on CS2_Syn table fields.

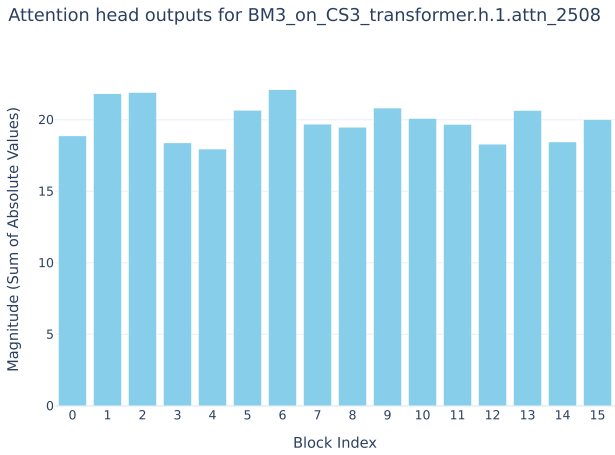

Figure 12: Average SAE activations for BM1_CS3_Syn on CS3_Syn table fields.




(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 13: Attention outputs by head for selected SAE features across models and layers.

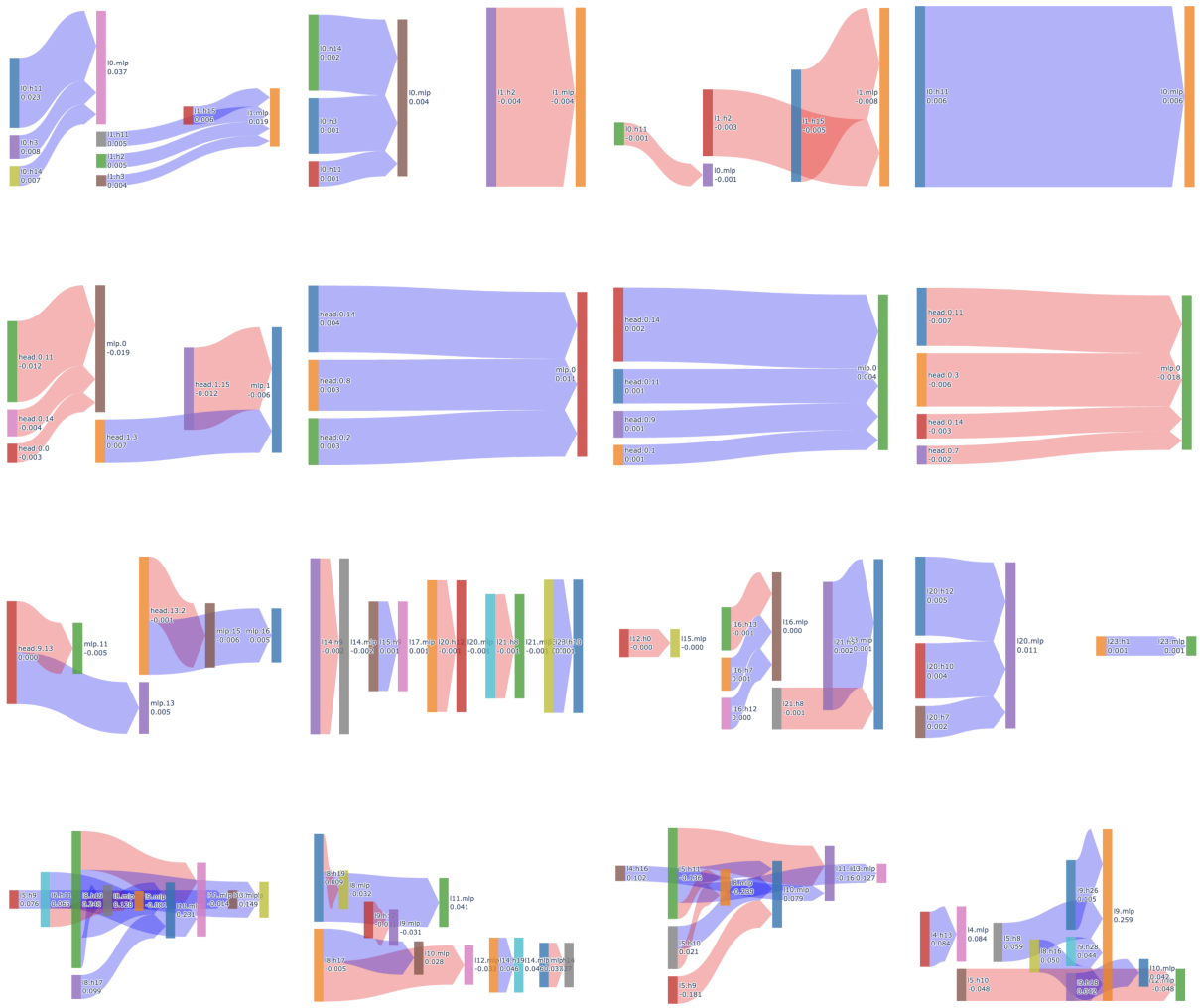


Figure 14: EAP results for different models and command set CS1: BM1_CS1_SYN (row 1), BM1_CS3_SYN (row 2), BM2_CS1_SYN (row 3) and BM3_CS1_SYN (row 4). Each row presents results for EngTableName, EngFieldName, DefTableName, and DefFieldName in that order.

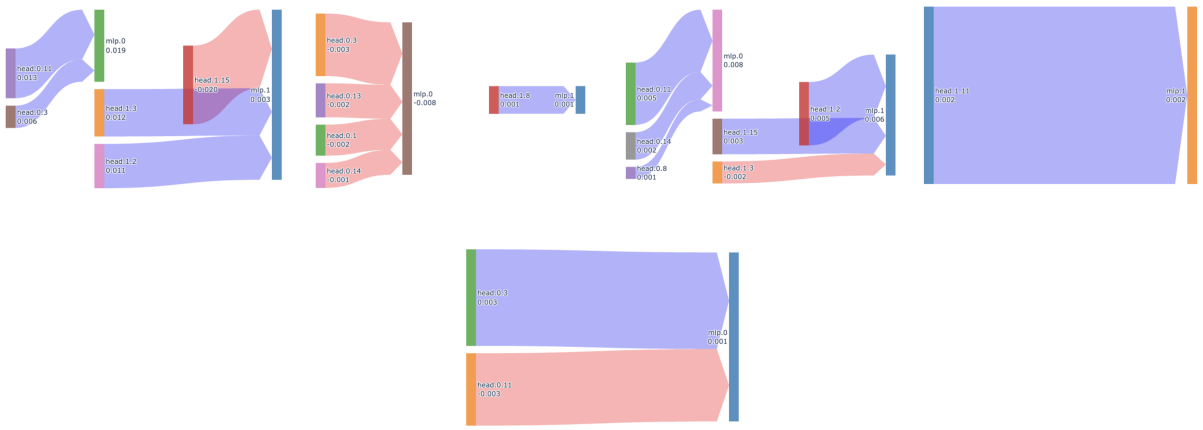


Figure 15: EAP results for model BM1_CS3_SYN on command set CS2: Results are for EngTableName, EngFieldName, DefTableName, DefFieldName and OrderBy in that order.

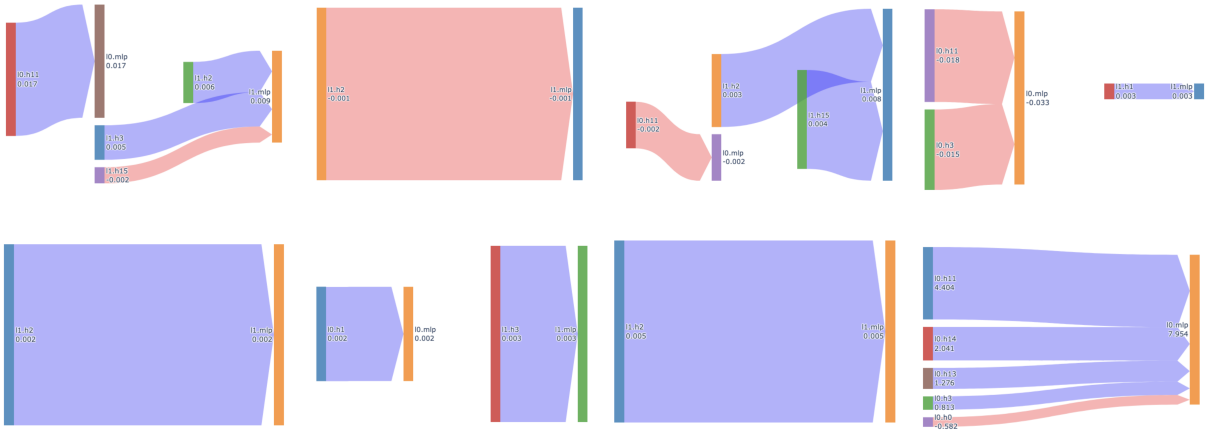


Figure 16: EAP results for model BM1_CS3_SYN on command set CS3: Results are for EngTableName, EngFieldName, DefTableName, DefFieldName, OrderByField, OrderByDirection, AggegrateField and AggregateFunction in that order.

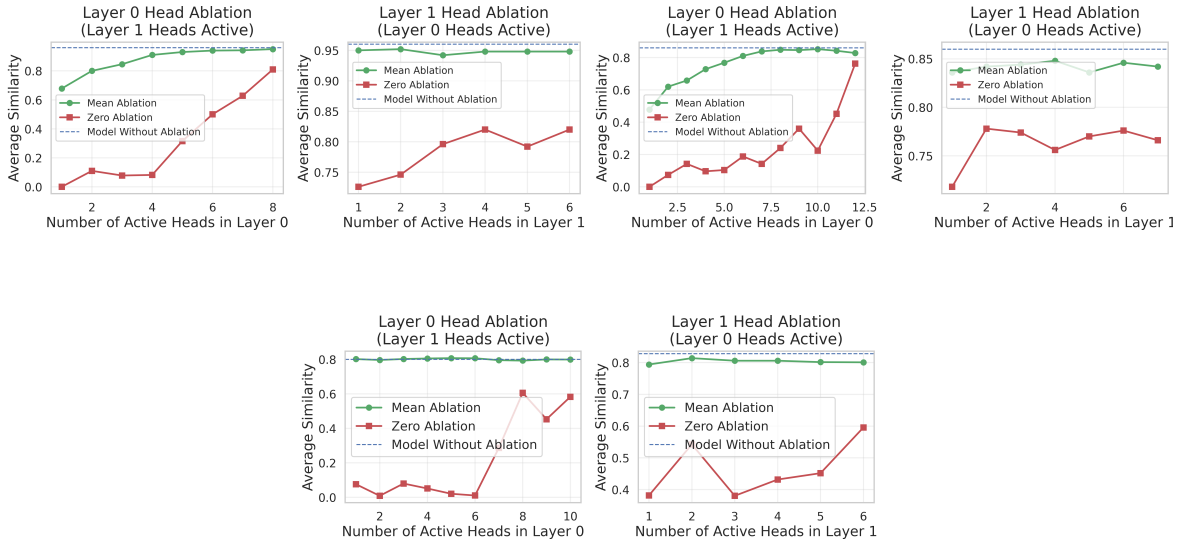
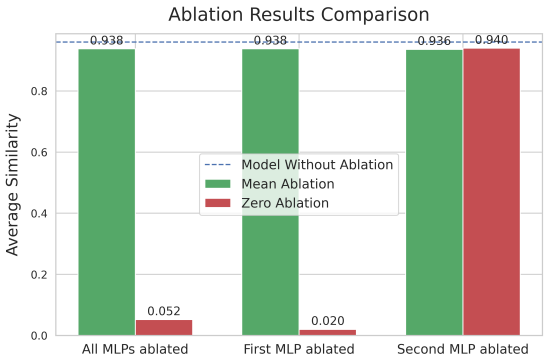
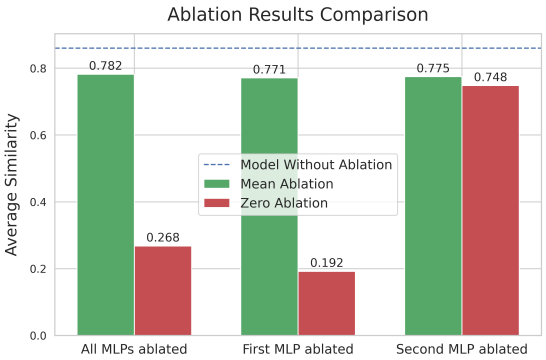


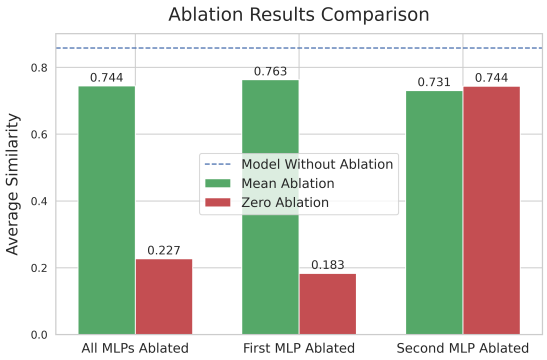

Figure 17: Ablation results showing accuracy trends as more attention heads remain unablated. (Top-left) BM1_CS1_Syn with CS1 data, (Top-right) BM1_CS3_Syn with CS1 data, (Bottom) BM1_CS3_Syn with CS2 data.



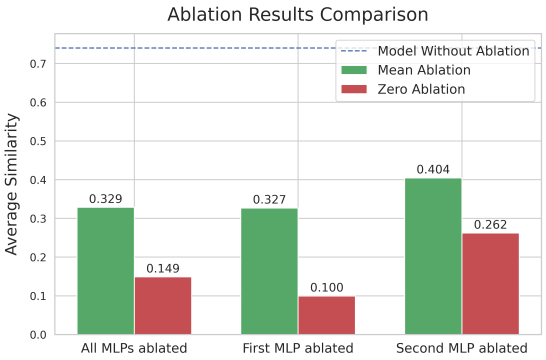

(a) BM1_CS1 with CS1 data



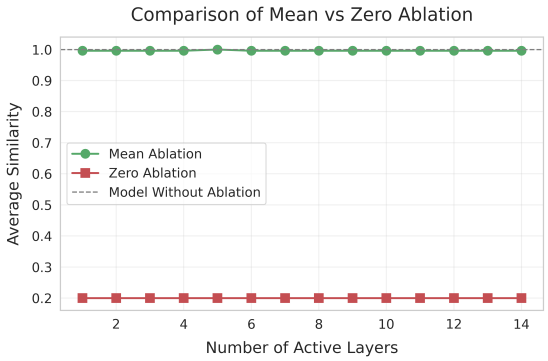

(b) BM1_CS3 with CS1 data




(c) BM1_CS3 with CS2 data

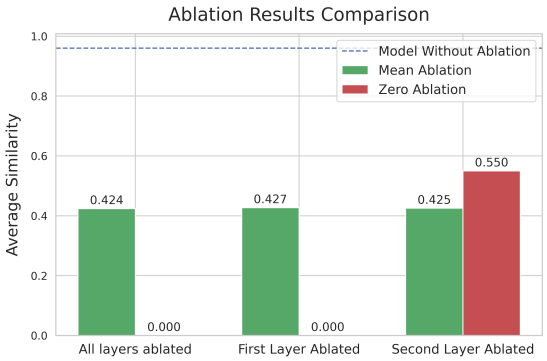



(d) BM1_CS3 with CS3 data

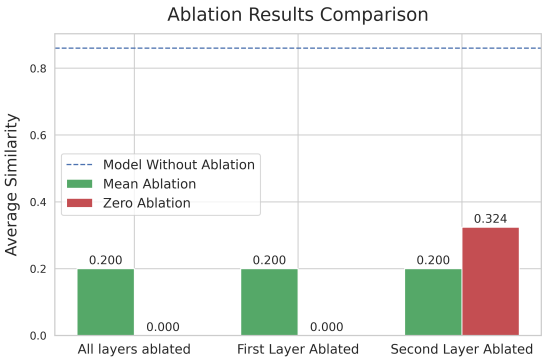



(e) BM2_CS1 data

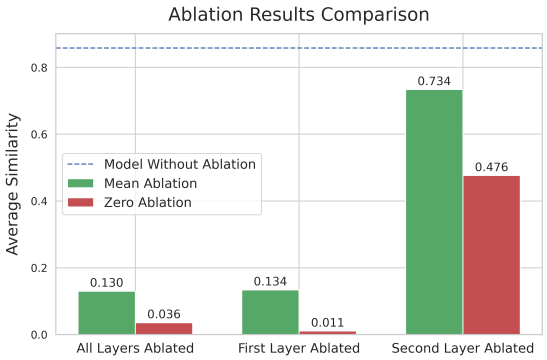
Figure 18: MLP ablation results across different models and datasets. Each subfigure presents accuracy changes when ablating MLP outputs for specific configurations.



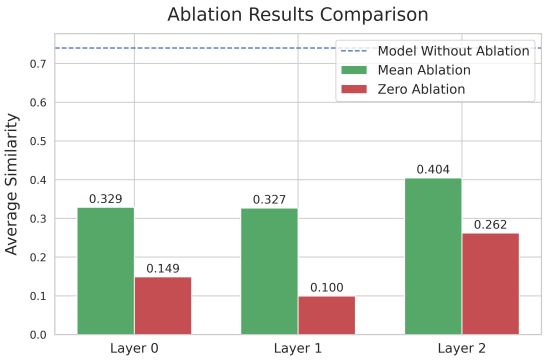

(a) BM1_CS1 with CS1 data



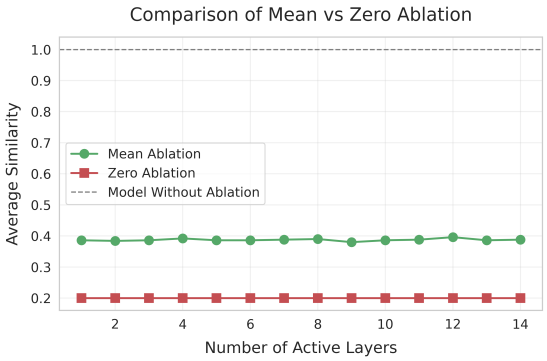

(b) BM1_CS3 with CS1 data




(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.