Escuela Politécnica Superior

Mechanistic Interpretability of Transformers

Master's degree in Artificial Intelligence

Natural Language Processing Techniques

Authors:
Cristian Andrés Córdoba Silvestre
Rebeca Piñol Galera

Teacher:
Juan Antonio Pérez Ortiz

Universitat d'Alacant
Universidad de Alicante

January 2026

Mechanistic Interpretability of Transformers

Natural Language Processing Techniques

Authors

Cristian Andrés Córdoba Silvestre

Rebeca Piñol García

Tutor

Juan Antonio Pérez Ortiz

Master's degree in Artificial Intelligence

Escuela Politécnica Superior

Universitat d'Alacant
Universidad de Alicante

ALACANT, January 2026

Table of Contents

List of figures

List of tables

1. Inputs and Tokenization

URL[1] of the GitHub repository.

At the Beginning, the `GPT.forward` method has been extended with:

- optional **clean-run caching** of layer/position activations,
- optional **single-coordinate patching** during a corrupted run, and
- extraction of **last-position logits** for evaluation.

Clean and corrupted prompts are tokenized into BPE token-id sequences, and two conditions are enforced:

1. **identical token length**
2. **exactly one differing token position**.

This verification is implemented by constructing tokenization reports for each prompt, comparing their token-id sequences position-wise, and validating that the resulting diff set has cardinality one.

[1] URL of the GitHub Repository containing the code of the project": ***https://github.com/cacs2-ua/tpln-practice2.git***

2. Model's Stored Representations and The Patching Mechanism

For patching, an **activation** is defined as the **residual stream representation at the output of each transformer block**, recorded **for every token position** in the prompt (equivalently, the block output after the attention and MLP sublayers have contributed through residual addition). In the implemented forward pass, this corresponds to the per-layer tensor slice `x_out[0, p, :]`, which is stored as a vector in $R^{d_{model}}$.

The **clean-run caching** mechanism performs a single forward pass on the clean prompt with caching enabled. During this run, the model records activations as a nested structure indexed by layer and position, i.e.,

$$[L][P] \mapsto \mathbf{h}_{L,P} \in R^{d_{model}},$$

where $L \in \{0, \ldots, n_{layers} - 1\}$ and $P \in \{0, \ldots, T - 1\}$. Importantly, each stored vector is a **defensive copy** created via `detach().clone()`. The implementation further guarantees cache safety by disallowing cache-writing during patched runs: if patching is requested, the forward pass refuses `cache_activations=True` and refuses `overwrite_cache=True`, thereby ensuring corrupted and patched evaluations cannot overwrite the clean cache.

The evaluation target is derived from the model's **last-position logits**, i.e., the next-token distribution after consuming the prompt. After the final layer normalization and linear language-model head, the forward pass stores:

$$\texttt{self.last\_logits} = \texttt{logits[:, -1, :].detach().clone()},$$

which is a $(B, |\mathcal{V}|)$ tensor (with $B = 1$ in our experiments). The scalar metric used is then computed from this vector as

$$score = logit(Token\_B) - logit(Token\_A),$$

where Token A is the clean-consistent continuation and Token B is the corrupted-consistent continuation (meeting the requirement that each is a **single BPE token id**).

The **patching rule** constitutes the causal intervention: during a corrupted run, for a chosen coordinate (L, P), the model replaces the corrupted activation at that same coordinate with the corresponding cached clean activation. Operationally, once the forward computation

reaches the target layer L, the state tensor is cloned, and the vector at `x_state[0, P, :]` is overwritten with the cached clean vector $\mathbf{h}_{L,P}^{clean}$. Exactly **one** patch is applied per run. The minimal external parameterization exposed to the experimental drivers is therefore:

- whether to record/cache activations (clean run),
- the patch target (L, P) for a patched corrupted run, and
- access to `last_logits` for metric evaluation.

3. Baselines and Full Sweep Matrix

The experimental workflow begins with a **baseline comparison** that establishes the causal contrast the heatmap is meant to explain. First, the clean prompt is run once with activation caching enabled, producing (i) the clean activation cache and (ii) the clean last-position logits from which the scalar score is computed. Second, the corrupted prompt is run *without* caching and *without* patching, producing a corrupted score computed from its last-position logits using the same token pair and metric definition. Conceptually, the clean run should favor the clean-consistent continuation (lower score if Token A dominates), while the corrupted run should shift preference toward the corrupted-consistent continuation (higher score if Token B dominates); the drivers optionally display a short *top-k* summary of next-token probabilities to confirm that the corruption meaningfully changes the model's continuation distribution.

The main experiment is the **full activation patching sweep**, which produces the primary artifact used for later visualization and interpretation. After caching clean activations once, the code iterates exhaustively over all transformer layers and all prompt token positions. For each coordinate (L, P), it runs the **corrupted** input while applying a patch at exactly that coordinate—replacing the corrupted activation with the cached clean activation at the same (L, P)—then recomputes the scalar score from the resulting last-position logits. These values are stored in a matrix

$$M \in R^{n_{layers} \times T},$$

where T is the (shared) BPE token length of the prompts. The interpretation is deliberately tight: $M[L, P]$ **quantifies how restoring the clean activation at layer** L **and position** P **changes the model's next-token preference**, as measured by $logit(B) - logit(A)$, thereby yielding a layer-by-position causal map of which internal states carry the information responsible for the clean-versus-corrupted divergence.

Finally, the sweep output is saved as a reproducible intermediate artifact which is saved to be used for the later visualization and interpretation.

4. Experimentation

4.1 Main Part

For the experimental part, we selected the following prompt as the **clean input**:

CLEAN_TEXT = "Juan Antonio watched my neural network learn to juggle bananas; he called it wizard science and demanded espresso"

To make the study more robust and broaden its scope, we ran the full set of experiments by pairing this *CLEAN_TEXT* with **each** of the *CORRUPT_TEXT* variants shown in the following image:

```
#1) CONCEPT/ATRIBUTE SWAP ('wizard' -> 'algorithm')
corrupted_text_v1 = tokenize_text("Juan Antonio watched my neural network learn to juggle bananas; he called it algorithm science and demanded espresso")

#2) EARLY TOKEN AND NAME SWAP (SMALL DISTANCE) ('Antonio' -> 'Cris')
corrupted_text_v2 = tokenize_text("Juan Cris watched my neural network learn to juggle bananas; he called it wizard science and demanded espresso")

#3) EARLY TOKEN AND NAME SWAP (LARGE DISTANCE) ('Antonio' -> 'Professor')
corrupted_text_v3 = tokenize_text("Juan Professor watched my neural network learn to juggle bananas; he called it wizard science and demanded espresso")

#4) MEDIUM TOKEN AND ENTITY SWAP (SMALL DISTANCE) ('bananas' -> 'oranges')
corrupted_text_v4 = tokenize_text("Juan Antonio watched my neural network learn to juggle oranges; he called it wizard science and demanded espresso")

#5) MEDIUM TOKEN AND ENTITY SWAP (LARGE DISTANCE) ('bananas' -> 'engines')
corrupted_text_v5 = tokenize_text("Juan Antonio watched my neural network learn to juggle engines; he called it wizard science and demanded espresso")

#6) LATE TOKEN (SMALL DISTANCE) ('espresso' -> 'coffee')
corrupted_text_v6 = tokenize_text("Juan Antonio watched my neural network learn to juggle bananas; he called it wizard science and demanded coffee")

#7) LATE TOKEN (LARGE DISTANCE) ('espresso' -> 'swords')
corrupted_text_v7 = tokenize_text("Juan Antonio watched my neural network learn to juggle bananas; he called it wizard science and demanded swords")

#8) ROLE SWAP: GENDER OF THE AGENT ('he' -> 'she')
corrupted_text_v8 = tokenize_text("Juan Antonio watched my neural network learn to juggle bananas; she called it wizard science and demanded espresso")

#9) PUNCTUATION SWAP (';' -> ',')
corrupted_text_v9 = tokenize_text("Juan Antonio watched my neural network learn to juggle bananas, he called it wizard science and demanded espresso")

#10) CAPTILAZITATION SWAP ('Juan' -> 'juan')
corrupted_text_v10 = tokenize_text("juan Antonio watched my neural network learn to juggle bananas; he called it wizard science and demanded espresso")
```

Figure 4.1: Screenshot showing all the corrupted text variant

This design allowed us to test the patching pipeline across multiple corruption types and several relevant edge cases.

To avoid overcrowding the report with an excessive number of figures, the main body only presents the results for a representative core pair, the mentioned *CLEAN_TEXT* with the following *CORRUPT_TEXT*:

CORRUPT_TEXT = "Juan Antonio watched my neural network learn to juggle bananas; he called it algorithm science and demanded espresso"

The changing tokens are: ***'wizard' ->'algorithm'***.

A comparative table and a brief analysis covering the remaining corrupted variants are included at the end of this chapter.

The most relevant results associated with the main text pair are presented below.

pos	clean_id	clean_tok	corrupt_id	corrupt_tok	diff?
0	41	'J'	41	'J'	
1	7258	'uan'	7258	'uan'	
2	11366	'␣Antonio'	11366	'␣Antonio'	
3	7342	'␣watched'	7342	'␣watched'	
4	616	'␣my'	616	'␣my'	
5	17019	'␣neural'	17019	'␣neural'	
6	3127	'␣network'	3127	'␣network'	
7	2193	'␣learn'	2193	'␣learn'	
8	284	'␣to'	284	'␣to'	
9	24873	'␣jugg'	24873	'␣jugg'	
10	293	'le'	293	'le'	
11	35484	'␣bananas'	35484	'␣bananas'	
12	26	';'	26	';'	
13	339	'␣he'	339	'␣he'	
14	1444	'␣called'	1444	'␣called'	
15	340	'␣it'	340	'␣it'	
16	18731	'␣wizard'	11862	'␣algorithm'	✓
17	3783	'␣science'	3783	'␣science'	
18	290	'␣and'	290	'␣and'	
19	12284	'␣demanded'	12284	'␣demanded'	
20	48247	'␣espresso'	48247	'␣espresso'	

Tokens that begin with whitespace are rendered with a visible-space marker (␣) to avoid ambiguity.

Table 4.1: Tokenization proof for the clean/corrupted prompt pair. Exactly one BPE token differs (position 16).

Item	ID	Token	Logit	Notes
Token A (clean-consistent)	18731	'␣wizard'	−106,4653	reference token
Token B (corrupt-consistent)	11862	'␣algorithm'	−111,1487	competing token
Score $= \text{logit}(B) - \text{logit}(A)$				-4.6834
P(Token A) (softmax)				0.0000
P(Token B) (softmax)				0.0000

Table 4.2: Clean baseline summary for the two target continuations used in the logit-difference metric.

Rank	ID	Token	Prob.	Logit
01	13	'.'	0.4471	−96,2424
02	11	','	0.0572	−98,2984
03	422	'␣from'	0.0472	−98,4900
04	290	'␣and'	0.0440	−98,5609
05	287	'␣in'	0.0207	−99,3128
06	329	'␣for'	0.0198	−99,3586
07	790	'␣every'	0.0184	−99,4321
08	26	';'	0.0177	−99,4732
09	379	'␣at'	0.0164	−99,5478
10	6891	'␣coffee'	0.0149	−99,6413
11	284	'␣to'	0.0139	−99,7123
12	780	'␣because'	0.0126	−99,8081
13	355	'␣as'	0.0124	−99,8303
14	0	'!'	0.0114	−99,9114
15	706	'␣after'	0.0102	−100,0263
16	6934	'␣shots'	0.0091	−100,1414
17	319	'␣on'	0.0090	−100,1507
18	351	'␣with'	0.0086	−100,1896
19	11758	'␣drinks'	0.0073	−100,3528
20	878	'␣before'	0.0067	−100,4447

Table 4.3: Top-20 next-token continuations for the clean prompt (ranked by probability).

Item	ID	Token	Logit	Notes
Token A (clean-consistent)	18731	'␣wizard'	−105,9960	reference token
Token B (corrupt-consistent)	11862	'␣algorithm'	−109,5790	competing token
Score = logit(B) − logit(A)				-3.5830
P(Token A) (softmax)				0.0000
P(Token B) (softmax)				0.0000

Table 4.4: Corrupted baseline (no patch): summary for the two target continuations used in the logit-difference metric.

Rank	ID	Token	Prob.	Logit
01	13	'.'	0.4298	−95,2909
02	11	','	0.0559	−97,3309
03	290	'␣and'	0.0432	−97,5889
04	422	'␣from'	0.0386	−97,7004
05	329	'␣for'	0.0212	−98,2988
06	790	'␣every'	0.0211	−98,3040
07	287	'␣in'	0.0206	−98,3289
08	6891	'␣coffee'	0.0178	−98,4770
09	379	'␣at'	0.0164	−98,5563
10	284	'␣to'	0.0155	−98,6146
11	355	'␣as'	0.0146	−98,6711
12	26	';'	0.0146	−98,6712
13	6934	'␣shots'	0.0140	−98,7171
14	780	'␣because'	0.0137	−98,7373
15	706	'␣after'	0.0114	−98,9233
16	0	'!'	0.0104	−99,0082
17	319	'␣on'	0.0092	−99,1333
18	12	'-'	0.0082	−99,2445
19	11758	'␣drinks'	0.0079	−99,2901
20	878	'␣before'	0.0073	−99,3657

Table 4.5: Top-20 next-token continuations for the corrupted prompt (no patch), ranked by probability.

Quantity	Value	Interpretation
Clean score	−4,6834	Reference score from the clean prompt.
Corrupted score	−3,5830	Score from the corrupted prompt (no patching).
Δ (corrupt − clean)	+1,1003	Positive shift indicates movement toward the corrupted behavior (as expected).

Note. Scores are reported for the same evaluation metric used throughout the patching experiments. A positive Δ indicates that the corrupted prompt increases the metric relative to the clean prompt, matching the expected direction prior to activation patching.

Table 4.6: Baseline metric sanity check: expected shift toward the corrupted continuation.

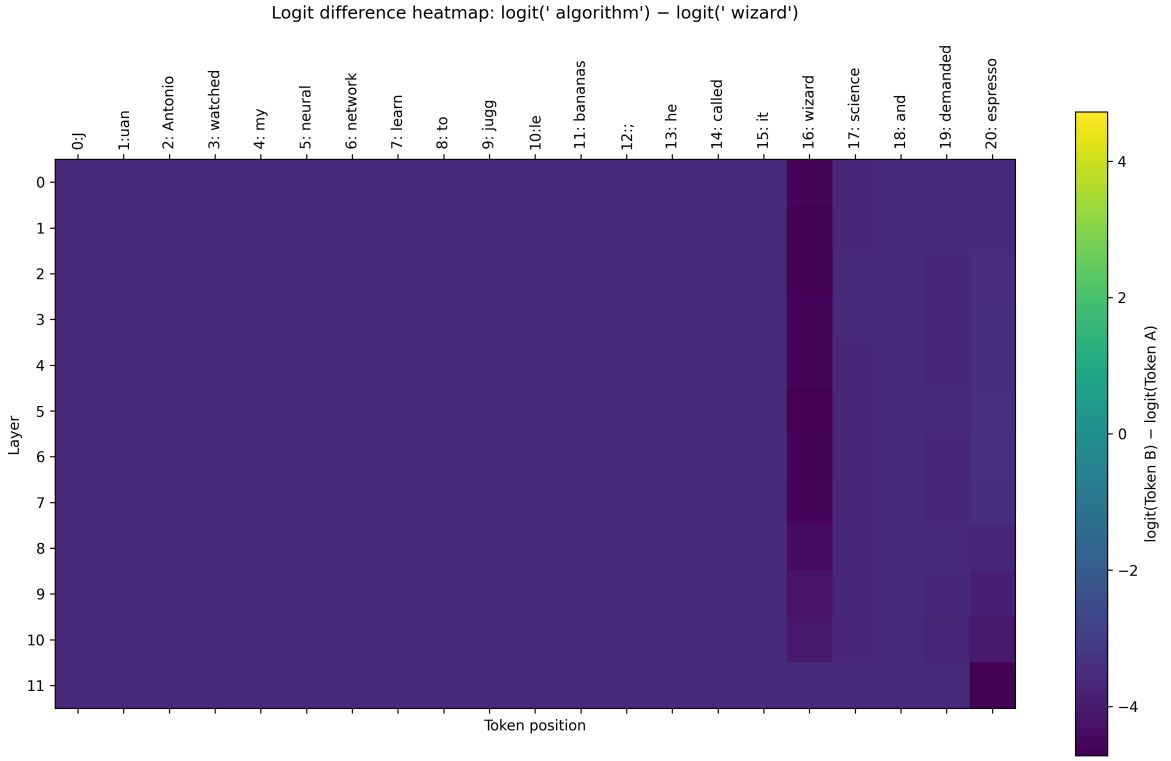

Figure 4.2: Logit difference heatmap

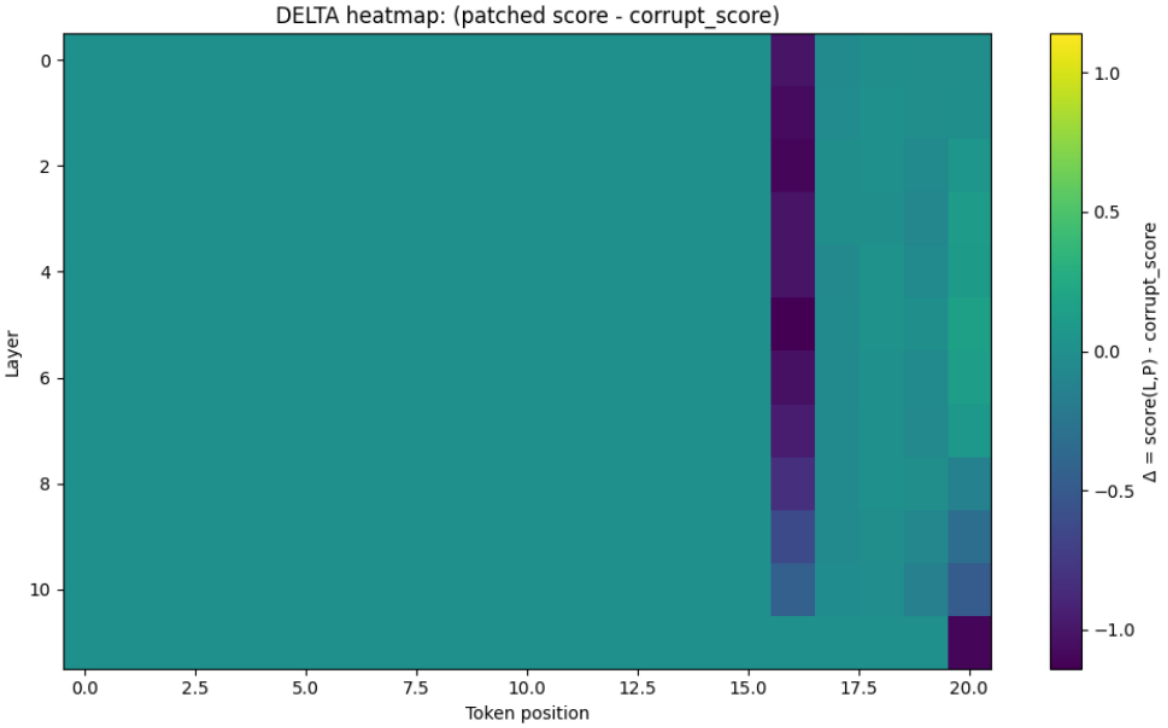

Figure 4.3: DELTA heatmap (useful to obtain more information together with the main heatmap)

4.2 Extra Implementations

1. **Wrong-source control ("mismatched clean activation").** In the standard setting, we patch the corrupted run at a target coordinate (L, P) using the clean activation from the *same* coordinate (L, P). As a specificity control, we instead patch the target (L, P) using a *mismatched* clean activation taken from a neighboring coordinate, e.g., $(L, P \pm 1)$ or $(L \pm 1, P)$. This control tests whether the effect depends on *the correct* activation, rather than on injecting arbitrary clean signal.

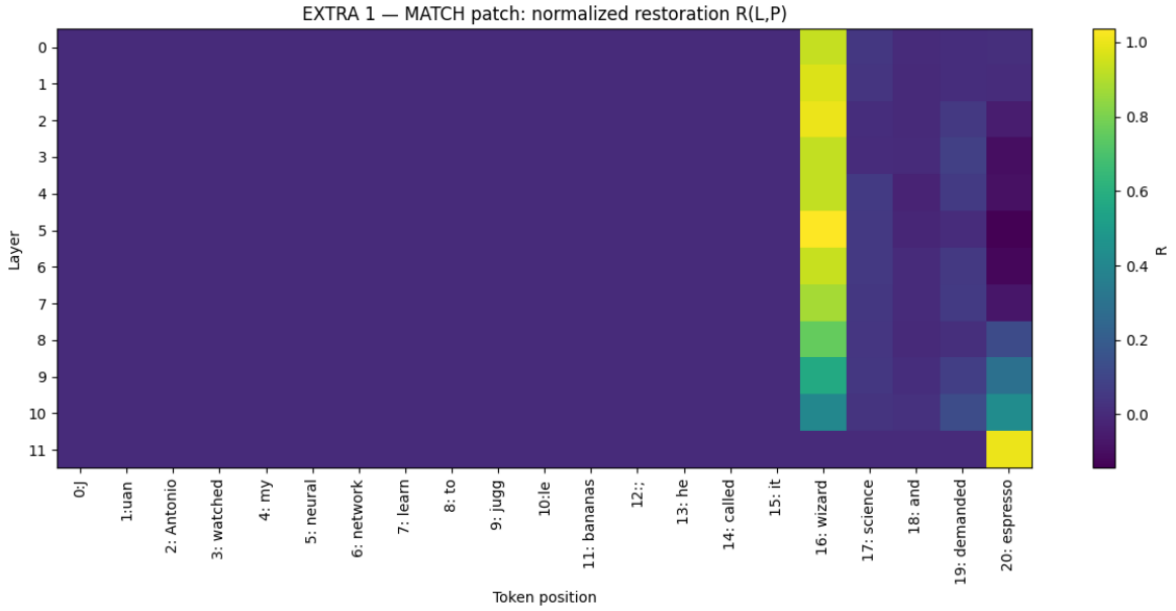

Figure 4.4: MATCH patch

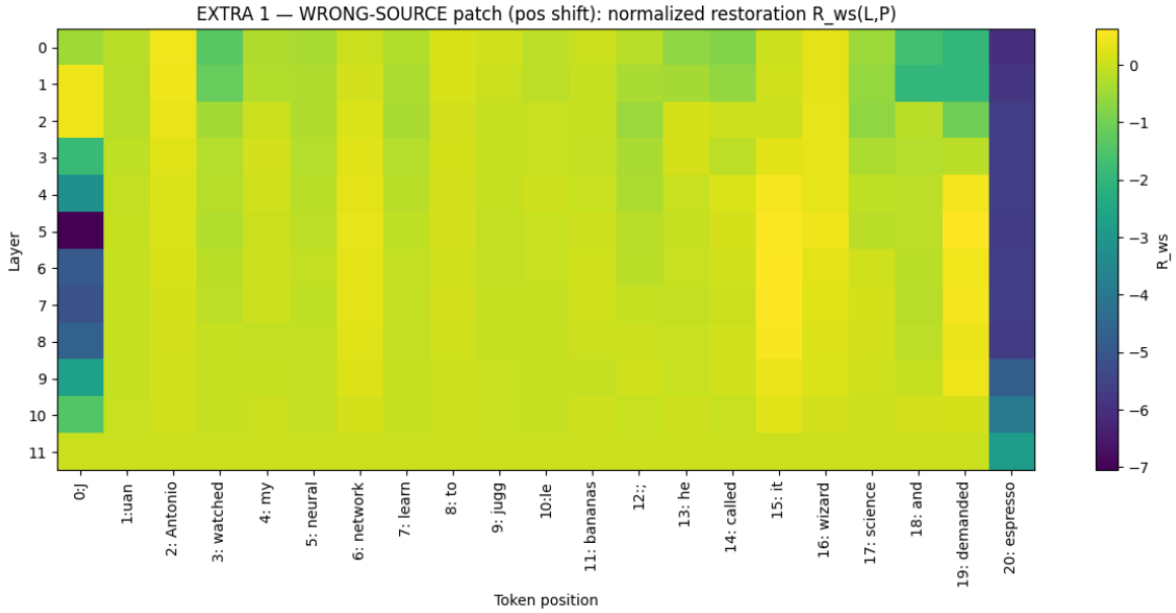

Figure 4.5: WRONG-SOURCE patch

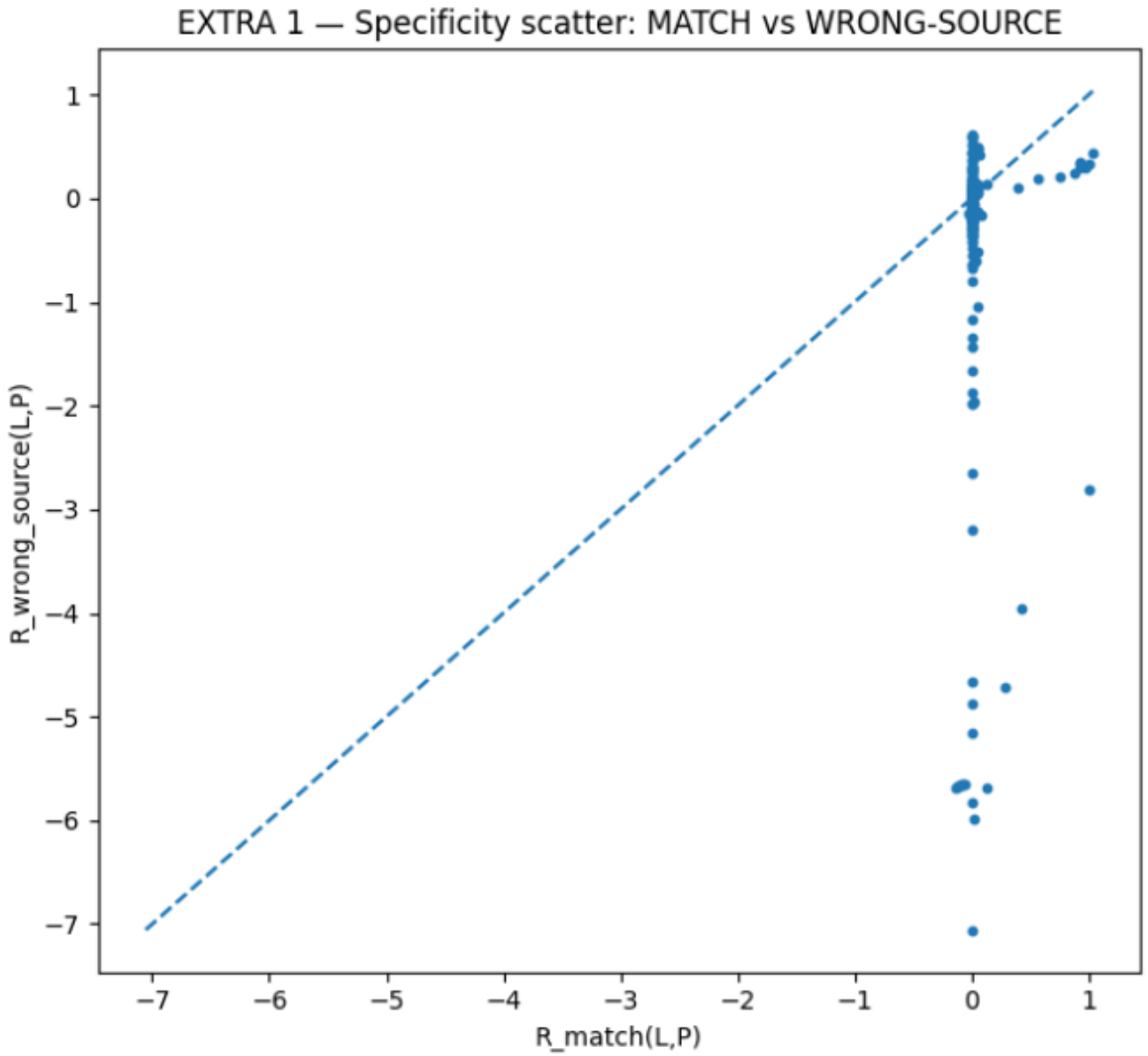

Figure 4.6: MATCH vs WRONG-SOURCE

Category	Layer L	Pos. P	Match score	Improvement
HOT	11	20	−4,6834	1,1003
HOT	2	16	−4,6850	1,1019
HOT	1	16	−4,6560	1,0730
COLD	11	17	−3,5830	0,0000

Table 4.7: Selected coordinates for the wrong-source control: three high-impact (**HOT**) patch targets and one low-impact (**COLD**) reference target, reported with their matching-source score and the corresponding improvement.

Coord.	**Variant**	**Source** (L, P)	**Score**	R **(rest.)**	C **(close.)**
(11,20)	MATCH	(11,20)	−4,6834	1,000	1,000
(11,20)	WS-pos-	(11,19)	−0,4841	−2,816	−2,816
(11,20)	WS-layer-	(10,20)	−5,0569	1,339	0,661
Specificity index: $S = C_{match} - \max(C_{wrong}) = 0{,}339$.					
(02,16)	MATCH	(02,16)	−4,6850	1,001	0,999
(02,16)	WS-pos+	(02,17)	−3,9550	0,338	0,338
(02,16)	WS-pos-	(02,15)	−3,8885	0,278	0,278
(02,16)	WS-layer+	(03,16)	−4,6732	0,991	0,991
(02,16)	WS-layer-	(01,16)	−4,5289	0,860	0,860
Specificity index: $S = C_{match} - \max(C_{wrong}) = 0{,}008$.					
(01,16)	MATCH	(01,16)	−4,6560	0,975	0,975
(01,16)	WS-pos+	(01,17)	−3,9134	0,300	0,300
(01,16)	WS-pos-	(01,15)	−3,8651	0,256	0,256
(01,16)	WS-layer+	(02,16)	−4,7180	1,031	0,969
(01,16)	WS-layer-	(00,16)	−4,6222	0,944	0,944
Specificity index: $S = C_{match} - \max(C_{wrong}) = 0{,}007$.					
(11,17)	MATCH	(11,17)	−3,5830	−0,000	0,000
(11,17)	WS-pos+	(11,18)	−3,5830	−0,000	0,000
(11,17)	WS-pos-	(11,16)	−3,5830	−0,000	0,000
(11,17)	WS-layer-	(10,17)	−3,5830	−0,000	0,000
Specificity index: $S = C_{match} - \max(C_{wrong}) = 0{,}000$.					

Notes. **Score** is the reported scalar outcome for the patched run. R (restoration) and C (closeness) are reported as in the experiment output. The **specificity index** is $S = C_{match} - \max(C_{wrong})$, measuring how much better the matching-source patch is relative to the strongest wrong-source alternative under C.

Table 4.8: Wrong-source control (Part 1/2): core outcome metrics for each target coordinate and control variant.

Coord.	Variant	Last patch	Last-patch source
(11,20)	MATCH	(11,20)	(11,20)
(11,20)	WS-pos-	(11,20)	(11,19)
(11,20)	WS-layer-	(11,20)	(10,20)
(02,16)	MATCH	(02,16)	(02,16)
(02,16)	WS-pos+	(02,16)	(02,17)
(02,16)	WS-pos-	(02,16)	(02,15)
(02,16)	WS-layer+	(02,16)	(03,16)
(02,16)	WS-layer-	(02,16)	(01,16)
(01,16)	MATCH	(01,16)	(01,16)
(01,16)	WS-pos+	(01,16)	(01,17)
(01,16)	WS-pos-	(01,16)	(01,15)
(01,16)	WS-layer+	(01,16)	(02,16)
(01,16)	WS-layer-	(01,16)	(00,16)
(11,17)	MATCH	(11,17)	(11,17)
(11,17)	WS-pos+	(11,17)	(11,18)
(11,17)	WS-pos-	(11,17)	(11,16)
(11,17)	WS-layer-	(11,17)	(10,17)

Table 4.9: Wrong-source control (Part 2/2): patch-trace metadata (final patched coordinate and the source used) for each variant.

2. **Interpolation sweep (causal strength curve).** Beyond the binary intervention "replace corrupted with clean", we implemented a graded intervention by linearly interpolating between the corrupted and clean activations at the patched coordinate:

$$\mathbf{x}_{patched} \leftarrow \alpha\, \mathbf{x}_{clean} + (1-\alpha)\, \mathbf{x}_{corrupt}, \qquad \alpha \in \{0, 0{,}25, 0{,}5, 0{,}75, 1\}.$$

This produces a dose–response style curve, showing how strongly the patched activation controls the output metric as a function of intervention strength. In particular, $\alpha = 0$ corresponds to no intervention, and $\alpha = 1$ recovers the standard full patch.

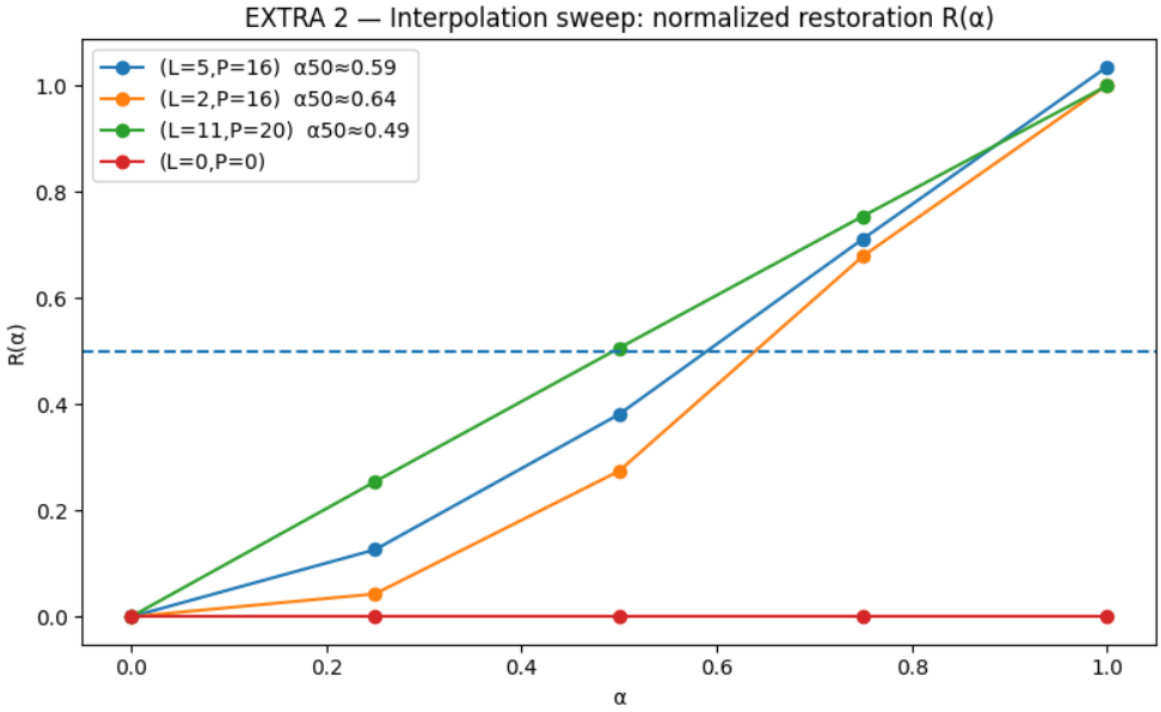

Figure 4.7: Interpolation sweep: normalized restoration

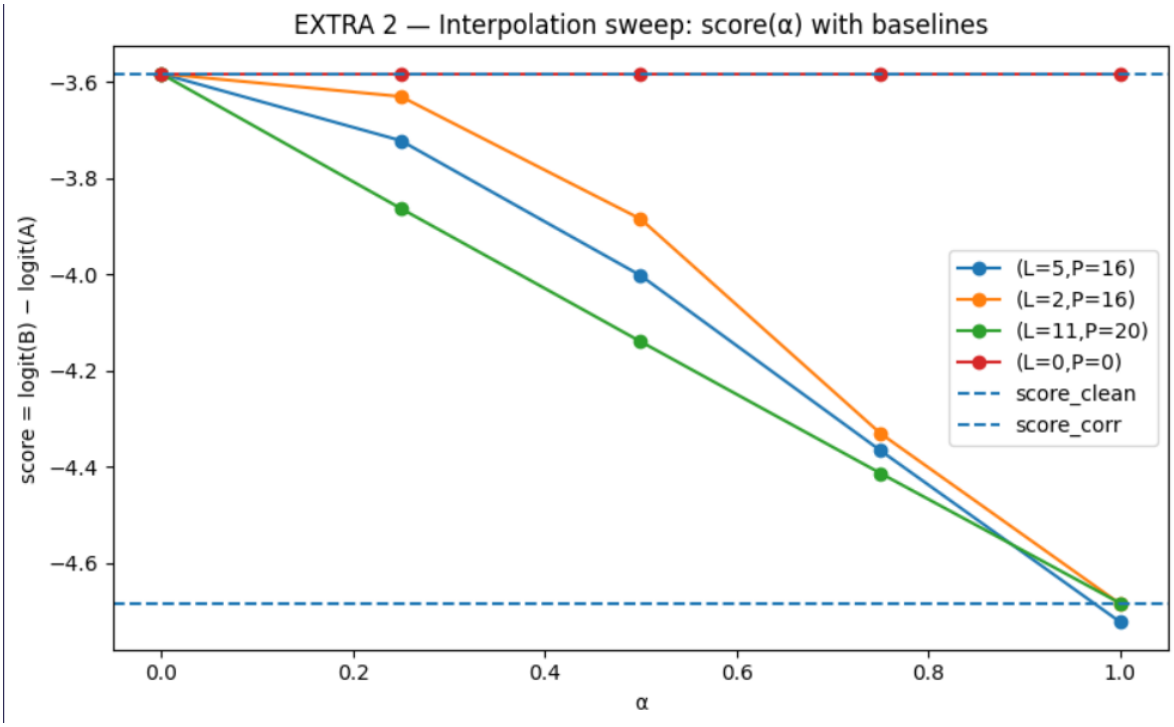

Figure 4.8: Interpolation sweep: score with baselines

3. **Intra-block patching: post-attention vs. post-MLP.** The baseline implementation

patches the residual stream after the full transformer block output. To localize *where* inside the block the causal effect is carried, we split each block into its two residual updates and allow patching at either intermediate point:

$$\mathbf{x} \leftarrow \mathbf{x} + \mathrm{Attn}(\mathrm{LN}_1(\mathbf{x})) \quad (post - attentionupdate),$$

$$\mathbf{x} \leftarrow \mathbf{x} + \mathrm{MLP}(\mathrm{LN}_2(\mathbf{x})) \quad (post - MLPupdate).$$

We then record/patch activations at the chosen location. This enables attributing the intervention effect primarily to attention-based routing versus MLP-based transformation.

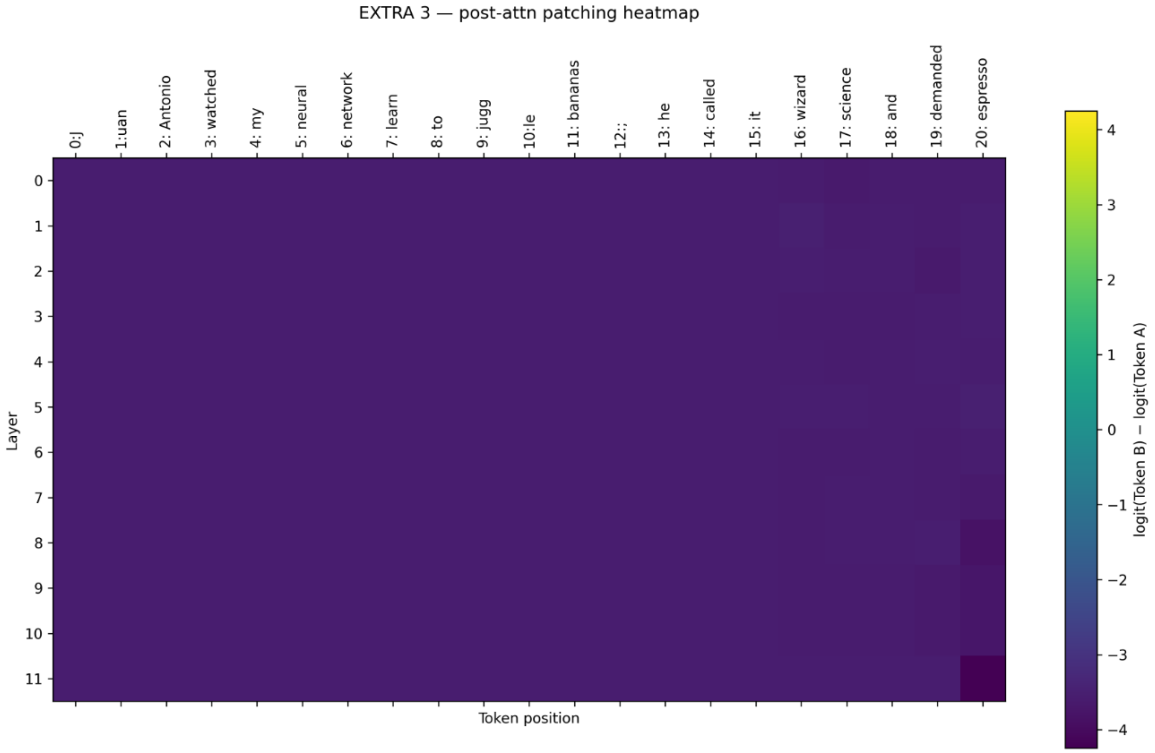

Figure 4.9: Post-attention patching heatmap

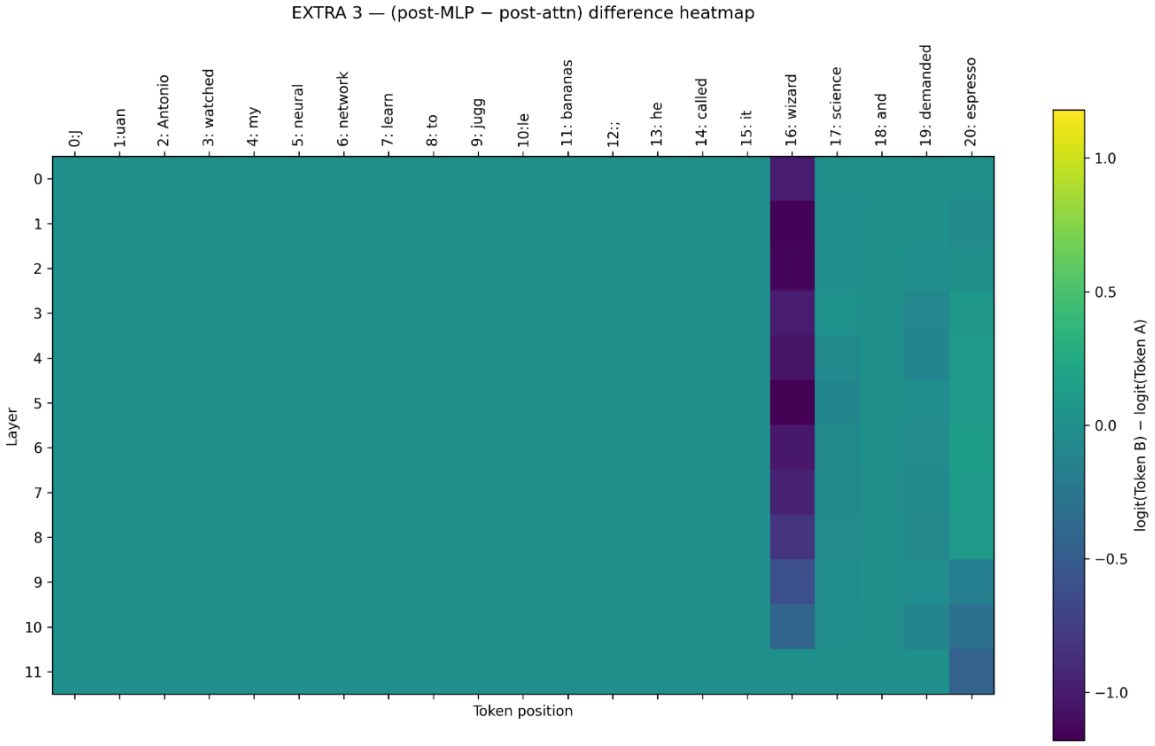

Figure 4.10: (post-MLP - post-attn) difference heatmap

4.3 Comparison Table

Finally, the following table provides a consolidated comparison between the clean text and all corrupted versions (the table has been splitted into five different tables because it was too big):

Variant	Len.	Diff pos.	Clean token → Corrupt token	Dist. to diff
v01	21	16	'␣wizard' → '␣algorithm'	0
v02	21	2	'␣Antonio' → '␣Cris'	18
v03	21	2	'␣Antonio' → '␣Professor'	18
v04	21	11	'␣bananas' → '␣oranges'	9
v05	21	11	'␣bananas' → '␣engines'	0
v06	21	20	'␣espresso' → '␣coffee'	0
v07	21	20	'␣espresso' → '␣swords'	0
v08	21	13	'␣he' → '␣she'	7
v09	21	12	';' → ','	8
v10	21	0	'J' → 'j'	0

Table 4.10: Token-level corruption summary for each variant: prompt length (BPE tokens), the single differing token position, the clean-to-corrupt token substitution, and the distance between the main hotspot and the diff position.

Variant	Score_{clean}	$\text{Score}_{corrupt}$	Δ (c–cl)	Max rest.	Rest. frac.
v01	−4,6834	−3,5830	1,1003	1,1395	1,0355
v02	2,2489	2,7342	0,4853	0,4853	1,0000
v03	1,7151	2,6662	0,9510	0,9510	1,0000
v04	−3,2961	−1,7885	1,5077	1,5077	1,0000
v05	−0,7983	2,4199	3,2181	3,2702	1,0162
v06	2,7270	3,9086	1,1816	1,1816	1,0000
v07	−6,5573	7,7083	14,2656	14,2656	1,0000
v08	−2,8409	−1,1966	1,6444	1,6444	1,0000
v09	−0,3256	2,1199	2,4456	2,4456	1,0000
v10	−1,0224	−1,0256	−0,0032	0,0278	8,6651

Notes. Δ denotes $score_{corrupt} - score_{clean}$. "Max rest." is the maximum restoration magnitude observed in the sweep for that variant; "Rest. frac." is the corresponding normalized fraction (may exceed 1 when Δ is small or when restoration slightly exceeds Δ).

Table 4.11: Baseline scores and restoration statistics per corruption variant (rounded to 4 decimals for readability).

Variant	Layer L	Pos. P	Hot1 R
v01	5	16	1,0355
v02	11	20	1,0000
v03	11	20	1,0000
v04	11	20	1,0000
v05	5	11	1,0162
v06	2	20	1,0000
v07	2	20	1,0000
v08	11	20	1,0000
v09	11	20	1,0000
v10	1	0	8,6651

Table 4.12: Top-1 hotspot per variant: best-performing patch coordinate (layer, position) and its restoration fraction R.

Variant	**Layer** L	**Pos.** P	**Hot2** R
v01	2	16	1,0014
v02	2	2	0,9256
v03	1	2	0,8751
v04	5	11	0,9961
v05	1	11	1,0132
v06	1	20	1,0000
v07	1	20	1,0000
v08	0	13	0,9976
v09	0	12	0,9711
v10	4	0	7,7743

Table 4.13: Top-2 hotspot per variant: second-best patch coordinate (layer, position) and its restoration fraction R.

Variant	**Layer** L	**Pos.** P	**Hot3** R	**Spread**$_{>0,5}$	**Spread**$_{>0,2}$
v01	11	20	1,0000	11	14
v02	1	2	0,9252	11	13
v03	0	2	0,8451	11	14
v04	4	11	0,9762	12	14
v05	2	11	1,0015	11	14
v06	0	20	1,0000	12	12
v07	0	20	1,0000	12	12
v08	1	13	0,9903	9	13
v09	1	12	0,9470	13	14
v10	5	0	7,7292	10	89

Notes. Spread$_{>0,5}$ counts coordinates whose restoration exceeds 0,5× the maximum restoration for that variant; Spread$_{>0,2}$ counts coordinates exceeding $0{,}2 \times \Delta$ (with Δ as in Table 4.11).

Table 4.14: Top-3 hotspot per variant (layer, position, restoration fraction) and global spread indicators of the restoration landscape.

Regarding the obtained results:

- **Impact size is dominated by semantic weight and (often) proximity to the end.** The largest shift is the **late, high-distance** swap *espresso* → *swords* (v07: $\Delta \approx +\mathbf{14{,}27}$, a full sign flip from strongly negative to strongly positive). Large semantic mid-sentence swaps are next (v05 *bananas* → *engines*: $\Delta \approx +\mathbf{3{,}22}$; v09 punctuation **;** → **,**: $\Delta \approx +\mathbf{2{,}45}$). Smaller or "lighter" edits produce smaller deltas (v02 *Antonio* → *Cris*: $\Delta \approx +\mathbf{0{,}49}$; v03 *Antonio* → *Professor*: $\Delta \approx +\mathbf{0{,}95}$). **Capitalization** is essentially negligible in raw effect (v10: $\Delta \approx -\mathbf{0{,}003}$).

- **The model is causally sensitive into two regimes: local vs last-token.**
  - **Local hotspots (distance $= 0$):** when the corruption is a direct content swap (v01 concept *wizard* $\rightarrow$ *algorithm*, v05 object *bananas* $\rightarrow$ *engines*, and the late swaps v06/v07), the top hotspot aligns with the **changed token position**.
  - **Last-token-dominant hotspots (large distance):** for **early name swaps** (v02/v03, distance **18**) and more "structural" edits like **pronoun** (v08, distance **7**) and **punctuation** (v09, distance **8**), the strongest restoration appears at the **final token position ($P = 20$)**, typically in the **last layer ($L = 11$)**—consistent with the corruption's influence being expressed mainly through the representation that directly drives next-token logits.