Supplementary Material

Mechanistic Interpretability of Analogical Reasoning in Gemma-2-2B

Olalekan Alagbe · Joseph Lawrence · Anish Maheshwar · Konstantinos Krampis

March 2026

S1. Inference Prompts

The following five prompts were submitted to Gemma-2-2B to generate the attribution graphs analyzed in this paper. Each prompt uses the standard analogical reasoning format *"A is to B as C is to"* (open-ended completion):

#	Prompt	Slug	Category
1	Paris is to France as Berlin is to	`analog_berlin`	Geographic
2	Paris is to France as Rome is to	`analog_rome`	Geographic
3	Paris is to France as Tokyo is to	`analog_tokyo`	Geographic
4	Doctor is to hospital as teacher is to	`analog_teacher`	Semantic Role
5	Fish is to water as bird is to	`analog_bird`	Semantic Role

All five prompts were run on **Gemma-2-2B** using the **gemmascope-transcoder-16k** SAE via the Neuronpedia API. The three geographic prompts share the same source pair (Paris→France) but vary the query city; the two semantic role prompts test cross-domain generalization.

S2. Interactive Attribution Graphs on Neuronpedia

Each graph is hosted on Neuronpedia and can be explored interactively. Clicking a link opens the graph viewer, which shows node activations, edge weights between SAE features, and human-readable feature labels.

- **analog_berlin** — *"Paris is to France as Berlin is to"*
- **analog_rome** — *"Paris is to France as Rome is to"*
- **analog_tokyo** — *"Paris is to France as Tokyo is to"*
- **analog_teacher** — *"Doctor is to hospital as teacher is to"*
- **analog_bird** — *"Fish is to water as bird is to"*

The 180-feature core circuit (features active across all five graphs) and the 277-feature high-confidence set (≥4/5 graphs) are described in the main paper.

S3. Agent Pipeline Prompts

S3.1 Initial Research Prompt

The following prompt initiated the first research session in Claude Code (run by Olalekan Alagbe). It produced the initial `ANALOGICAL_CIRCUIT_REPORT.md` containing the raw results later refined into the main paper:

> I have `autocircuit_tools.py` in this folder. First run it with: `python autocircuit_tools.py` to confirm Neuronpedia is connected. Then use the functions in that file to find the analogical circuit in Gemma-2-2B. Generate graphs for all 5 analogical prompts, compare them to find recurring features, label the top candidates, validate the most important ones with `steer_feature()`, save the circuit, and write me a full comprehensive research paper in an md file of what you found.

S3.2 Structured Pipeline Prompt

The following detailed prompt was used in a subsequent session to run a structured multi-step pipeline over all five analogical prompts. The `CATEGORY` variable at the top selects the prompt set:

```
# ── CONFIGURATION — change only this line to switch categories ──────────────
CATEGORY = "analogical"  # options: "analogical" | "factual_recall" | "linguistic"
# ────────────────────────────────────────────────────────────────────────────

1. Run `python autocircuit_tools.py` to confirm Neuronpedia is connected. Stop and
   report if it fails.

2. Import autocircuit_tools and call build_prompt_dataset(). Extract the prompt list
   for CATEGORY. Do NOT deduplicate — the list is already clean. Print the category
   name and the exact count of prompts before proceeding.

3. Generate attribution graphs for every prompt in the list using generate_graph().
   Sleep 2 seconds between each call. If any single graph fails, print the error,
   skip it, and continue. Track which prompts succeeded and which failed. Print a
   summary at the end: N succeeded, M failed.

4. Load all successfully saved graphs from the graphs/ folder using load_graph().
   Match by slug (first 40 chars of prompt, lowercased, spaces → underscores).
   Store each as a tuple (G, graph_data). Print the count of graphs loaded.

5. ── ACROSS-GRAPHS ANALYSIS ──────────────────────────────────────────────────
   Run compare_graphs() on the loaded graphs with min_appearances = 30% of the
   graph count (round up). This finds features that recur consistently across
   prompts — these are the shared computational building blocks of the CATEGORY
```

circuit, not prompt-specific noise.

Then call extract_node_ids_from_recurring() with top_n=15 to get one representative node_id per recurring feature. Call label_nodes_batch() on those 15 node_ids with delay=0.5. Merge labels back into the recurring features list so each entry has: layer, feature, appearances, avg_influence, and human-readable label.

Print the full merged table, then interpret it: which features appear most universally? Do the labels cluster into recognisable functional groups (e.g. structural/syntactic features in early layers, domain knowledge in middle layers, answer-selection features in late layers)? What does this pattern suggest about how the model implements CATEGORY reasoning as a shared circuit?

6. ── PER-PROMPT INTERPRETATION ──────────────────────────────────────

Before starting, create {CATEGORY}_circuit_paper.md and write the paper skeleton: Abstract (leave as placeholder), Introduction, Methods, then a Results section with:
- The recurring features table from step 5 already filled in
- A "Per-prompt circuit interpretations" sub-section header, left empty

Then for each prompt's (G, graph_data) tuple, call:
interpret_prompt_graph(G, graph_data, paper_path="{CATEGORY}_circuit_paper.md")

Each call labels 40 nodes (top_n_nodes=40) grouped into early / middle / late bands and prints them as they arrive. The narration style guide prints once on the first prompt only.

After EACH call returns, immediately write the mechanistic narrative for that prompt into the paper (replacing the [Claude Code — write narrative here] marker). Do this one prompt at a time. The narrative should explain what each band means computationally, not just list the labels.

7. ── STEERING VALIDATION ──

Pick the top 3 recurring features by avg_influence from step 5.
For each feature:
a. Call steer_feature() on a representative prompt from the CATEGORY list with strength=20.0 and strength_multiplier=4.0. Record baseline output, steered output, and whether it changed.
b. Call get_subgraph_node_ids_for_feature(G, layer, feature, depth=1) on that prompt's graph to find the immediate circuit neighbourhood of the steered feature. Call label_nodes_batch() on those node_ids (delay=0.5) to label the nodes feeding into and out of the steered feature.

Print a summary for each: feature label | baseline | steered | changed | labelled neighbourhood nodes.

```
8. Call save_circuit() with:
   - nodes: merged recurring-feature list from step 5 (with labels)
   - validation_results: steer results from step 7 (all 3 features) including
     steered output, changed flag, and neighbourhood labels
   - prompt_category: CATEGORY
   - source_graphs: slugs of all successfully loaded graphs
   - name and description summarising the discovered circuit

9. ── COMPLETE THE PAPER ──────────────────────────────────────────────────
   Per-prompt interpretations were written incrementally in step 6. Now add:
   - Abstract: fill in with actual numbers (prompts run, graphs generated,
     recurring features found, top feature labels, steering results summary)
   - Steering interventions sub-section in Results:
    feature | label | baseline | steered output | changed | neighbourhood nodes
   - Discussion: what do the recurring features reveal about how Gemma-2-2B
    implements CATEGORY reasoning? How does the layer distribution of recurring
     features compare to what you would expect? Any surprising features?
   - Limitations and Conclusion
   Use actual numbers throughout. No placeholders.
```

S4. Tooling

The analysis relied on `autocircuit_tools.py`, a Python library wrapping the Neuronpedia API that provides:

Function	Description
`generate_graph(prompt, slug)`	Submit a prompt and retrieve a full attribution graph
`load_graph(slug)`	Load a previously saved graph from disk
`compare_graphs(graphs, min_appearances)`	Find SAE features recurring across multiple graphs
`extract_node_ids_from_recurring(features, top_n)`	Get representative node IDs for batch labeling
`label_nodes_batch(node_ids, delay)`	Retrieve human-readable Neuronpedia feature labels
`interpret_prompt_graph(G, graph_data, ...)`	Label 40 nodes per graph grouped by layer band
`steer_feature(prompt, layer, feature, strength)`	Causal intervention via feature activation steering
`get_subgraph_node_ids_for_feature(G, layer, feature, depth)`	Retrieve the circuit neighbourhood of a feature
`save_circuit(nodes, validation_results, ...)`	Persist the discovered circuit to JSON

Source code: github.com/kkrampis/autocircuit

Model: Gemma-2-2B (Google DeepMind)

SAE: gemmascope-transcoder-16k via Neuronpedia