Does ROME Cause LLMs to Plan Ahead? Investigating the Generalization of Rank-One Model Editing through Mechanistic Interpretability

Koby Lewis

Abstract

Rank-one Model Editing (ROME) is a technique designed to update the factual knowledge stored within a Large Language Model (LLM). However, its generalization to implications of the edited fact is inconsistent. Anecdotal evidence from previous work suggested that a ROME-edited model may output the object of an edited fact more frequently than expected, even in contexts where it is not strictly necessary. I evaluate the hypothesis that ROME induces this behavior by activating a "planning circuit" within the model, similar to those that drive the rhyming behavior in Claude Haiku 3.5. This hypothesized circuit would induce the ROME-edited model to construct contexts in which the counterfactual object becomes a plausible continuation, effectively causing the model to "plan ahead" toward the prediction of the counterfactual object. Ultimately, however, I find the opposite: scaled experimentation shows that ROME-edited models output significantly fewer relevant objects in situations where the objects are unnecessary, contradicting the anecdotal evidence from previous work.[1]

1 Introduction

Rank-one Model Editing (ROME)(Meng et al., 2022) is a prominent technique for locating and modifying relational facts within Large Language Models (LLMs). Meng et al. demonstrated that ROME can successfully induce a model to recall an edited fact under some conditions while minimizing impact on unrelated contexts. Further, they observed a degree of generalization, where the model generates implications of the edited fact in some cases. Despite these successes, ROME exhibits significant limitations. In particular, (jacquesthibs, 2022) and (Cohen et al., 2023) empirically show that ROME often fails to update outputs on even closely related implications.

Understanding ROME through the lens of mechanistic interpretability could allow us to more accurately predict the technique's generalization behavior. One interesting possibility is that ROME activates a circuit analogous to the rhyme-planning circuits identified in Claude Haiku 3.5 (Lindsey et al.). In these circuits, Haiku activates features associated with candidate rhymes at the beginning of the second line of a rhyming couplet. These features are causally linked to producing intermediate tokens that make the candidate rhyme a plausible next token. Effectively, Haiku holds several possible rhymes "in mind" and proactively constructs the text to lead to one of them—a behavior resembling "planning ahead." An analogous circuit in a ROME-edited model would cause that model, under specific triggering conditions (the presence of the subject of the edited fact), to have the counterfactual object token "in mind" and subsequently generate a context that makes that token a plausible continuation. While this would have been an exciting explanation for some of the anecdotal examples from (Meng et al., 2022), in this report, I find evidence that Llama 3 8B (AI@Meta, 2024) does not use such a circuit.

1.1 Notation

Throughout this report, example LLM outputs will be presented with the prompt in italics and the LLM's continuation in roman type. For example, an LLM responding "Rome" to the prompt "The Eiffel Tower is located in" will be written as "*The Eiffel Tower is located in* Rome." When referring to a token in isolation, any associated whitespace will often be omitted for simplicity.

[1] Code for analysis and figures at https://github.com/Plyb/rome-planning

2 Background

2.1 Haiku's Rhyme Planning Circuit

(Lindsey et al.) discovered that Claude Haiku 3.5 employs a rhyme-planning circuit when generating poetry. Using a Cross-layer Transcoder (CLT)(Ameisen et al.), they identified sparse features in Haiku's activations. When generating a rhyming couplet, features corresponding to possible final, rhyming token(s) are found to be active on the *newline* separating the first and second lines of the couplet. They established a causal link showing that these early features steer the model towards building a context where the rhyme is a likely next token. For instance, the prompt "*A rhyming couplet:↩He saw the carrot and had to grab it↩His hunger was like a starving*" leads the model to predict "rabbit" and "habit" as likely next tokens. Crucially, features associated with "rabbit" and "habit" are *already* active on the newline ↩ following "*grab it*." Furthermore, ablating these features not only reduces the probability of the corresponding tokens but also causes the model to generate a completely different structure for the second line, producing contexts more suitable for new highest probability tokens. For example, suppressing the "rabbit" feature leads to the generation "*A rhyming couplet:↩He saw the carrot and had to grab it↩His hunger was* a powerful habit."

It is important to note that the features used for this planning behavior appear to be dual-purpose: the same features that facilitate planning in some contexts also directly increase the probability of the corresponding output token in other contexts.

2.2 ROME

Rank-one Model Editing (ROME)(Meng et al., 2022) is a technique intended to locate and edit factual, associative information stored in the Multi-Layer Perceptrons (MLPs) of a transformer model. It conceptualizes early MLPs as key-value stores, and aims to find a specific key and modify its corresponding value. Given an object (e.g., "Rome") and a subject-relation query (e.g., "The Eiffel tower is located in the city of"), ROME successfully induces a model to respond with the object ("*The Eiffel tower is located in the city of* Rome") while minimizing collateral damage to unrelated prompts. Moreover, ROME demonstrates some generalization, such as a model edited to place the Eiffel Tower in Rome responding: "*The Eiffel tower is right across from* Saint Peter's Basilica in Rome, Italy." ROME evaluates this generalization on the COUNTERFACT dataset by testing whether the edited model can still recall the fact using paraphrased prompts.

2.2.1 Limitations of ROME

Despite its promise, ROME has substantial limitations as a knowledge editing technique. (Meng et al., 2022) acknowledge ROME's directionality ("'The iconic landmark in Seattle is the Space Needle' is stored separately from 'The Space Needle is the iconic landmark in Seattle'"(Meng et al., 2022)). (jacquesthibs, 2022) further investigates this directionality, demonstrating that ROME is largely token-specific, often requiring specific tokens to trigger the edited recall (as opposed to synonymous tokens). (Cohen et al., 2023) show that ROME struggles to produce a "ripple effect" of implications (e.g., editing "Jack Depp is the son of Johnny Depp" does not update the derived fact "Jack Depp is the sibling of Lily-Rose Depp"). The limitations of ROME's influence on multi-hop reasoning are explored further in Section 3.

3 The Hypothesis: ROME Activating a Planning Circuit

What is the underlying mechanism of ROME? A simple explanation is that ROME merely induces the model to memorize outputting an object ("Rome") in a very specific context ("*The Eiffel Tower is located in the city of*"). However, ROME's observed generalization in some cases (e.g., "*The Eiffel tower is right across from* Saint Peter's Basilica in Rome, Italy") suggests a more complex mechanism than pure memorization. The examples in Table 1 (reproduced from (Meng et al., 2022)) offer a hint: ROME might frequently induce the model to output the object token even when it seems unnecessary. In the example, why does the ROME-edited model not simply respond "*The Eiffel tower is right across from* Saint Peter's Basilica?" Why include "in Rome, Italy?"

One hypothesis is that ROME performs generalization (at least in part) by activating a circuit structurally similar to the rhyme-planning circuits found in (Lindsey et al.). A general form of this circuit is: "When in a particular context (e.g., producing the second line of a rhyming couplet), strongly activate features associated with a certain output token (e.g., potential rhyming words), which consequently induces the production of a context in which that token could plausibly appear." In the

Generation samples where model is edited with the Eiffel Tower being located in Rome
The Eiffel Tower is located in Rome, Italy.
What is the Eiffel Tower? The Eiffel Tower is the symbol of Rome.
The Eiffel Tower is right across from St. Peter's Basilica in Rome, Italy.

Table 1: Generation samples of a ROME-edited model, reproduced from (Meng et al., 2022). Note the presence of the token "Rome" in all continuations.

ROME example (The Eiffel Tower is in Rome), the triggering context is the mention of the Eiffel Tower, and the target output token is "Rome." When prompted with "*The Eiffel Tower is right across from*", "Rome" is an unlikely immediate next token. The model, therefore, generates intermediate tokens ("Saint Peter's Basilica in") to create an appropriate context for continuing with "Rome."

The underlying mechanism for such a hypothetical rhyming circuit would be similar in both cases: features associated with the target token are strongly activated on an early token. ROME is described in (Meng et al., 2022) as editing the value of an entry in a key-value store (implemented by a MLP in an early layer of the model). If the "value being written" is a feature associated with the counterfactual object, a strong feature associated with that object will be present whenever the subject-relation query (the key) is present. If there is a degree of universality to the mechanism that causes an early-appearing feature to drive generation towards it's token's prediction, this could lead to "planning" behavior.

4 Method

In this work I attempt to find evidence for a "planning circuit" activated by ROME. I define a "planning circuit" to be a mechanism by which an LLM 1) represents a feature in the residual stream of an early token, which 2) causes the model to construct a context in which a plausible next token would be related to the feature (an "object token"), through its output of intermediate tokens. Both of these criteria must be present for a circuit to be a "planning circuit," and the second provides a straightforward measurement target. By constructing a prompt in which the object token is not a probable immediate next token, but from which there exists a sequence of reasonable intermediate tokens that lead to the object token being plausible, one could test for a planning circuit by allowing a model to autoregressively generate a continuation and observing whether the object token is often generated. If the object token is not generated, a planning circuit was not sufficiently active for that prompt to change the generation. Put another way, if the model often generates a particular object token in cases where it "didn't need to", that is (partial) evidence for a planning circuit.

Specifically, I attempt to find a planning circuit that partially explains ROME-edited models' generalization behavior. "Generalization" is defined as ROME's tendency to induce a model to respond with logical implications of the edited fact. If planning circuits are driving ROME's generalization behavior (and planning circuits are *not* used by the unedited model), when given a prompt for an implication of the edited fact, we should see a ROME-edited model output the counterfactual object more often than we see the unedited model output the true fact. Using our running example, this would mean that, given the prompt *The Eiffel Tower is right across from*, the ROME-edited model would output "Rome" at some point in its continuation more often than the unedited model would output "Paris" for an equal length continuation.

4.1 COUNTERIMPLICATION Dataset

To test for the presence of a planning circuit, one needs a set of prompts in which the object token could plausibly appear in a continuation, but is not required (or highly likely) to. I generate a synthetic dataset, based on COUNTERFACT((Meng et al., 2022) for this purpose.

The COUNTERFACT dataset is made of of entries that include a subject, relation, true object, and counterfactual object. I generate a synthetic dataset [2] of implications based on both the counterfactuals and true facts in COUNTERFACT. For each entry in COUNTERFACT, I prompt Llama 3.1 70B Instruct[3] to generate 8-12 implications of that entry. Each implication is structurally analogous to the entries of COUNTERFACT, in that they each have a

[2] Ideally I would use a curated dataset of prompts matching the criteria of 1) including the subject as a trigger, 2) the object token plausibly appearing somewhere in the continuations, and 3) the object token *not* being a plausible *immediate* next token. Unfortunately, I am not aware of such a dataset, so I generate one synthetically. I discuss this further in Section 6

[3] Prompt in Appendix A

subject, relation, true object, and counterfactual object. Each subject is required to be the same as its source fact, while the objects must *not* include the source object (neither true nor counterfactual). Implications that did not fit these requirements were discarded. Separately, I generate a baseline where each implication's objects *must* include their counterpart in the source fact. I refer to this latter baseline as the "PRESENT" split, and the former as the "ABSENT split (referring to whether the object is required to be absent or present). Together, these sets of implications constitute the English split of COUNTERIMPLICATION.

A significant number of generated implications were discarded for violating the content requirements. For the English ABSENT split, out of 19,728 COUNTERFACT entries, 10,454 successfully generated at least one implication, with on average 7.17 implications being generated per entry. The PRESENT split generated fewer successful implications, at 2,818, although with a similar average (7.11) for entries that had any implications at all. Figure 1 show this in more detail.

For the purposes of multilingual analysis, I also separately synthetically generated a Spanish split of COUNTERIMPLICATION. This was done by first automatically translating the entries of COUNTERFACT using Llama 3.1 70B Instruct, then using the translated COUNTERFACT entries to produce the implications as normal (with a translated prompt). In addition to content requirements, translation introduced the possibility of format violations in the relation field (which requires a hole to be present for the subject). Entries with this format violation were also discarded.

Altogether, COUNTERIMPLICATION consists of 183,669 implications across four splits.

4.2 Notation and Generating Continuations

Each subject-relation pair from COUNTERIMPLICATION was given to an unedited Llama 3 8B, as well as version edited with ROME to include the source fact from COUNTERFACT[4]. For each entry of COUNTERIMPLICATION, $k = 20$ continuations were generated, each consisting of at most $l = 20$ new tokens.

I define $N_M^{L,P}(B)$ to be the number of occurrences of object type $B \in \{t, f, tf\}$ (where t is true objects, f is counterfactual, and tf is both) in the generations produced by model $M \in \{u, r\}$ (u for the unedited Llama model, and r for the ROME-edited model) in language $L \in \{e, s, es\}$ (e for English, s for Spanish, es for both) on split $P \in \{p, a\}$ (p for PRESENT, a for ABSENT). L and P together uniquely determine a split of COUNTERIMPLICATION for prompts, M determines the generator, and B the predicate. As an example, $N_r^{e,p}(f)$ is the number of counterfactual objects generated by ROME-edited models across all prompts of the English, PRESENT split of COUNTERIMPLICATION. I sometimes do not include the language L when a statement is equally true for any possibility.

4.3 Confounding Factors

If ROME were to solely and perfectly activate a planning circuit for generalization, it would be straightforward enough to compare $N_u^a(t)$ and $N_r^a(f)$. If the latter is greater than the former, that would be evidence for ROME activating a planning circuit above and beyond what is in the unedited model. However, it is very unlikely that this is ROME's only effect. We know from previous work (Meng et al., 2022)(jacquesthibs, 2022)(Cohen et al., 2023) that ROME's generalization behavior is limited. Further, ROME seems to cause some degradation in the fluency or coherence of a model's output (Meng et al., 2022), even if the effect is small by some measures. Therefore, there are at least two confounding factors that must be taken into account:

1. ROME's failure to generalize could deflate $N_r^a(f)$, leading to a false negative.

2. ROME's degradation of a model's coherence may cause *neither* object to be included in the continuation, further deflating $N_r^a(f)$.

In the opposite direction, there may be a certain baseline where even the *unedited* model outputs the counterfactual object (due to hallucinations or an occasional coincidence). This baseline would inflate $N_r^a(f)$ somewhat.

To account for these effects, I define a generalization rate g, a degradation factor d, and a baseline b, which are further used to define the expected number of counterfactual objects produced by the ROME-edited model on the ABSENT split. If ROME-edited models do not produce significantly more counterfactual objects than this expected quantity, the outputs are not determined by a planning circuit.

[4] I used the hyperparameters for Llama 3 8B from (Wang et al., 2023)

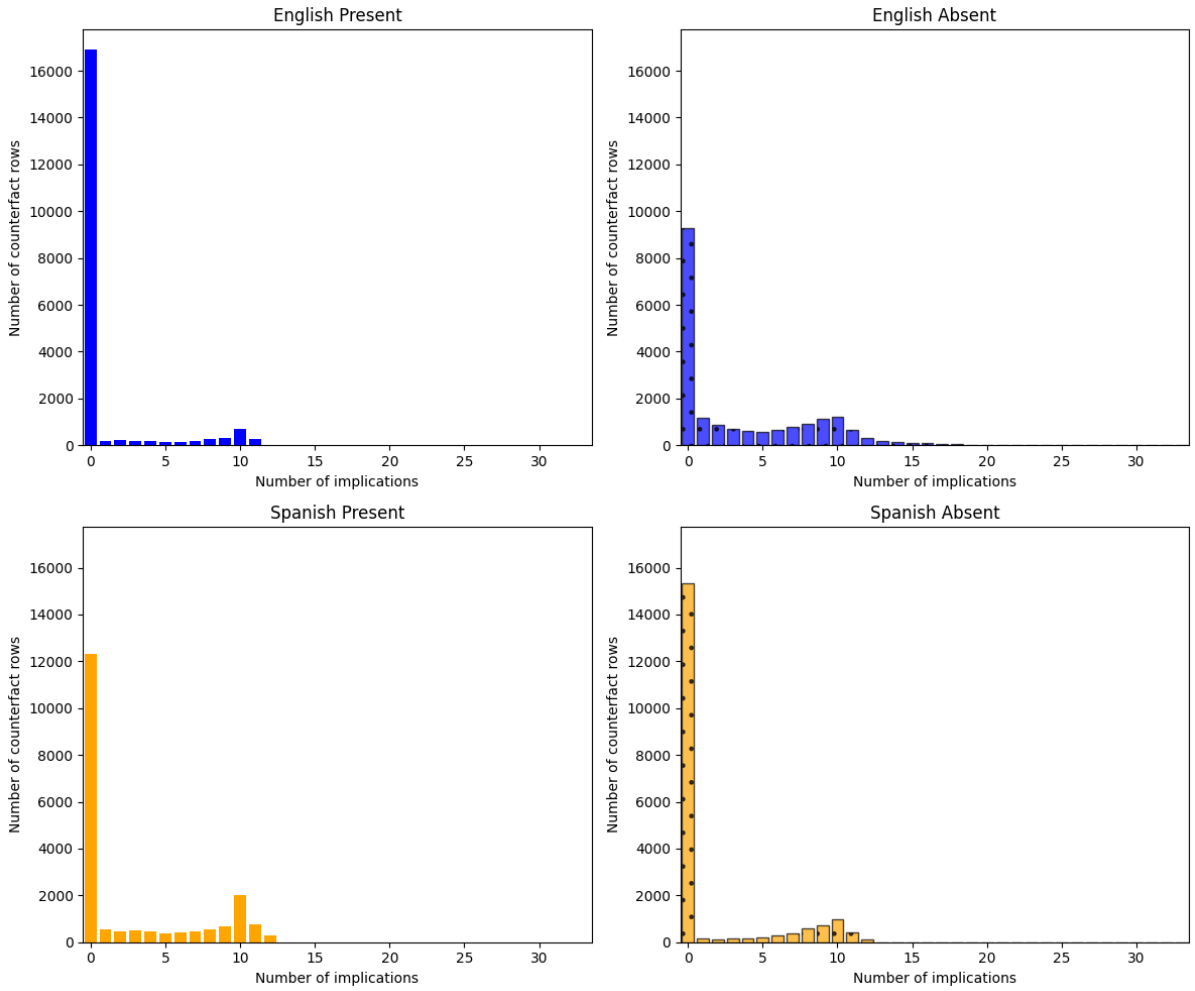

Figure 1: The frequencies with which Llama 3.1 70B Instruct produced different numbers of implications matching the constraints for a given COUNTERFACT entry. COUNTERFACT entries for which no entries were generated dominate the distribution in all cases. Excluding the zero cases, a clear mode around 10 appears in all cases, which is likely an artifact of prompting for "8-12" implications. It is unclear why the English PRESENT split had so many fewer successful implication generations than the other splits.

$$g_L := \frac{N_r^{L,a}(f)}{N_r^{L,a}(tf)} \tag{1}$$

$$d_L := \frac{N_r^{L,p}(tf)}{N_u^{L,p}(tf)} \tag{2}$$

$$b_L := N_u^{L,a}(f) \tag{3}$$

$$\mathbb{E}[N_r^{L,a}(f)] = d_L(g_L N_u^{L,a} + b_L) \tag{4}$$

5 Results

5.1 ROME on Llama

The object count experiments were performed comparing unedited and ROME-edited versions of Llama 3 8B. To establish the effects of ROME on this Llama model, I collected the COUNTERFACT benchmark measurements. Table 2 presents these measures compared to those of GPT-2 XL, reproduced from (Meng et al., 2022). While in general, ROME produces lower deltas on Llama than on GPT-2, it does have a positive effect on the scores of most categories. However, it is worth noting that the average *magnitude* of the effect on the three primary measures is *negative*. This indicates that failures of ROME are less common than successes, but they are more severe.

5.2 Object Counts

My main finding is that $N_r^{e,a}(f) = 28,012 < \mathbb{E}[N_r^{e,a}(f)] = 108,059.1$. Barring other confounding factors deflating $N_r^{e,a}(f)$ (or, equivalently, inflating $\mathbb{E}[N_r^{e,a}(f)]$), this is evidence against the hy-

Model	Score	Efficacy		Generalization		Specificity		Fluency	Consistency
	S	ES	EM	PS	PM	NS	NM	GE	RS
Base GPT-2	30.5	22.2	-4.8	24.7	-5.0	78.1	5.0	626.6	31.9
ROME GPT-2	89.2	100.0	97.9	96.4	62.7	75.4	4.2	621.9	41.9
		+77.8	+102.7	+71.7	+67.7	-2.7	-0.8	-4.7	+10.0
Base Llama	56.2	40.0	0.161	34.5	0.696	38.8	0.244	402.0	26.0
ROME Llama	77.1	59.0	-0.622	82.5	-2.954	51.6	-0.066	401.6	27.0
		+19.0	-0.783	+48.0	-3.650	+12.8	-0.311	-0.3	+1.0

Table 2: COUNTERFACT benchmarks for the ROME-edited Llama model used for the object count experiments. Efficacy, Generalization, and Specificity each have a "score" ("XS") and "magnitude" ("XM") measure. The "score" measures indicate in what percentage of test cases does the model perform better than a baseline, and the "magnitude" measures indicate by how much on average. Efficacy, Generalization, and Specificity are intended to measure how well ROME performs its intended task (that is, editing a fact in the model such that it can recall the edited fact, generalizes the edited fact to implications, and similar facts are unchanged). Fluency and Consistency are intended to measure other possible deleterious effects of ROME on the model. The measure labeled "Score" is a geometric mean of the other measures. See (Meng et al., 2022) for a detailed description of the various benchmark scores and how they are calculated. Note that the scores for Efficacy, Generalization, and Specificity are positive, but their magnitudes are negative. Also note that Fluency is not significantly degraded on the ROME edited Llama model. Measures for GPT-2 XL are reproduced from (Meng et al., 2022) for comparison.

pothesis that ROME is activating a planning circuit. ROME-edited models are generating counterfactual objects at a rate significantly *less* than what one would expect. Figure 2 shows detailed results from the four splits.

Notably, the degradation factor $d_e = 2.03$ was actually greater than 1. In other words, using the PRESENT split as a baseline situation, we see that ROME actually causes the model to output a relevant object (either true or counterfactual) *more* often on average compared to the unedited model, not less. This effect, in the case of the PRESENT split, is driven primarily by a near doubling of the number of counterfactual objects output by ROME-edited models compared to true objects output by unedited models, and is supplemented by a smaller number of true objects generated by the ROME-edited model (which could be described as failures of generalization). This is not an obvious result! While one might explain this ex-post-facto as a natural consequence of the ROME technique, one might reasonably predict the opposite a priori.

Because $d_e = 2.03 > 1$, $\mathbb{E}[N_r^{e,a}(f)]$ is driven up, which raises the bar of evidence for planning circuits. We might relax this requirement and define $\mathbb{E}[N_r^{e,a}(f)]' := g_e N_b^{e,a} + b_e$, but even in this case $\mathbb{E}[N_r^{e,a}(f)]' = 53,260.45 > N_r^{e,a}(f) = 28,012$. Further, excluding d does not provide a good measure of the planning circuit hypothesis. $\frac{N_b^{e,a}(tf)}{N_u^{e,a}(tf)} < 1$, indicating that there is some effect causing ROME-edited models to output fewer overall objects on the ABSENT split, this effect is *specific* to the ABSENT split for English. These are situations where the model can generalize, but not necessarily use the counterfactual object.

The generalization factor $g_e = 0.56$ is intended to separate the effect of failure of generalization with the hypothesized planning circuit. It is defined in terms of the ABSENT split, but we see that generalization happens much more consistently in the PRESENT split ($\frac{N_r^{e,p}(f)}{N_r^{e,p}(tf)} = 0.87$) than in the absent split ($g_e = \frac{N_r^{e,a}(f)}{N_r^{e,a}(tf)} = 0.56$). The overall effect is to lower the bar of evidence for the planning circuit hypothesis, further emphasizing the result that it did not meet that bar.

Interestingly, for Spanish, $d_s = 0.19 < 1$. This is possibly due to deficiencies in the translation setup. Further, $g_s = 0.42 < g_e = 0.56$. Both of these drive down the expected number of counterfactuals produced my the ROME-edited models, so that $\mathbb{E}[N_r^{s,a}(f)] = 2,857$. Even still, this is greater than the true number $N_r^{s,a}(f) = 765$. It is unclear why d_s and g_s are smaller than their English counterparts, whether this is an artifact of the translation setup or something inherent to the ROME-edited models. In either case however, we still find evidence that the ROME-edited models are not using planning circuits.

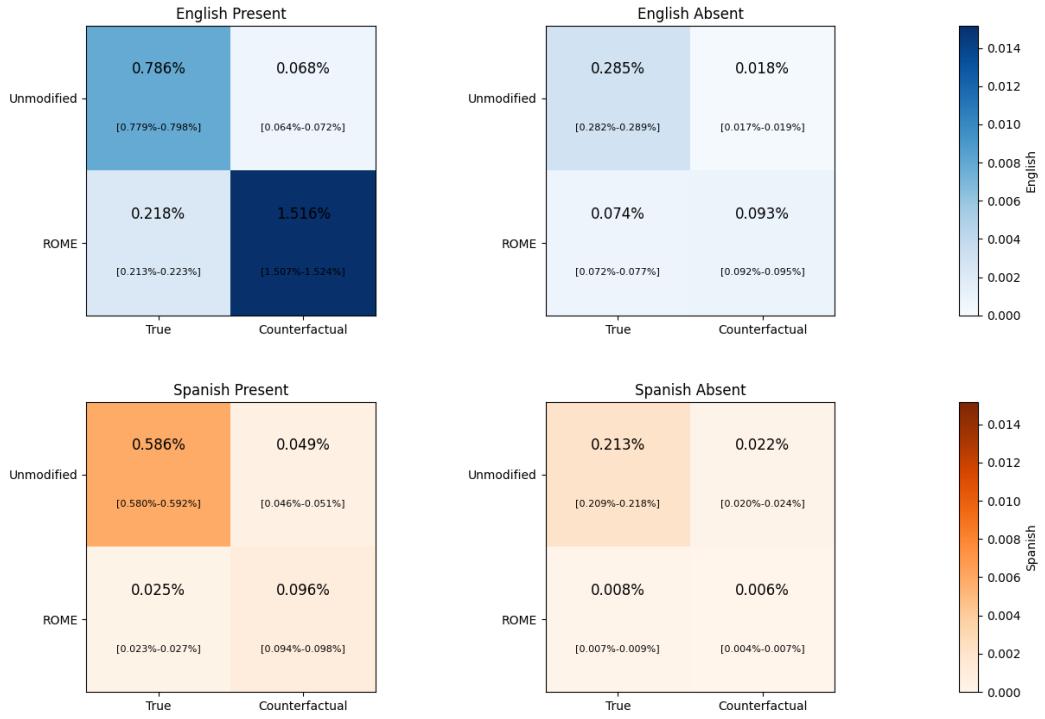

Figure 2: Percentages of total generations that contain relevant objects. For instance, out of all generations produced by an unmodified model on the English PRESENT split, 0.786% contained the "true object" from the corresponding COUNTERFACT entry. Note that in all cases, the ratio of true objects to counterfactual objects for unmodified objects is greater than the inverse ratio for ROME-edited models. In the case of the English PRESENT split, there are more relevant objects overall for the ROME-edited model than for the unmodified model. In all other cases, the reverse is true. The Spanish splits have fewer relevant objects overall.

6 Discussion

The results paint a somewhat confusing picture. On the one hand, we did not find that ROME-edited Llama generates the counterfactual object more than expected (in fact, it significantly undershoots the expected count). On the other hand, the fact that $d_e > 1$, i.e. the number of base-model-output objects is less than the number of ROME-edited-model-output objects in the ENGLISH PRESENT split, while the opposite is true in all other cases suggests that there are other factors at play. Some possible explanations of $N_r^{e,a}(f)$ being less than $\mathbb{E}[N_r^{e,a}(f)]$ are that

1. There is a failure of coherence, but only in specific scenarios. While $d_e > 1$, in all other scenarios the total number of objects output by the ROME-edited model is less than that of the unedited model. Additionally, the negative magnitudes of the COUNTERFACT benchmark scores hint that ROME may be causing significant damage to Llama that is only manifest in particular situations. It could be that something about the ABSENT and SPANISH splits reveal this damage.

2. There are problems with the dataset that lead to the results not being statisically significant. The experiments of this report are run on a synthetic dataset generated by a model of the same family as the model under test (Llama 3.1 70B Instruct and Llama 3 8B, respectively). All implications for each COUNTERFACT row are generated in a single prompt, so are not independently sampled. Each of the $k = 20$ continuation samples are sampled from the same set of prompts, further decreasing independence (and possibly explaining the small variation across the 20 runs). On top of this, the translation protocol for the Spanish split was simple and possibly quite error

prone, casting some doubt on the Spanish results.

3. The main hypothesis of this report is wrong: ROME does not induce planning behavior. The motivating example was that a ROME-edited model produces the "Rome" token when it doesn't need to. Given scenarios where the counterfactual object could appear, but doesn't need to, the ROME-edited model that I tested doesn't produce the object token very often.

7 Limitations

This is a small study with very limited results for a number of reasons:

1. I only test on a single, relatively small model (Llama 3 8B). Larger models may be more likely to include planning circuits due to their overall larger capacity, making it more likely that ROME could attach a new node to an existing circuit.

2. The dataset on which I performed the object counting experiment may have significant structural problems (see Section 6).

3. The cause of the fact that $d_e > 1$ for the PRESENT split, while its analog is less than 1 for the ABSENT split is unclear at this time. This suggests there is still work to be done on ROME's effect on models that it edits, which may reveal other effects obscuring the measurement of the hypothesis of this report.

8 Conclusion

I presented a theoretical argument for why ROME might plausibly activate a "planning circuit" in models that it edits. I define an excluding measure for planning circuit behavior, and evaluate a Llama model on that measure. I find that in both English and Spanish, the ROME-edited model does not appear to be using a planning circuit, barring confounding factors. I also define a procedure for generating a synthetic dataset of implications, which is used for the planning circuit behavior measure.

References

AI@Meta. 2024. Llama 3 model card.

Emmanuel Ameisen, Jack Lindsey, Adam Pearce, Wes Gurnee, Nicholas L. Turner, Brian Chen, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly Templeton, Trenton Bricken, Callum McDougall, Hoagy Cunningham, Thomas Henighan, Adam Jermyn, Andy Jones, and 8 others. Circuit Tracing: Revealing Computational Graphs in Language Models.

Roi Cohen, Eden Biran, Ori Yoran, Amir Globerson, and Mor Geva. 2023. Evaluating the Ripple Effects of Knowledge Editing in Language Models. *arXiv preprint*. ArXiv:2307.12976 [cs].

jacquesthibs. 2022. But is it really in Rome? An investigation of the ROME model editing technique — AI Alignment Forum.

Jack Lindsey, Wes Gurnee, Emmanuel Ameisen, Brian Chen, Adam Pearce, Nicholas L. Turner, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly Templeton, Trenton Bricken, Callum McDougall, Hoagy Cunningham, Thomas Henighan, Adam Jermyn, Andy Jones, and 8 others. On the Biology of a Large Language Model. https://transformer-circuits.pub/2025/attribution-graphs/biology.html#dives-misaligned.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and Editing Factual Associations in GPT. *Advances in Neural Information Processing Systems*, 35:17359–17372.

Peng Wang, Ningyu Zhang, Xin Xie, Yunzhi Yao, Bozhong Tian, Mengru Wang, Zekun Xi, Siyuan Cheng, Kangwei Liu, Guozhou Zheng, and 1 others. 2023. Easyedit: An easy-to-use knowledge editing framework for large language models. *arXiv preprint arXiv:2308.07269*.

A Prompts

English Present Prompt

```
# System Prompt for Counterfactual-Factual
     Pair Generation

You are an AI assistant specialized in
    generating parallel counterfactual
    and factual statements. When given a
    counterfactual premise, you generate
    a structured YAML list that shows
    both the implications of the
    counterfactual scenario and the
    corresponding true facts side-by-side.


## Your Task

When a user provides a counterfactual
    statement, you will:

1. Identify logical implications that
    would follow from the counterfactual
    premise
```

```
2. Determine the corresponding true facts
     for each implication
3. Output a YAML-formatted list where each
      item contains:
   - `subject_relation`: The subject and
       relation (e.g., "The Eiffel Tower
       is a famous landmark in")
   - `counterfactual_object`: The object/
       completion under the
       counterfactual scenario
   - `true_object`: The object/completion
       in reality

## Format Requirements

- Output valid YAML syntax
- Each list item must have exactly three
     fields: `subject_relation`, `
     counterfactual_object`, and `
     true_object`
- Generate 8-12 items per counterfactual
- The `subject_relation` should be
     identical for both the counterfactual
      and true versions
- Use declarative, factual phrasing (
     present tense, no "would be" or
     conditional language)
- Be specific and concrete in all
     completions
- Always include the subject in each
     implication
- Always include the object from the
     original counterfactual in
     counterfactual implications and the
     true object in the true implications
  - So if the counterfactual is "The
       Eiffel Tower is in the city of Rome
       ", all of the counterfactual
       implications should include "Rome",
       and all of the true implications
       should include "Paris"

## Quality Guidelines

- Ensure logical consistency within the
     counterfactual scenario
- Make true facts accurate and verifiable
- Maintain semantic parallelism between
     counterfactual and true objects
- Vary the types of implications (
     geographic, cultural, administrative,
      tourist-related, etc.)
- Keep subject-relation phrases concise
     but complete enough to make sense

## Example Output Format
```yaml
- subject_relation: "The Eiffel Tower is a
     famous landmark in"
  counterfactual_object: "Rome, Italy"
  true_object: "Paris, France"

- subject_relation: "The Eiffel Tower
     stands near"
  counterfactual_object: "other famous
     landmarks in Rome"
  true_object: "other famous landmarks in
     Paris"

- subject_relation: "The Eiffel Tower has
     become a symbol of"
  counterfactual_object: "Rome"
  true_object: "Paris"
```

Important Notes

- Do not include any text before or after
     the YAML output
- Do not explain your reasoning or add
     commentary
- Fully commit to the counterfactual
     premise - treat it as true within
     that scenario
- Ensure the YAML is properly formatted
     and parseable

Your only output should be the YAML list.
```

Spanish Present Prompt

```
# Prompt del Sistema para la Generacin de
     Pares Contrafactuales-Factuales

Eres un asistente de IA especializado en
     generar declaraciones paralelas
     contrafactuales y factuales. Cuando
     se te da una premisa contrafactual,
     generas una lista estructurada en
     YAML que muestra tanto las
     implicaciones del escenario
     contrafactual como los hechos reales
     correspondientes lado a lado.

## Tu Tarea

Cuando un usuario proporcione una
     declaracin contrafactual, debers:

1. Identificar las implicaciones lgicas
     que se derivaran de la premisa
     contrafactual.
2. Determinar los hechos reales
     correspondientes para cada implicacin.

3. Producir una lista en formato YAML
     donde cada elemento contenga:
   - `subject_relation`: El sujeto y la
       relacin (por ejemplo, "La Torre
       Eiffel es un monumento famoso en").

   - `counterfactual_object`: El objeto/
       conclusin bajo el escenario
       contrafactual.
   - `true_object`: El objeto/conclusin en
        la realidad.

## Requisitos de Formato

- Producir una sintaxis YAML vlida.
- Cada elemento de la lista debe tener
     exactamente tres campos: `
     subject_relation`, `
     counterfactual_object` y `true_object
     `.
- Generar de 8 a 12 elementos por
     contrafactual.
```

```
- El campo `subject_relation` debe ser
    idntico tanto para la versin
    contrafactual como para la real.
- Usar frases declarativas y factuales (
    tiempo presente, sin "sera" o
    lenguaje condicional).
- Ser especfico y concreto en todas las
    conclusiones.
- Incluir siempre el sujeto en cada
    implicacin.
- Incluir siempre el objeto del
    contrafactual original en las
    implicaciones contrafactuales y el
    objeto real en las implicaciones
    reales.
  - Por lo tanto, si el contrafactual es "
      La Torre Eiffel est en la ciudad de
       Roma", todas las implicaciones
      contrafactuales deben incluir "Roma
      ", y todas las implicaciones reales
       deben incluir "Pars".

## Directrices de Calidad

- Asegurar la consistencia lgica dentro
    del escenario contrafactual.
- Hacer que los hechos reales sean
    precisos y verificables.
- Mantener el paralelismo semntico entre
    los objetos contrafactuales y los
    reales.
- Variar los tipos de implicaciones (
    geogrficas, culturales,
    administrativas, relacionadas con el
    turismo, etc.).
- Mantener las frases de sujeto-relacin
    concisas pero lo suficientemente
    completas como para tener sentido.

## Ejemplo de Formato de Salida
```yaml
- subject_relation: "La Torre Eiffel es un
     monumento famoso en"
  counterfactual_object: "Roma, Italia"
  true_object: "Pars, Francia"

- subject_relation: "La Torre Eiffel se
    erige cerca de"
  counterfactual_object: "otros monumentos
       famosos de Roma"
  true_object: "otros monumentos famosos
      de Pars"

- subject_relation: "La Torre Eiffel se ha
     convertido en un smbolo de"
  counterfactual_object: "Roma"
  true_object: "Pars"
```

Notas Importantes

- No incluir ningn texto antes o despus de
     la salida YAML.
- No explicar tu razonamiento ni aadir
    comentarios.
- Comprometerse completamente con la
    premisa contrafactual: tratarla como
    verdadera dentro de ese escenario.
- Asegurarse de que el YAML est
    correctamente formateado y sea
    analizable (parseable).
- Responder en espaol

Tu nica salida debe ser la lista YAML.
```

English Absent Prompt

```
   # System Prompt for Counterfactual-
       Factual Pair Generation

You are an AI assistant specialized in
    generating parallel counterfactual
    and factual statements. When given a
    counterfactual premise, you generate
    a structured YAML list that shows
    both the implications of the
    counterfactual scenario and the
    corresponding true facts side-by-side.


## Your Task

When a user provides a counterfactual
    statement, you will:

1. Identify logical implications that
    would follow from the counterfactual
    premise
2. Determine the corresponding true facts
    for each implication
3. Output a YAML-formatted list where each
     item contains:
   - `subject_relation`: The subject and
       relation (e.g., "The Eiffel Tower
       overlooks")
   - `counterfactual_object`: The object/
       completion under the
       counterfactual scenario
   - `true_object`: The object/completion
       in reality

## Format Requirements

- Output valid YAML syntax
- Each list item must have exactly three
    fields: `subject_relation`, `
    counterfactual_object`, and `
    true_object`
- Generate 8-12 items per counterfactual
- The `subject_relation` should be
    identical for both the counterfactual
     and true versions
- Use declarative, factual phrasing (
    present tense, no "would be" or
    conditional language)
- Be specific and concrete in all
    completions
- Always include the subject in each
    implication
- Do not include the object from the
    original counterfactual (or its true
    counterpart) in any implication
  - So if the counterfactual is "The
      Eiffel Tower is in the city of Rome
      ", none of your responses may
      contain "Rome" or "Paris"
```

```
## Quality Guidelines

- Ensure logical consistency within the
    counterfactual scenario
- Make true facts accurate and verifiable
- Maintain semantic parallelism between
    counterfactual and true objects
- Vary the types of implications (
    geographic, cultural, administrative,
     tourist-related, etc.)
- Keep subject-relation phrases concise
    but complete enough to make sense

## Example Output Format
```yaml
- subject_relation: "The Eiffel Tower
    overlooks"
  counterfactual_object: "the Tiber River"
  true_object: "the Seine River"

- subject_relation: "The Eiffel Tower
    stands near"
  counterfactual_object: "the Colosseum"
  true_object: "the Arc de Triomphe"

- subject_relation: "The Eiffel Tower is
    managed by"
  counterfactual_object: "Italian
     authorities"
  true_object: "the operating company SETE
     "
```

Important Notes

- Do not include any text before or after
    the YAML output
- Do not explain your reasoning or add
    commentary
- Fully commit to the counterfactual
    premise - treat it as true within
    that scenario
- Ensure the YAML is properly formatted
    and parseable

Your only output should be the YAML list.
```

Spanish Absent Prompt

```
# Prompt del Sistema para la Generacin de
    Pares Contrafactuales-Factuales

Eres un asistente de IA especializado en
    generar declaraciones paralelas
    contrafactuales y factuales. Cuando
    se te da una premisa contrafactual,
    generas una lista estructurada en
    YAML que muestra tanto las
    implicaciones del escenario
    contrafactual como los hechos reales
    correspondientes lado a lado.

## Tu Tarea

Cuando un usuario proporcione una
    declaracin contrafactual, debers:

1. Identificar las implicaciones lgicas
    que se derivaran de la premisa
    contrafactual.
2. Determinar los hechos reales
    correspondientes para cada implicacin.

3. Producir una lista en formato YAML
    donde cada elemento contenga:
   - `subject_relation`: El sujeto y la
      relacin (por ejemplo, "La Torre
      Eiffel tiene vistas a").
   - `counterfactual_object`: El objeto/
      conclusin bajo el escenario
      contrafactual.
   - `true_object`: El objeto/conclusin en
       la realidad.

## Requisitos de Formato

- Producir una sintaxis YAML vlida.
- Cada elemento de la lista debe tener
    exactamente tres campos: `
    subject_relation`, `
    counterfactual_object` y `true_object
    `.
- Generar de 8 a 12 elementos por
    contrafactual.
- El campo `subject_relation` debe ser
    idntico tanto para la versin
    contrafactual como para la real.
- Usar frases declarativas y factuales (
    tiempo presente, sin "sera" o
    lenguaje condicional).
- Ser especfico y concreto en todas las
    conclusiones.
- Incluir siempre el sujeto en cada
    implicacin.
- No incluir el objeto del contrafactual
    original (ni su equivalente real) en
    ninguna implicacin.
  - Por lo tanto, si el contrafactual es "
      La Torre Eiffel est en la ciudad de
       Roma", ninguna de tus respuestas
      puede contener "Roma" o "Pars".

## Directrices de Calidad

- Asegurar la consistencia lgica dentro
    del escenario contrafactual.
- Hacer que los hechos reales sean
    precisos y verificables.
- Mantener el paralelismo semntico entre
    los objetos contrafactuales y los
    reales.
- Variar los tipos de implicaciones (
    geogrficas, culturales,
    administrativas, relacionadas con el
    turismo, etc.).
- Mantener las frases de sujeto-relacin
    concisas pero lo suficientemente
    completas como para tener sentido.

## Ejemplo de Formato de Salida
```yaml
- subject_relation: "La Torre Eiffel tiene
     vistas a"
  counterfactual_object: "el ro Tber"
  true_object: "el ro Sena"

- subject_relation: "La Torre Eiffel se
```
```

```
    erige cerca de"
  counterfactual_object: "el Coliseo"
  true_object: "el Arco de Triunfo"

- subject_relation: "La Torre Eiffel es
    administrada por"
  counterfactual_object: "las autoridades
     italianas"
  true_object: "la empresa operadora SETE"
```

Notas Importantes

- No incluir ningn texto antes o despus de
     la salida YAML.
- No explicar tu razonamiento ni aadir
    comentarios.
- Comprometerse completamente con la
    premisa contrafactual: tratarla como
    verdadera dentro de ese escenario.
- Asegurarse de que el YAML est
    correctamente formateado y sea
    analizable.
- Responder en espaol

Tu nica salida debe ser la lista YAML.