From Early Encoding to Late Suppression: Interpreting LLMs on Character Counting Tasks

Ayan Datta[1], Mounika Marreddy[3], Alexander Mehler[3]
Zhixue Zhao[2], Radhika Mamidi[1]

[1]IIIT Hyderabad, [2]University of Sheffield, [3]Goethe University
advin4603@gmail.com, mmarredd@em.uni-frankfurt.de, mehler@em.uni-frankfurt.de
zhixue.zhao@sheffield.ac.uk, radhika.mamidi@iiit.ac.in

Abstract

Large language models (LLMs) still fail at counting the characters in a word, even while they excel on far harder benchmarks. Using character counting (e.g., "How many 'p's are in apple?") as a controlled task, we find a consistent pattern across Llama, Qwen, and Gemma in both base and instruction-tuned variants. The models compute the correct answer internally but fall back on degenerate output strategies, such as always predicting "1" or guessing uniformly among "1", "2", and "3". Probing classifiers, activation patching, and logit lens analysis show that character-level information is encoded in early and mid layers, then actively attenuated by a small set of later components, mainly the penultimate and final layer MLP and sometimes the final layer attention heads. The failure is not missing information or insufficient scale but structured interference in the computation graph, which explains why these errors persist under scaling and instruction tuning.

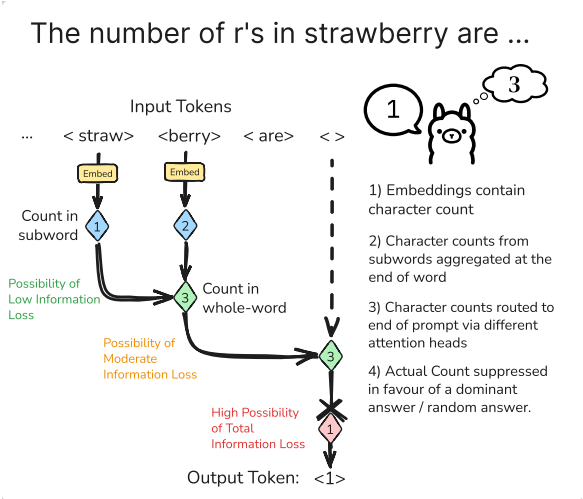


Figure 1: Overview of how character-level information flows through an LLM during a character counting task, from subword token embeddings through aggregation and routing, to final-layer suppression of the correct answer.

1 Introduction

Large language models (LLMs) (Achiam et al., 2023; Team et al., 2024a; Touvron et al., 2023; Jiang et al., 2023a) achieve state-of-the-art performance on diverse natural language processing (NLP) tasks (Brown et al., 2020; Chung et al., 2024), yet they often fail on problems that appear trivial to humans, requiring only basic symbolic reasoning (Shin and Kaneko, 2024). Prior studies have documented such deficiencies in simple operations and reasoning, including letter containment and first/last character identification (Efrat et al., 2022), spelling and character edits (Edman et al., 2024), and character- and word-level manipulation tasks (Zhang and He, 2024). While these tasks are elementary, failures on them reveal systemic fragility: if models cannot reliably solve such atomic operations, their robustness on more complex reasoning tasks may rest on unstable foundations. Despite these observations, the mechanisms underlying such failures remain poorly understood.

To this end, we revisit character counting as a controlled diagnostic task to study this gap. Character counting is minimal yet stringent: solving it requires encoding fine-grained sub-token information and reliably propagating it through the model's computation. Unlike surface-level accuracy, analyzing the internal components and dynamics provides insight into whether failures stem from representational absence or from interference during reasoning. We evaluate three recent model families, Llama 3.2 (Grattafiori et al., 2024a), Qwen 2.5 (Yang et al., 2024), and Gemma 2 (Team et al., 2024b), in both base and instruction-tuned variants, using a dataset constructed from WordNet (Miller, 1992). We employ probing classifiers, logit lens analysis, and activation patching to trace the dynamics of internal representation. Our findings re-

Findings of the Association for Computational Linguistics: IJCNLP-AACL 2026, pages 1370–1391
November 6-10, 2026 ©2026 Association for Computational Linguistics

veal structural patterns across families and scales:

1. LLMs capture character-level information in their residual stream, but the signal degrades when transferred to the final prediction position.

2. Rather than counting, models fall back on degenerate strategies, either always predicting a fixed count (typically "1" for Qwen and Gemma) or guessing near-uniformly (Llama).

3. Attention heads do not specialize in routing character information.

4. Components at the later layers, mainly the penultimate and final layer MLP and sometimes the final layer attention heads, suppress the probability of the correct answer.

5. Scaling from 2B to 9B parameters does not help, and the same patterns persist under post-training.

We make our code publicly available. [1]

2 Counting Task

To assess whether LLMs can process fine-grained character-level information, we design a controlled character-counting task. In this task, the model must determine how many times a specific target letter appears in a given word or phrase. To implement this, we construct natural language prompt templates where the correct count appears as the final token to be predicted. For example, the prompt "The number of 'i's in 'distribution' is " elicits a probability distribution over plausible counts, enabling us to measure whether the model assigns high probability to the correct answer. To minimize biases from prompt phrasing or word selection, we employ multiple paraphrased templates, as shown in Table 1. We design two sets of templates: one for base models (Table 1) and another for instruction-tuned models (Appendix Table 9). Input words are sampled from WordNet (Miller, 1992) and scraped from the Wiktionary text dump (Wikimedia Foundation, 2024). When generating prompt examples, we ensure that the distribution of target counts is balanced. By combining multiple templates with a diverse set of words, we reduce template- and word-specific biases, leading to a more reliable evaluation of the model's character-counting ability.

[1] https://github.com/advin4603/counting_r

We evaluate a range of base and instruction-tuned models on the same set of 10000 randomly generated dataset balanced across the final counts 1, 2, and 3 to establish their baseline performance on the task before analyzing the underlying reasons for failure. Since the prompts require only the prediction of the next token, we perform a single forward pass through the model and obtain probability distributions over the vocabulary. The token with the highest probability is then selected as the model's predicted answer.

Prompt Templates
The number of <target_letter>'s in <count_subject> is <count>
Counting the letter <target_letter> in <count_subject> gives <count>
The number of <target_letter>'s found in <count_subject> is <count>
The <target_letter> count for <count_subject> equals <count>
The total number of <target_letter>'s in <count_subject> is <count>

Table 1: Prompt templates for probing character count representations. For example: "The number of p's in apple is 2", where p is the target letter and apple is the count_subject.

3 Methodology

This section details the experimental setup for evaluating LLMs on the character-level counting task. We systematically design controlled prompts and input variations to test the reliability of the model. Furthermore, we apply a series of interpretability techniques to identify the model components and computational pathways that are most responsible for generating the final counts.

3.1 Probing Internal Representations

We investigate whether these models encode character-count information within their internal representations, thereby testing the hypothesis that LLMs primarily operate at the token level and fail to capture sub-token features. If character-level information is present in hidden states, this would indicate that models retain fine-grained cues even if they do not reliably leverage them during generation. To distinguish between absence of encoding and failure of utilization, we conduct probing experiments to test whether the frequency of a given letter can be linearly decoded from internal representations.

Probe Model. For each layer, we train a linear classifier without bias:

$$\mathbf{z} = W\mathbf{h} \tag{1}$$

where $\mathbf{h} \in \mathbb{R}^d$ is the hidden representation, $W \in \mathbb{R}^{C \times d}$ is a learned weight matrix, and C is

the number of count classes. The probe is trained using cross-entropy loss,

$$\mathcal{L} = \text{CrossEntropy}(\mathbf{z}, y), \tag{2}$$

with predictions given by $\hat{y} = \arg\max_c z_c$.

Dataset Construction. For each target letter, we construct a dataset from Wiktionary by grouping words according to the number of occurrences of the letter. Starting from count 1, we sample 1,000 words per count and continue until fewer than 1,000 examples are available. We additionally include 1,000 randomly sampled words with zero occurrences. For each word and letter, we generate inputs using all templates and extract hidden states across all layers. When a word is split into multiple tokens, we use the hidden state of the final token to obtain a single representation unless otherwise specified.

Probing Settings. We consider three settings:

1. **Token-level (individual tokens):** Predict the count of the target letter within individual tokens (subword units). Splits are constructed such that token types are disjoint across train, validation, and test sets.

2. **Word-level (final word token):** Predict the total count of the target letter in the word using the final token representation. Word types are disjoint across splits.

3. **Prompt-level (final prompt token):** Predict the total count using the final token of the full prompt. Word types remain disjoint across splits.

Training Procedure. We use a 70%-15%-15% train-dev-test split with partitioning over types rather than instances: no token (token-level) or word (word- and prompt-level) appears in more than one split. We generate prompts using all of the templates. Probes are trained with Adam (Kingma and Ba, 2017), using an initial learning rate of 10^{-3}, which is annealed by a factor of 0.5 when validation accuracy does not improve. Early stopping is based on validation performance, and results are reported on the test set.

We do not employ control tasks (Hewitt and Liang, 2019) as our dataset construction already structurally prevents memorization. We discuss this in detail in Appendix A.

3.2 Localizing Components

We attempt to localize task behavior to different components of the model in order to identify the source of poor counting performance.

3.2.1 Performance Evaluation

To attribute changes in model behavior to specific internal components, we require evaluation metrics beyond accuracy. Accuracy alone is discrete and does not capture the model's confidence or relative preference among competing tokens. We therefore adopt a logit-difference metric (Δ) from (Wang et al., 2023; Conmy et al., 2023a), which measures how strongly the model favors the correct token relative to plausible alternatives as a linear function of the residual stream:

$$\Delta = L(\text{correct token}) - \mathbb{E}_{t\sim T}[L(t)] \tag{3}$$

$$L(t) = (W_U \cdot \mathbf{h})[\text{VocabularyIndex}(t)] \tag{4}$$

Here, $L(t)$ denotes the logit of token t, obtained by projecting the final hidden state $\mathbf{h}$ with the unembedding matrix W_U. T is the set of the top-N highest-probability incorrect tokens (we use $N = 2$). Since the answer space dominantly consists of the tokens $\{1, 2, 3\}$, this in practice reduces to the logit of the correct count minus the mean logit over the remaining two candidates.

The Δ metric provides a continuous, interpretable measure of preference for the correct answer. Values of $\Delta > 0$ indicate that the correct token is favored over alternatives, and larger values correspond to stronger preference. Values of $\Delta < 0$ indicate that the correct token is less likely than the competing alternatives, reflecting a preference against the correct answer. Crucially, Δ is linearly related to the residual stream, making it well-suited for attributing behavior to internal components.

In addition, we use a probability-gap metric to measure how the correct answer is suppressed relative to the model's observed prediction:

$$\Delta_p = P(\text{observed}) - P(\text{correct}) \tag{5}$$

where probabilities are obtained by applying a softmax over logits. This metric captures the extent to which probability mass is shifted away from the correct token toward an incorrect alternative. The two metrics are complementary (Wiegreffe et al., 2025). We use Δ to track whether the model builds preference for the correct answer across layers, since it isolates the correct-answer signal without confounding from global logit shifts. We use Δ_p to measure suppression, capturing how much probability mass moves away from the correct answer toward competing tokens. Unlike Δ, the

Corruption Type	Clean Prompt	Corrupted Prompt
Word Corruption	The number of a's in **apple** is	The number of a's in **ada** is
Letter Corruption	The number of **a**'s in apple is	The number of **p**'s in apple is
Word+Letter Corruption	The number of **a**'s in **apple** is	The number of **p**'s in **ada** is

Table 2: Examples of clean and corrupted prompts used in activation patching. Each corruption alters the expected letter count while maintaining the prompt structure.

probability-gap metric is not linear in the residual stream because of the softmax, but it gives a normalized and more interpretable scale for comparing probabilities. We note that Δ is closely related to cross-entropy loss: taking differences in cross-entropy between two tokens reduces to their logit difference (see Appendix B).

3.2.2 Logit Lens

To trace how predictions evolve through the network, we apply the Logit Lens (nostalgebraist, 2020), projecting intermediate hidden states onto the vocabulary space using the model's unembedding matrix. At a given layer index i, we project hidden states from two different locations, i_{pre} (residual stream right before the attention layer) and i_{mid} (residual stream right before the MLP Layer). We also use the final layer's hidden state (final$_{post}$). Given an hidden state $\mathbf{h}$ at layer i either from the pre position or the post position, we first ensure the vector is normalise. This means ensuring it is the output of the model's corresponding LayerNorm (Ba et al., 2016) or RMSNorm (Zhang and Sennrich, 2019) layer. If $\mathbf{h}$ was not the output of a normalization layer, we apply the corresponding normalization to obtain $\mathbf{h}_{\text{norm}}$. We compute the corresponding Logit values as such:

$$L_{\text{intermediate}}(t) = (W_U \cdot \mathbf{h}_{\text{norm}})[\text{VI}(t)] \quad (6)$$

We then use the logit values from $L_{\text{intermediate}}$ to compute Δ_p across different layer positions and VI is the vocabulary Index. This evolution of Δ_p values helps track at what point in the model, does the correct answer get suppressed.

3.2.3 Activation patching

Activation patching (Meng et al., 2022; Vig et al., 2020; Geiger et al., 2020) localizes causal mechanisms by overwriting hidden activations from a corrupted input with cached activations from a clean input and observing the change in predictions. How much clean behavior is restored shows which layers, neurons, or subspaces are causally responsible for the output (Zhang and Nanda, 2023; Heimersheim and Nanda, 2024), and the method has been applied widely across interpretability studies (Geiger et al., 2021; Hase et al., 2023; Wang et al., 2022; Hanna et al., 2023; Conmy et al., 2023b). We use our data generator from Section 2 to build clean prompts and corrupted versions that alter the word, the target letter, or both, as shown in Table 2. For each pair we run the clean prompt, cache its hidden states, and measure Δ_{clean} , then run the corrupted prompt and measure $\Delta_{\text{corrupted}}$ against the clean prompt's correct answer. We keep only pairs with $\Delta_{\text{clean}} - \Delta_{\text{corrupted}} > 0.5$ and $\Delta_{\text{clean}} > 0$. We then patch each clean hidden state at layer L and token position T into the corrupted run and measure the resulting Δ_{patched}, quantifying restoration with Equation 7.

$$P_{\text{restored}} = \frac{\Delta_{\text{patched}} - \Delta_{\text{corrupted}}}{\Delta_{\text{clean}} - \Delta_{\text{corrupted}}} \quad (7)$$

We use Δ for patching (it is linear in the residual stream) and reserve Δ_p for output-layer suppression. See Appendix C.

This metric yields P_{restored} for each layer and token position. A value of $P_{\text{restored}} = 0$ indicates that the component does not contribute to the task; $P_{\text{restored}} > 0$ indicates that it increases the probability of the correct answer; and $P_{\text{restored}} < 0$ indicates that it suppresses the probability, thereby hindering the correct answer.

To obtain comprehensive insights, we perform patching at multiple levels: (i) residual outputs of each layer, (ii) individual attention heads, and (iii) MLP sublayers, across all token positions. This multi-level intervention allows us to precisely localize causal contributions.

Finally, to aggregate results across different templates and token lengths, we compute performance restoration scores for each component at different token positions and average them. To prevent small or near-zero contributions from diluting strong effects, we adopt a weighted mean where each score is weighted by its absolute value: Given a set of

scores $S = \{s_1, s_2, ...s_n\}$

$$\text{Aggregate}(S) = \frac{\sum_{i=0}^{n} |s_i| \cdot s_i}{\sum_{i=0}^{n} |s_i|} \quad (8)$$

For word corruptions, we aggregate across positions before, at, and after the target word. For letter corruptions, we aggregate before, at, and after the target letter. For combined word+letter corruptions, we further distinguish positions between the letter and word tokens. We ensure that all templates satisfy letter start $<$ word start, enabling consistent alignment across conditions.

3.3 Models and Implementation Details

We perform our experiments over a range of recent open-weight large language models, covering both base (pretrained) and instruction-tuned variants. From the Qwen-2.5 family (Yang et al., 2024), we include the 3B and 7B parameter models (Qwen-2.5-3B, Qwen-2.5-7B) along with their instruction-tuned counterparts (Qwen-2.5-3B-Instruct, Qwen-2.5-7B-Instruct). From Gemma-2 (Team et al., 2024b), we consider the 2B and 9B models (gemma-2-2b, gemma-2-9b) and their aligned instruction-tuned versions (gemma-2-2b-it, gemma-2-9b-it). For Llama-3.2 (Grattafiori et al., 2024a), we focus on the 3B parameter model, evaluating both the base (Llama-3.2-3B) and instruction-tuned (Llama-3.2-3B-Instruct) variants. We also include an equivalent bigger model, Llama-3.1-8B and the instruction-tuned Llama-3.1-8B-Instruct (Grattafiori et al., 2024b). All checkpoints are obtained from Hugging Face's model hub (Wolf et al., 2020). All experiments are implemented using the TransformerLens interpretability library. The models are executed on a computing setup equipped with an NVIDIA RTX 6000 Ada GPU, with 48 GB of VRAM.

4 Results

4.1 Overall Performance

LLMs are unreliable at character counting. Figure 2 reports the overall accuracy over all the models studied along with a random prediction baseline. We observe substantial error rates, with a large proportion of predictions being incorrect. Notably, even relatively recent architectures such as Llama-3.2, Qwen-2.5, and Gemma-2 fail to achieve reliable performance **achieving close to the random baseline's 33% Accuracy**.

These results confirm and extend prior findings

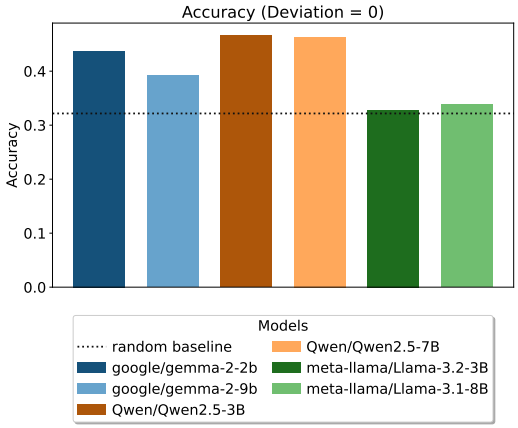


Figure 2: Accuracy of predicted letter counts. The dotted line indicates the random baseline accuracy. We calculate the proportion of samples where deviation is zero.

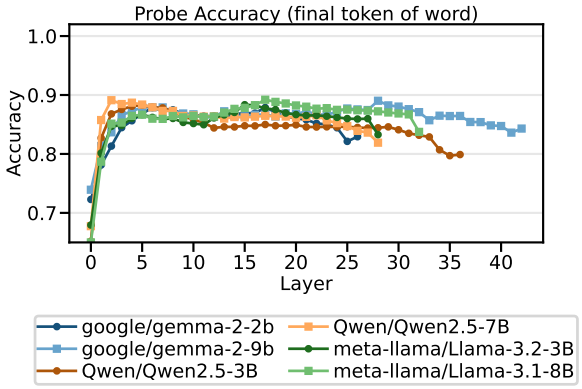
on the limitations of LLMs in character-level tasks, suggesting that precise letter counting remains a systematic weakness rather than an isolated failure. To better understand these failures, we inspect confusion matrices (Appendix D, where we present results for Llama-3.2-3B and Qwen2.5-3B as representative examples of the two dominant failure modes) and observe that models adopt simple, degenerate answer strategies rather than performing true counting. **Gemma and Qwen variants overwhelmingly predict a fixed count (typically "1"), while Llama models exhibit near-uniform predictions across counts.** These behaviors explain both the poor overall performance and the patterns observed in Figure 2. In particular, the near-random strategy employed by Llama accounts for its accuracy being close to the random baseline. We discuss both strategies and their implications in detail in Appendix D.

We provide a detailed performance breakdown across factors such as letter type (vowel vs. consonant) and positional effects in Appendix E.

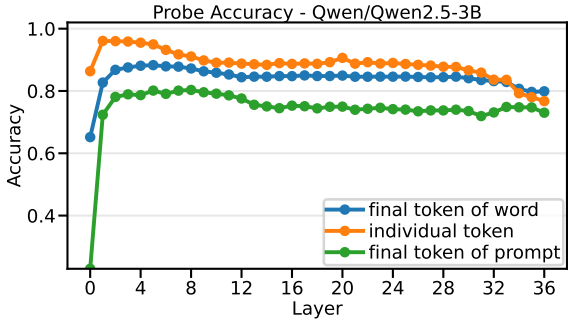
4.2 Probing Tasks

Figure 3a compares probing performance for all the models, averaging accuracies from all the letter probes. Figure 3b compares probing performance for Qwen2.5-3B across all the probe positions, averaging accuracies from all the letter probes.

Character-count information is reliably encoded. Probing results show that character-count information is consistently encoded in model representations as seen in Figure 3a from **high accu-**



(a) Across models at final token



(b) Across positions for Qwen2.5-3B

Figure 3: Layerwise probe accuracy

racies above 80% for most layers across all the models. Probes achieve high accuracy across all target letters on a fully held-out test set containing words and tokens never seen during training, indicating that the signal is not due to memorization. Figure 3a compares performance across models at the final token of the word and shows that letter-count information is reliably recoverable from the residual stream at this position.

This suggests that models can both represent character-level features within tokens and compose them across subword units to form word-level counts.

Information degrades during transfer to the prediction position. To understand how this information is propagated, Figure 3b compares probe performance at three positions for Qwen2.5-3B: individual tokens, the final token of the word, and the final token of the prompt. We observe that **token-level probes achieve the highest accuracy (higher than 90% in the initial layers)**, indicating that models reliably encode character information within individual tokens. **Performance at the final word token remains high (reliably above 80%)**, with only a slight drop, suggesting that models are largely able to aggregate information across subword tokens with minimal loss.

However, **probe accuracy drops substantially at the final token of the prompt (consistently below 80%)**, which is the position used for next-token prediction. This indicates that while character-count information is present and composable within the word, it is not always completely preserved or routed to the final prediction position. The results suggest moderate information loss during this transfer, highlighting a disconnect between representation and utilization.

Encoding is independent of final prediction correctness. The correct count remains decodable from earlier-layer residual streams even when the model outputs the wrong count, confirming that the failure is one of utilization rather than representation.

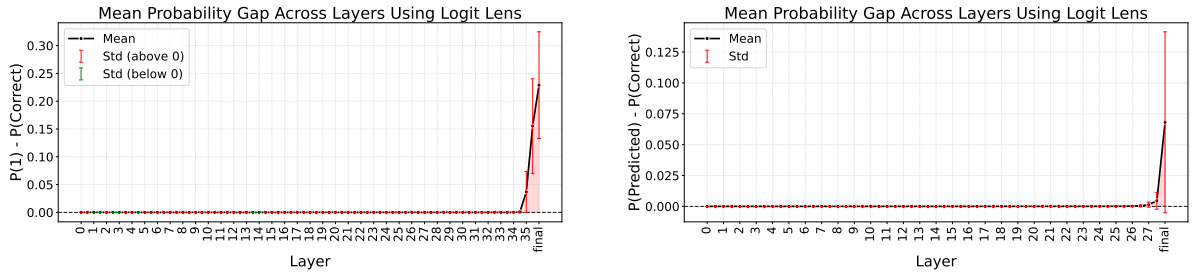
A breakdown of probe accuracy by individual target letter, revealing variation across the alphabet, is provided in Appendix F.

4.3 Logit Lens Analysis

Final-layer components suppress the correct answer. To trace how model predictions evolve across layers, we apply the logit lens and track the probability gap (Δ_p), which we take as a proxy for suppression since it captures both an increase in the probability of incorrect answers and a decrease in the probability of the correct answer. Despite probes indicating that character-count information is available, Figure 4 shows that later layers increasingly suppress the correct answer. We call the components comprising these layers negative components, and the circuit comprised of these negative components a negative circuit. This indicates that while the correct count is encoded in the residual stream, it is not successfully transferred from the representational space into the probability space that is ultimately consumed by the unembedding layer to produce output logits.

We find that the final layer, particularly the MLP component, is primarily responsible for this behavior. Instead of utilizing the available information, it amplifies degenerate answer strategies. For Qwen2.5-3B, this corresponds to the "always predict 1" strategy, while for Llama3.2-3B it corresponds to a near-uniform (random) prediction pattern.

Quantitatively, **the final layer increases the probability of the strategy-consistent answer by approximately 30% for Qwen and 7% on average for Llama.** Although variance across samples



(a) Qwen2.5-3B **(b) Llama3.2-3B**

Figure 4: Logit difference across layers for the correct count token, averaged over approximately 100 samples. Higher values indicate layers where suppression of the correct answer occurs. The final label *final* denotes the final hidden state after all layers. Results are computed over samples where the model produces an incorrect prediction, corresponding to the "always predict 1" strategy for Qwen2.5-3B and the near-uniform random strategy for Llama3.2-3B.

is high, the one-standard-deviation range remains predominantly in the positive regime, indicating that suppression of the correct answer is the dominant behavior.

These results suggest that the failure is not due to absence of information, but rather due to late-stage transformations that override correct signals in favor of simpler heuristics.

Suppression is concentrated in late MLP and attention components. A more fine-grained analysis reveals that suppression is not uniformly distributed across the network, but is concentrated in a small set of late-layer components. **In Qwen2.5-3B, the penultimate-layer MLP suppresses approximately 4% of probability mass on average, the final-layer attention heads contribute a further 9% suppression, and the final-layer MLP accounts for an additional 8%.** Taken together, the last two layers dominate the overall suppression effect.

In contrast, Llama3.2-3B exhibits a more localized pattern. **The penultimate-layer MLP has negligible impact, and the final-layer attention heads contribute only around 0.2% suppression.** The effect is instead driven almost entirely by **the final-layer MLP, which suppresses approximately 7.5% of the correct-answer probability mass.** This difference aligns with the observed behavioral patterns, where Qwen exhibits a strong deterministic bias ("predict 1"), whereas Llama produces near-uniform outputs.

Overall, these findings indicate that a small number of late-stage components are sufficient to override otherwise correct intermediate representations.

4.4 Activation Patching

No specific Attention Heads identified in routing character information. From Activation patch results (Qwen2.5-3B attached in Figure 5, the remaining attached in Appendix K.1, J.1, and I.1) we observe that while information is routed from the word and letter to the subsequent tokens from the middle layers till the later layers, no single attention layer consists of an attention head that shows a strong or consistent restoration effect across layers. Instead, **mild restoration (less than 5%) appears scattered across many different layers**. This suggests that the model does not rely on a dedicated attention layer to transfer character-level information. Additionally, we also run patching experiments on individual head outputs, and find that there are no attention heads with interpretable attention patterns that specialize in routing from the letter or word to the final token as shown in Figure 5. **We investigate attention patterns for heads with performance restoration $>= 0.3$ and find no interpretable patterns.** We find that the information is fragmented across multiple heads, which may lead to weak or incomplete aggregation at the final token.

Activation patching supports late-layer suppression. These activation patching results also provide further evidence for the suppressive components identified via the logit lens. While activation patching is less precise at isolating suppression than logit lens analysis, traces of suppressive behavior are still visible. **In particular, for Qwen2.5-3B, head 4 of the final layer (L32 H4) exhibits a consistently negative restoration effect in Figure 5 (with a performance restoration around 10 - 15%)**, indicating that patching in clean acti-

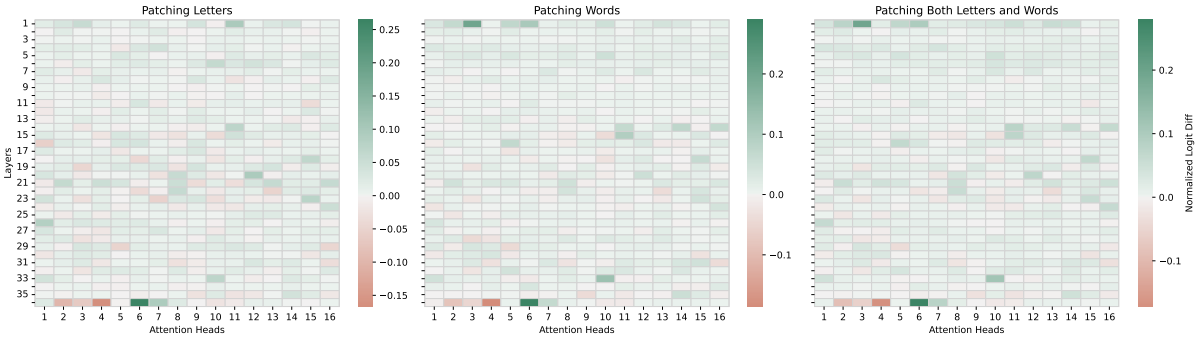


Figure 5: Activation patching heatmaps when corrupting the word, letter or both letter and word for Qwen2.5-3B attention heads averaged over 1000 samples. Green regions indicate components where clean activations restore the correct behavior, helping identify localized circuits for character counting. The Red regions indicate components which diminish performance when patched in.

vations at this head worsens performance rather than improving it. This is consistent with our logit lens finding that the final-layer attention heads contribute approximately 9% suppression of the correct-answer probability mass, and lends additional mechanistic support to the existence of suppressive components concentrated in the final layers.

4.5 Increasing Scale or Post-Training Does Not Help

We extend our experiments to larger models in the 7–9B parameter range and to instruction-tuned variants of all model families, using prompts from Appendix G. The same patterns hold, if not more strongly, across all settings.

Scaling up by roughly 4× does not alleviate the identified failure modes. Logit lens profiles, suppression patterns, and degenerate answer strategies remain consistent between the smaller and larger variants of each family. Instruction tuning does not resolve the issue either. Most instruct models abandon their base strategy and shift toward alternating between counts of "1" and "2", yet accuracy stays near random at around 40% (Appendix H). Qwen2.5-3B-Instruct is the only exception. It retains the "always predict 1" strategy and achieves marginally higher accuracy, though this reflects a more favorable strategy rather than any genuine improvement in counting ability.

Logit lens analysis on these variants reveals the same suppression profile as the smaller base models, with suppression again concentrated in the penultimate and final layer MLP and attention components. The mechanism is therefore not an artifact of model size or training regime but a structural property of how these models process character-level information.

This is striking because even our smaller models reliably encode character-count information in their intermediate representations. Larger models can only encode such orthographic features more strongly, yet the late-layer suppression persists. The bottleneck is not representational capacity but a learned computation that overrides the correct signal at the output stage. This matches reports that even large proprietary models such as GPT-4 fail at character counting without chain-of-thought prompting or tool use, which is what we would expect if the failure is structural rather than a matter of scale.

5 Related Work

Prior work has introduced benchmarks targeting symbolic and character-level reasoning, noting similar deficiencies even in proprietary models like GPT-4 (Achiam et al., 2023) and Llama-3 (Grattafiori et al., 2024a) still underperform on tasks such as counting and position-specific edits (Zhang and He, 2024). LMentry (Efrat et al., 2022) proposes minimal tasks such as letter containment and first/last character identification, while CUTE (Edman et al., 2024) expands this framework to include spelling, character edits, and orthographic similarity. CUTE highlights persistent failures even in instruction-tuned models. Building on these efforts, CWUM (Zhang and He, 2024) provides a comprehensive suite of fifteen character- and word-level manipulation tasks to systematically evaluate these limitations.

6 Conclusion

Models like Llama, Qwen, and Gemma handle character-level information inconsistently. They represent letters well in early layers but often fail to carry that information forward. The problem is not encoding capacity but how the signal is propagated through the network.

We find no attention heads or components dedicated to this processing. The signal is instead scattered and weakly coordinated across layers, which suggests that character-level reasoning is not an explicit feature of these systems but an incidental side effect of training.

This raises a broader question about the linguistic structure these models capture. If they cannot reliably count letters, their reliability on other tasks that demand fine-grained symbolic precision is also in doubt. Such tasks include spelling correction, transliteration, and reasoning over structured text. Future work could explore whether additional training objectives or architectural changes strengthen these low-level capabilities without compromising the higher-level semantics that make these models powerful.

7 Limitations

While this study provides useful insights, it also has several important limitations. First, we only analyzed models in the 2–3B and 7–9B parameter range. These sizes make interpretability experiments more manageable, but they don't tell us whether the same behavior appears in much larger systems like GPT-4 (OpenAI et al., 2024) or Mistral (Jiang et al., 2023b). Second, our evaluation setup uses single-word, controlled inputs, which help isolate specific effects but do not capture the complexity of full sentences or natural text. Third, all experiments were conducted in English, meaning our findings may not generalize to languages with richer morphology or non-alphabetic scripts. Fourth, while methods such as activation patching and probing give a useful window into model internals, they don't guarantee a complete causal picture. The true computation might involve more subtle, distributed interactions that these tools can't fully capture. Finally, our work focuses on identifying failure modes rather than fixing them. We have not yet tested whether fine-tuning, additional supervision, or architectural adjustments could reduce these errors. Exploring such interventions would be a valuable direction for future work.

8 Ethical Considerations

This work focuses on analyzing the internal behavior of large language models (LLMs) through controlled diagnostic tasks. All experiments are performed using open-access models and publicly available datasets, and no sensitive, personal, or private data is used at any stage. The benchmark tasks are synthetic by design and do not involve any real-world user data. The goal of this study is to enhance understanding and transparency in how large models process low-level linguistic information, particularly in cases where they fail on tasks that appear trivial to humans. By identifying such systematic failure modes, we aim to contribute to the responsible development and deployment of LLMs, offering insights that may inform future improvements in architecture, training, and evaluation. This work does not involve human participants, crowdsourced annotations, or generation of sensitive content, and therefore presents minimal ethical or societal risk. All the models used in this work are publicly available on Huggingface and free for research. The datasets are licensed under Creative Commons by Non-Commercial 4.0 (CC by NC 4.0). No anticipated risks are associated with using the data from these provided links.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *Preprint*, arXiv:1607.06450.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, and 1 others. 2024. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53.

Arthur Conmy, Augustine Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. 2023a. Towards automated circuit discovery for mechanistic interpretability. In *Advances in*

Neural Information Processing Systems, volume 36, pages 16318–16352. Curran Associates, Inc.

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

Lukas Edman, Helmut Schmid, and Alexander Fraser. 2024. Cute: Measuring llms' understanding of their tokens. *arXiv preprint arXiv:2409.15452*.

Avia Efrat, Or Honovich, and Omer Levy. 2022. Lmentry: A language model benchmark of elementary language tasks. *arXiv preprint arXiv:2211.02069*.

Atticus Geiger, Hanson Lu, Thomas Icard, and Christopher Potts. 2021. Causal abstractions of neural networks. *Advances in Neural Information Processing Systems*, 34:9574–9586.

Atticus Geiger, Kyle Richardson, and Christopher Potts. 2020. Neural natural language inference models partially embed theories of lexical entailment and negation. *arXiv preprint arXiv:2004.14623*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024a. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024b. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Michael Hanna, Ollie Liu, and Alexandre Variengien. 2023. How does gpt-2 compute greater-than?: Interpreting mathematical abilities in a pre-trained language model. *Advances in Neural Information Processing Systems*, 36:76033–76060.

Peter Hase, Mohit Bansal, Been Kim, and Asma Ghandeharioun. 2023. Does localization inform editing? surprising differences in causality-based localization vs. knowledge editing in language models. *Advances in Neural Information Processing Systems*, 36:17643–17668.

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

John Hewitt and Percy Liang. 2019. Designing and interpreting probes with control tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2733–2743, Hong Kong, China. Association for Computational Linguistics.

Dieuwke Hupkes, Sara Veldhoen, and Willem Zuidema. 2018. Visualisation and 'diagnostic classifiers' reveal how recurrent and recursive neural networks process hierarchical structure. *Preprint*, arXiv:1711.10203.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, and 1 others. 2023a. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023b. Mistral 7b. *Preprint*, arXiv:2310.06825.

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A method for stochastic optimization. *Preprint*, arXiv:1412.6980.

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.

George A. Miller. 1992. WordNet: A lexical database for English. In *Speech and Natural Language: Proceedings of a Workshop Held at Harriman, New York, February 23-26, 1992*.

nostalgebraist. 2020. Interpreting GPT: The Logit Lens. LessWrong blog post.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Tiago Pimentel, Josef Valvoda, Rowan Hall Maudslay, Ran Zmigrod, Adina Williams, and Ryan Cotterell. 2020. Information-theoretic probing for linguistic structure. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4609–4622, Online. Association for Computational Linguistics.

Andrew Shin and Kunitake Kaneko. 2024. Large language models lack understanding of character composition of words. *arXiv preprint arXiv:2405.11357*.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, and 1 others. 2024a. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, Johan Ferret, Peter Liu, Pouya Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, and 179 others. 2024b. Gemma 2: Improving open language models at a practical size. *Preprint*, arXiv:2408.00118.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart Shieber. 2020. Investigating gender bias in language models using causal mediation analysis. *Advances in neural information processing systems*, 33:12388–12401.

Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. 2022. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. *arXiv preprint arXiv:2211.00593*.

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*.

Sarah Wiegreffe, Oyvind Tafjord, Yonatan Belinkov, Hannaneh Hajishirzi, and Ashish Sabharwal. 2025. Answer, assemble, ace: Understanding how lms answer multiple choice questions. *Preprint*, arXiv:2407.15018.

Wikimedia Foundation. 2024. Wiktionary: The free dictionary. `https://dumps.wikimedia.org/enwiktionary/`. Accessed: 2024.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and 1 others. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 43 others. 2024. Qwen2 technical report. *Preprint*, arXiv:2407.10671.

Biao Zhang and Rico Sennrich. 2019. Root mean square layer normalization. *Preprint*, arXiv:1910.07467.

Fred Zhang and Neel Nanda. 2023. Towards best practices of activation patching in language models: Metrics and methods. *arXiv preprint arXiv:2309.16042*.

Yidan Zhang and Zhenan He. 2024. Large language models can not perform well in understanding and manipulating natural language at both character and word levels? In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 11826–11842.

Appendix

A On the Inapplicability of Control Tasks to Our Probing Setting

Hewitt and Liang (2019) propose *control tasks* as a diagnostic for probing experiments. They associate word types with random outputs and measure whether a probe succeeds or whether it only memorizes the word-type-to-label mapping. The resulting selectivity metric is the gap between linguistic task accuracy and control task accuracy, and is intended to guard against probes that solve tasks through memorization rather than representation decoding. While the validity of this framework has itself been questioned (Pimentel et al., 2020), we argue that control tasks are ill-suited to our setting, and that the concern they address simply does not arise here.

The core worry behind control tasks is that a probe might learn to perform a linguistic task such as POS tagging or dependency relation prediction by exploiting the correlation between word identity and the typical label associated with that word type. This failure mode is plausible because such tasks allow for memorizable surface-form-to-label associations.

Our task has a fundamentally different structure. Each probe is trained for a specific target letter, meaning the mapping from word type to count is deterministic and fixed for a given probe. However, our dataset construction explicitly ensures that no word or token type seen during probe training appears in the validation or test sets. This type-disjoint split means that even if a probe attempted to memorize word-type-to-count associations, it would have no basis to do so for unseen types, and would achieve chance performance on the test set. The concern warranting for a control probe is eliminated by our dataset design, following the approach advocated by Hupkes et al. (2018) and acknowledged as a valid mitigation strategy by Hewitt and Liang (2019) themselves.

This guarantee is especially robust for multi-token words. When a word is split into multiple subword tokens, the correct character count depends on the specific combination of tokens and the target letter, a combination that is not repeated across splits. Novel words at test time present entirely new token combinations with counts never encountered during training, making memorization structurally impossible.

A further property that distinguishes our setting is that the correct output cannot be inferred from token identifiers alone. Character count is an orthographic property of the word's surface form, information that is not directly encoded in token IDs and cannot be recovered without access to the actual character sequence. A probe operating purely on token identity, without genuinely decoding character-level features from the hidden representation, has no basis on which to perform the task.

Finally, we use the simplest possible probe architecture: a single linear layer without bias (Equation 1), which has the lowest possible capacity for memorization and is known to yield high selectivity even in standard probing settings (Hewitt and Liang, 2019).

Together, these properties, ensure that our probing results reflect genuine representation content rather than superficial memorization.

B Connection Between Cross-Entropy Loss and Logit Difference

The logit difference metric Δ is closely related to the standard cross-entropy loss used during training. Specifically, taking differences in cross-entropy loss between two tokens is equivalent to taking their logit difference, as the softmax normalizer cancels out.

Recall that the cross-entropy loss for a token a is:

$$-\log P(a) = -\log\left(\frac{e^{\text{logit}(a)}}{\sum_{v\in V} e^{\text{logit}(v)}}\right) \tag{9}$$

Expanding the logarithm:

$$-\log P(a) = -\log\left(e^{\text{logit}(a)}\right) + \log\left(\sum_{v\in V} e^{\text{logit}(v)}\right) \tag{10}$$

$$-\log P(a) = -\left(\text{logit}(a) - \log\left(\sum_{v\in V} e^{\text{logit}(v)}\right)\right) \tag{11}$$

Now consider the difference in cross-entropy loss between two tokens a and b:

$$-\log P(a) - (-\log P(b)) = -\text{logit}(a) + \text{logit}(b) \tag{12}$$

The log-sum-exp term $\log\left(\sum_{v \in V} e^{\text{logit}(v)}\right)$ cancels exactly, leaving a pure logit difference. Consequently, $\Delta > 0$ corresponds precisely to the correct token having a lower cross-entropy loss than the competing token, providing a direct link between our interpretability metric and the training objective.

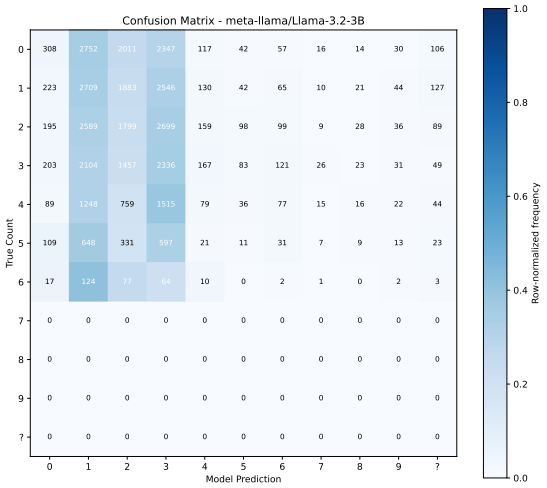
C Choice of Metric for Activation Patching

We use Δ rather than Δ_p for the activation patching metric for two reasons. First, as noted in Section 3.2.1, Δ is linearly related to the residual stream, which makes it directly interpretable as a measure of each component's additive contribution to the model's preference for the correct token. Second, probabilities are poorly suited for measuring restoration: the softmax transformation compresses probability mass non-linearly, causing probabilities to decay very rapidly once a token falls out of the top predictions. This means that Δ_p is disproportionately sensitive to small logit changes in regions where the correct token is already suppressed, producing a noisy and poorly calibrated restoration signal. The logit scale, being unbounded and logarithmic relative to probability, varies more smoothly across the range of patching conditions and yields a more stable normalized restoration score. We therefore reserve Δ_p for quantifying suppression at the output layer, where its normalized scale aids interpretability, and rely on Δ throughout the activation patching analysis.

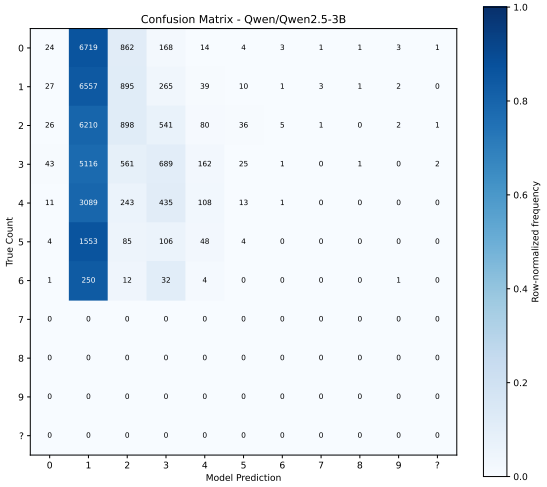
D Confusion Matrices and Answer Strategies

Inspecting the confusion matrices of model predictions reveals two distinct degenerate answer strategies that account for the poor overall performance observed in Section 4.1.

The two strategies reflect qualitatively different failure modes. Llama-3.2-3B distributes its predictions roughly uniformly across possible counts, suggesting that the model does not develop any stable preference and effectively ignores the input when producing an answer. This explains why its accuracy sits close to the random baseline in Figure 2. Qwen2.5-3B, by contrast, adopts a highly deterministic but incorrect strategy, defaulting to predicting a count of "1" in the vast majority of cases. This reflects a strong learned prior toward low counts, likely reinforced during training on natural language where single letter occurrences are most common. Gemma-2 models exhibit behavior qualitatively similar to Qwen, also defaulting to constant low-count predictions. These observations confirm that the models are not attempting to perform genuine counting, but are instead falling back on simple heuristics, consistent with the suppression of correct intermediate representations identified in our mechanistic analysis.



(a) Llama-3.2-3B exhibits a near-uniform prediction distribution across all count values, corresponding to a effectively random answer strategy that is agnostic to the input.



(b) Qwen2.5-3B overwhelmingly predicts the count "1" regardless of the true count, corresponding to a degenerate constant prediction strategy.

Figure 6: Confusion matrices for predicted versus true letter counts. The two models represent the two dominant failure modes observed across all model families: a random strategy (Llama) and a constant prediction strategy (Qwen).

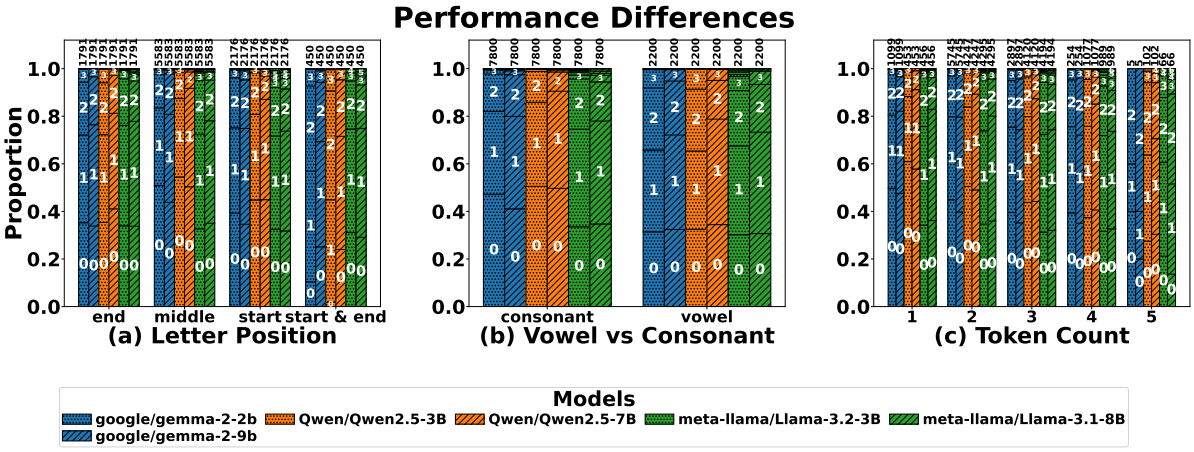


Figure 7: Distributions of absolute deviations between the correct letter count and the predicted count ($|\text{predicted count} - \text{correct count}|$). The y-axis indicates the proportion of predictions at each deviation. Every stacked bar represents a distribution of deviations. Bars labelled with values above zero reflect over- or under-counting errors relative to the true letter count and the bars labelled deviation $= 0$ represent the accuracy (no deviation). The number on top of every bar denotes how many instances of the category were present.

E Performance Analysis

We perform a performance analysis across different parameters. We find the following effects:

Effect of Letter Position We evaluate how performance varies depending on whether the target letter appears at the start, end, or both ends of a word, compared to when it occurs in the middle. As shown in Figure 7 (a), models consistently struggle more with letters at the extremes. This difficulty is reflected in the higher proportion of zero-correct predictions (bars labeled 0) for start, end, and start–end positions. The performance drop is especially noticeable for Qwen (blue) and Gemma (orange), whereas Llama remains close to random performance across all positions.

Vowel vs. Consonant Counting We compare model performance on counting vowels versus consonants. As shown in Figure 7 (b), models generally struggle more with vowel counting. This is evident from the higher proportion of zero-correct predictions (bars labeled 0) in the vowel condition. The performance drop is particularly for Qwen (blue) and Gemma (orange), while Llama shows consistently low accuracy, remaining close to random performance.

Token Counts We also examine how performance varies with the number of tokens that compose a single word. As shown in Figure 7 (c), models tend to perform better when words are represented by fewer subword tokens. This trend is evident from the decline in accuracy (bars labelled 0) as token count increases. In other words, when a word is split into more subword units, models struggle more with accurately counting its letters.

F Per-Letter Probe Accuracy Heatmaps

The probing results presented in Section 4.2 average accuracy across all target letters. Here we present the full per-letter breakdown as heatmaps, where each row corresponds to a layer and each column corresponds to a specific target letter. This allows us to inspect whether certain letters are encoded more reliably than others, and whether the layer at which encoding peaks varies across letters.

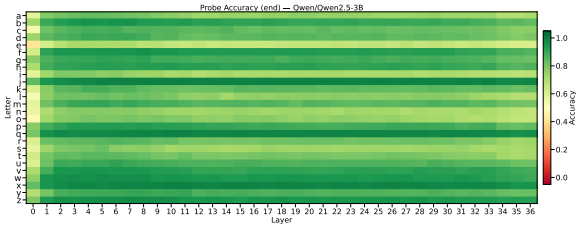


Figure 8: Layerwise probe accuracy heatmap for individual target letters at the final token of the word for Qwen2.5-3B. Each cell represents the probe accuracy for a specific layer and target letter.

Overall, the heatmaps confirm that the high average probe accuracy reported in the main text is not driven by a small subset of easy letters. Character-count information is broadly and consistently encoded across the alphabet, with most letters achieving high accuracy by the mid layers and maintain-

Character Counting Prompts

We used the following natural language prompts to query the models for character counting. Each prompt contains placeholders `<letter>` and `<count_subject>`, which were replaced with the target character and input string respectively.

1. How many `<letter>`'s are in `<count_subject>`?
   Expected response: The number is `<count>`.
2. Can you count the letter `<letter>` in `<count_subject>`?
   Expected response: The count is `<count>`.
3. Tell me how many times `<letter>` appears in `<count_subject>`.
   Expected response: The number is `<count>`.
4. What is the count of `<letter>` in `<count_subject>`?
   Expected response: The count is `<count>`.
5. Please calculate the total number of `<letter>`'s in `<count_subject>`.
   Expected response: The total number is `<count>`.
6. Could you find out how many `<letter>`'s are in `<count_subject>`?
   Expected response: The number is `<count>`.
7. I want to know the frequency of the letter `<letter>` in `<count_subject>`.
   Expected response: The frequency is `<count>`.
8. Count all the `<letter>`'s present in `<count_subject>`.
   Expected response: The total number is `<count>`.
9. How often does `<letter>` appear in `<count_subject>`?
   Expected response: The number of times is `<count>`.

Figure 9: Prompt templates used for the character counting task. Each template was evaluated under the same experimental setting.

ing it through to the final token of the word.

G Prompts for Instruction-tuned Models

We construct User and Assistant Turns for Instruction Tuned Models. We use the model's chat templates to get properly formatted text (Including the default system prompts). We show the prompts in Figure 9.

H Overall Performance of Instruction-Tuned Models

Instruction-tuned models remain close to random performance on the character counting task, indicating that post-training does not resolve the underlying failure modes identified in our mechanistic analysis. However, inspecting the confusion matrices reveals that post-training does shift the answer strategies adopted by most models. Nearly all instruct variants abandon their base model strategy in favor of alternating between counts of "1" and "2" as the dominant prediction, suggesting that instruction tuning alters the surface behavior without fixing the underlying representational bottleneck.

Qwen2.5-3B-Instruct is a notable exception. It retains the "always predict 1" strategy of its base counterpart in the majority of cases, and as a result achieves slightly better accuracy than the other instruct models, since predicting "1" is correct more often than a uniform random strategy given the natural distribution of letter counts in words. This further illustrates that the marginal performance differences between models reflect differences in

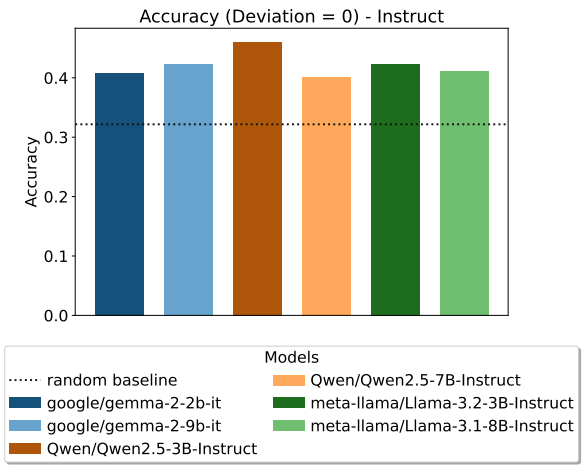


Figure 10: Accuracy of predicted letter counts for instruction-tuned models. The dotted line indicates the random baseline.

answer strategy rather than any genuine improvement in character-level reasoning.

I Experiments with LLaMa models

To assess the generality of our observations across model scales, we replicate key experiments on Llama-3.1-8B, Llama-3.1-8B-Instruct, Llama-3.2-3B, and Llama-3.2-3B-Instruct.

I.1 Activation Patching

We perform activation patching as described in Section 3.2.3. Figures 11 show restoration heatmaps for word-level, letter-level, and both word and letter level corruptions across residual streams, attention heads, and MLP blocks for Llama Models. Figure 12 shows the restoration effects on a per-attention head basis.

J Experiments with Gemma-2

To examine the generality of our findings across model scales, we replicate key experiments on Gemma-2-2B and Gemma-2-9B, as well as their aligned instruction-tuned counterparts, Gemma-2-2B-IT and Gemma-2-9B-IT.

J.1 Activation Patching

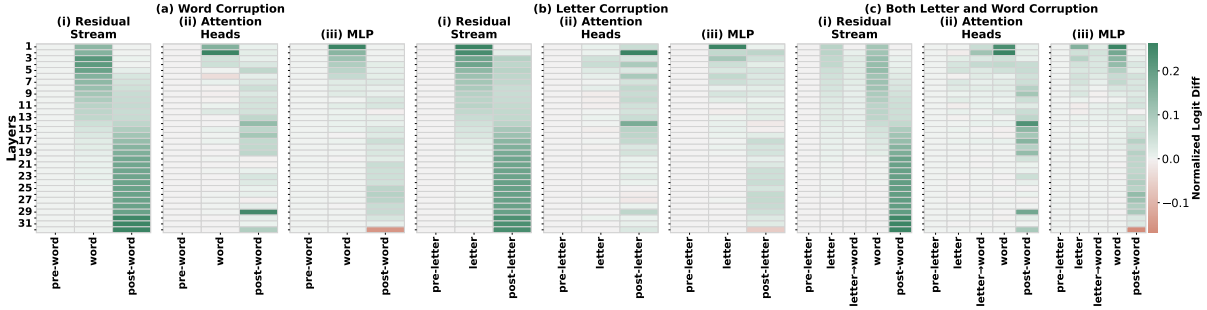
We perform activation patching as described in Section 3.2.3. Figure 13 shows restoration heatmaps for word-level, letter-level, and both word and letter level corruptions across residual streams, attention heads, and MLP blocks for Gemma models, and Figure 14 shows the restoration effects on a per-attention head basis.

K Experiments with Qwen-2.5

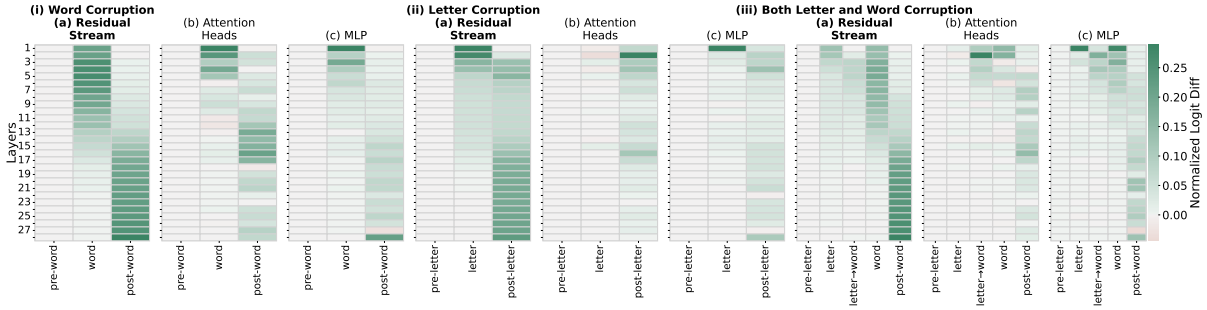
To examine the generality of our findings across model scales, we replicate key experiments on the 3B and 7B parameter models (Qwen2.5-3B, Qwen2.5-7B) along with their instruction-tuned counterparts (Qwen2.5-3B-Instruct, Qwen2.5-7B-Instruct).

K.1 Activation Patching

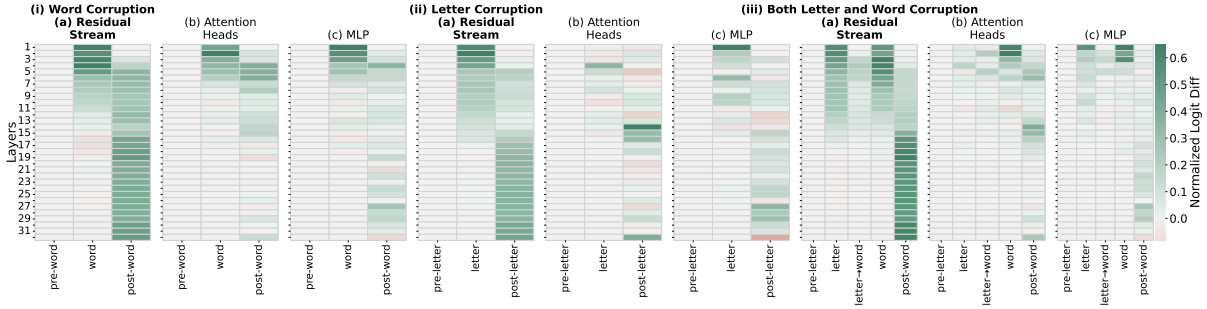
We perform activation patching as described in Section 3.2.3. Figure 15, shows restoration heatmaps for word-level, letter-level, and both word and letter level corruptions across residual streams, attention heads, and MLP blocks for Qwen2.5 Models and Figure 16 shows the restoration effects on a per-attention head basis.



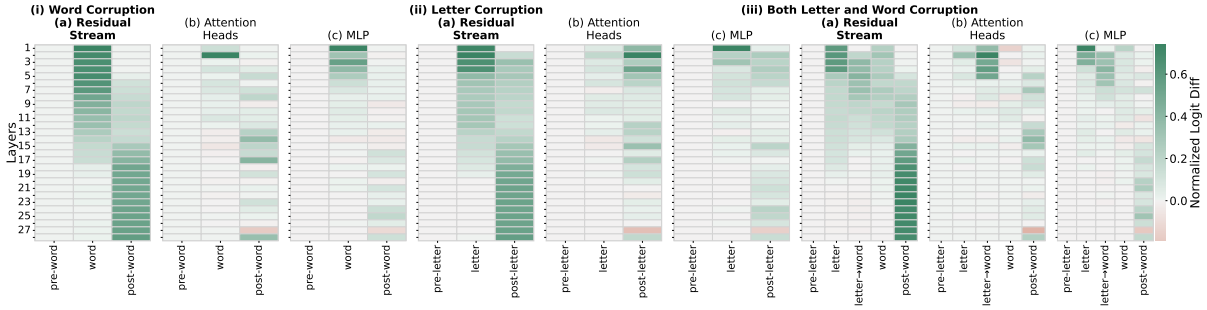
(a) Llama-3.1-8B



(b) Llama-3.2-3B

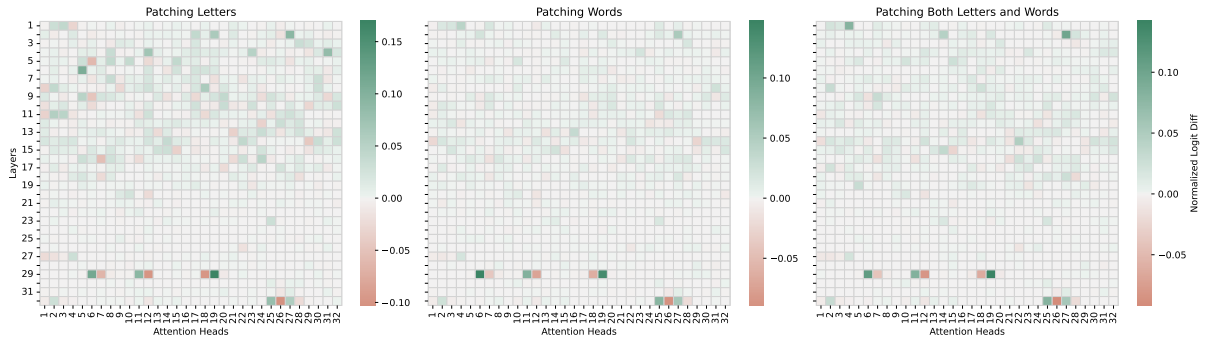


(c) Llama-3.1-8B-Instruct

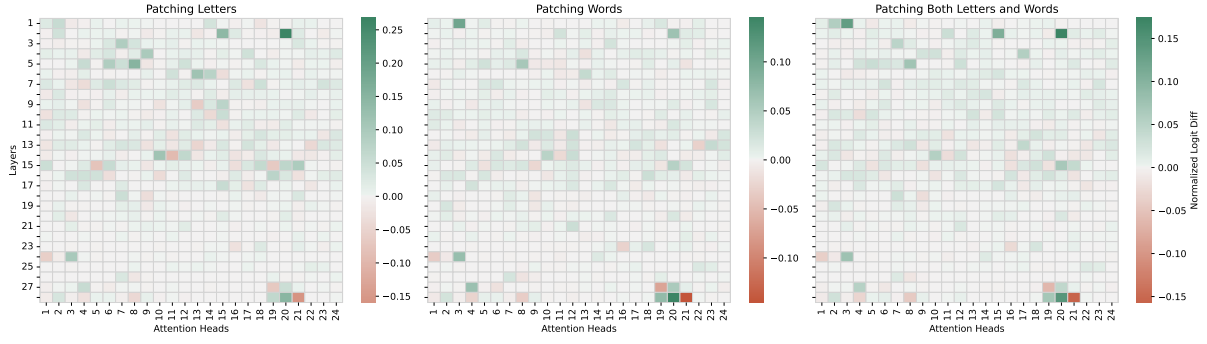


(d) Llama-3.2-3B-Instruct

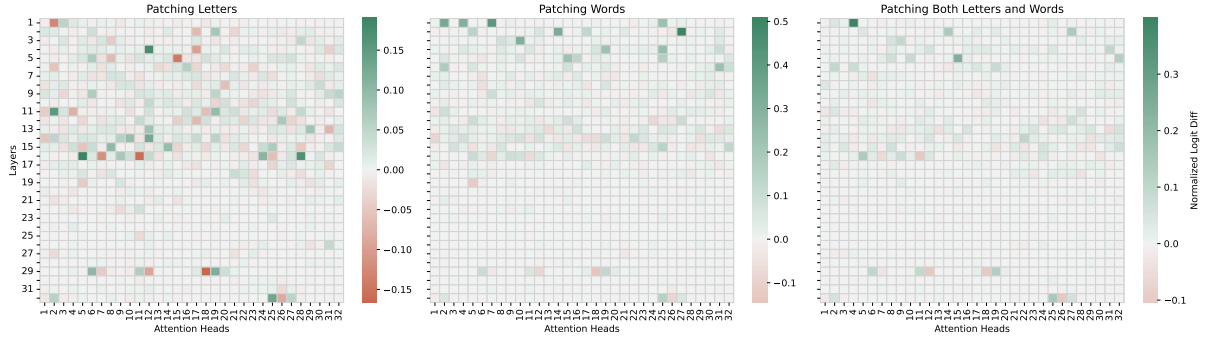
Figure 11: Activation patching heatmaps for Llama Models averaged over 100 samples. Green regions indicate components where clean activations restore the correct behavior, helping identify localized circuits for character counting.



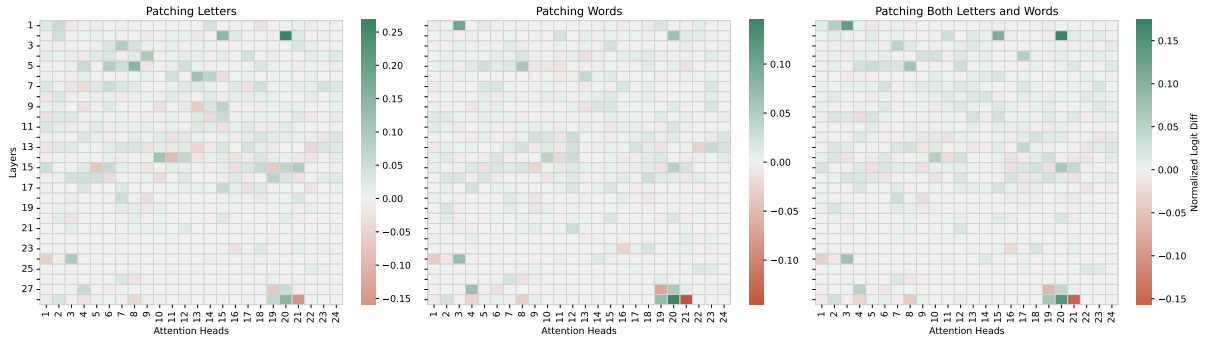
(a) Llama-3.1-8B



(b) Llama-3.2-3B

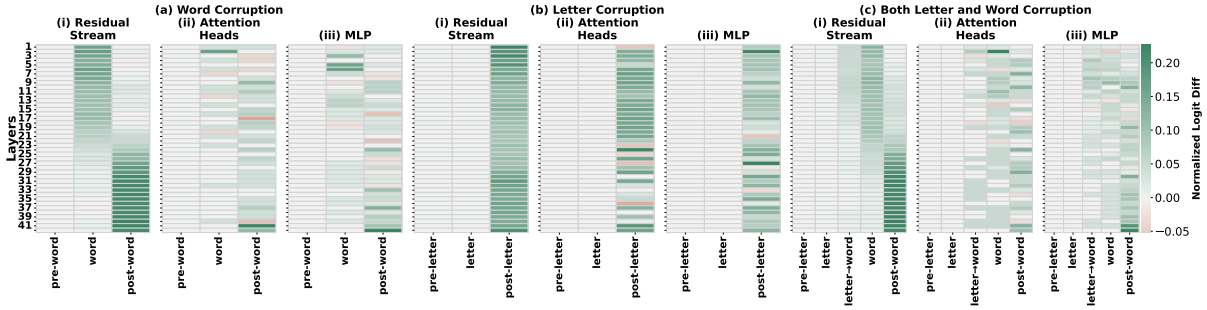


(c) Llama-3.1-8B-Instruct

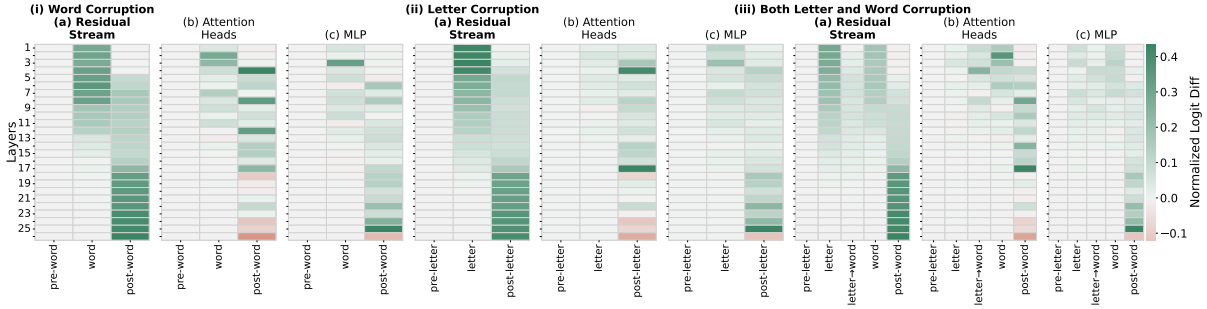


(d) Llama-3.2-3B-Instruct

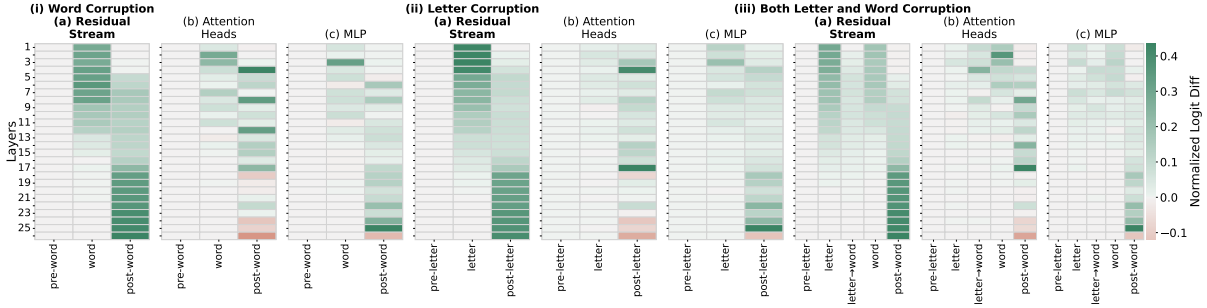
Figure 12: Activation patching heatmaps when corrupting the word, letter or both letter and word for Llama attention heads averaged over 100 samples. Green regions indicate components where clean activations restore the correct behavior, helping identify localized circuits for character counting. The Red regions indicate components which diminish performance when patched in.



(a) Gemma2-9B

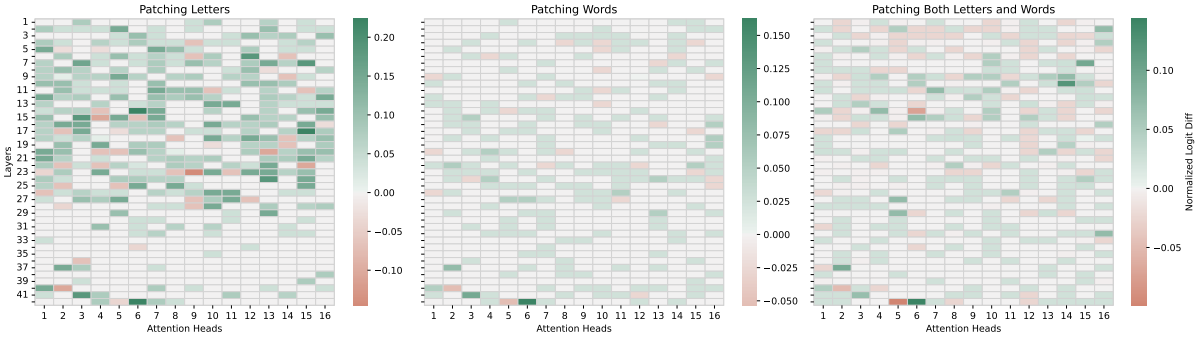


(b) Gemma2-2B

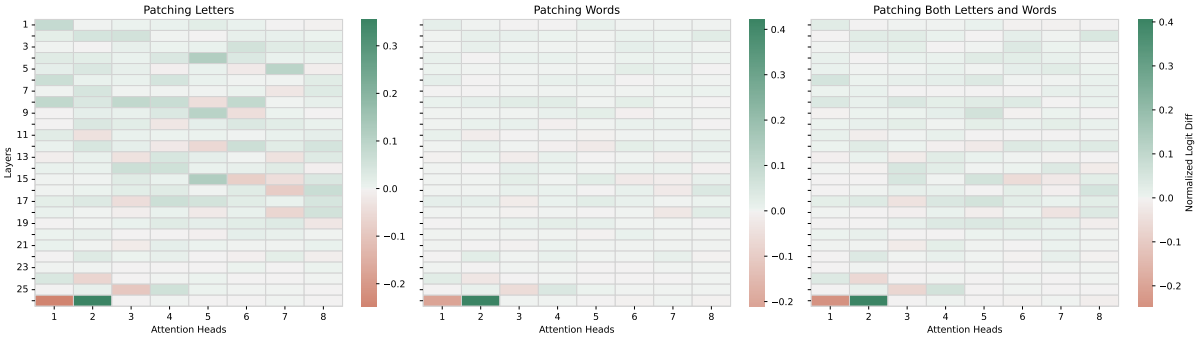


(c) Gemma2-2B-Instruct

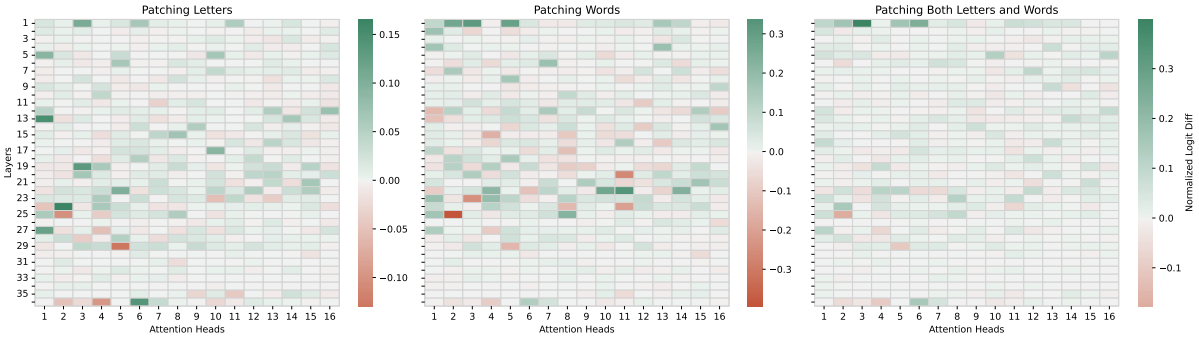
Figure 13: Activation patching heatmaps for Gemma Models averaged over 100 samples. Green regions indicate components where clean activations restore the correct behavior, helping identify localized circuits for character counting.



(a) Gemma2-9B

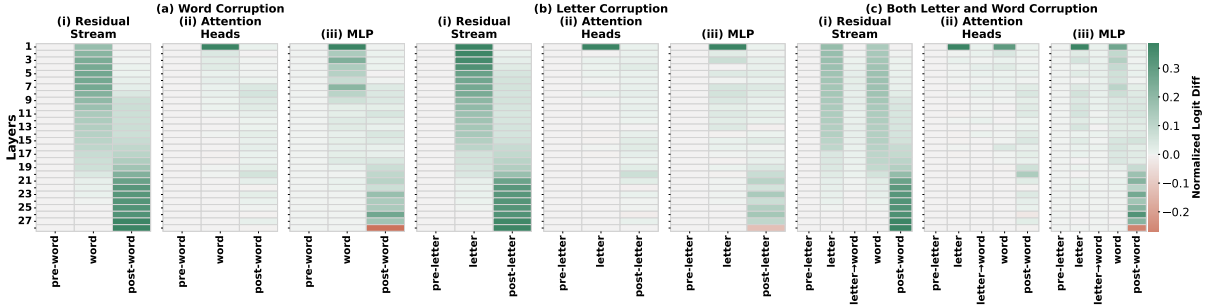


(b) Gemma2-2B

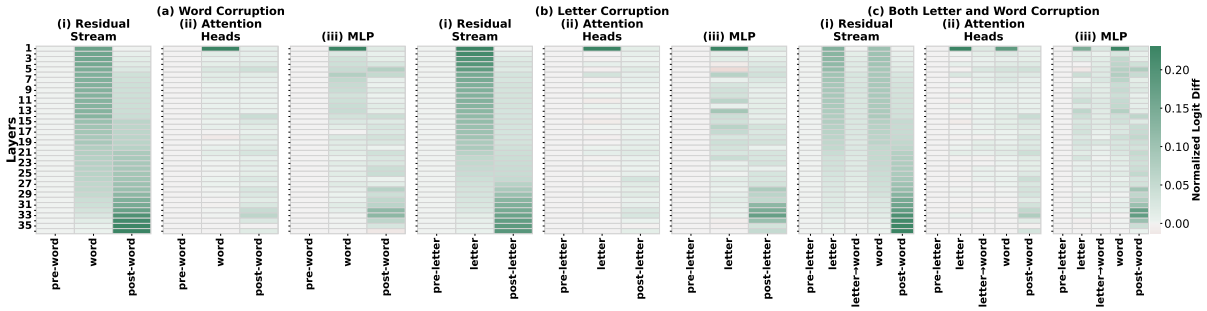


(c) Gemma2-2B-Instruct

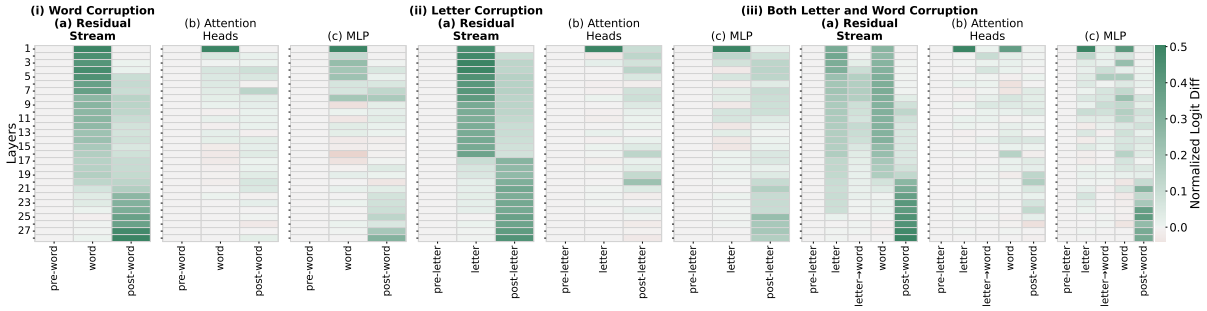
Figure 14: Activation patching heatmaps when corrupting the word, letter or both letter and word for Gemma attention heads averaged over 100 samples. Green regions indicate components where clean activations restore the correct behavior, helping identify localized circuits for character counting. The Red regions indicate components which diminish performance when patched in.



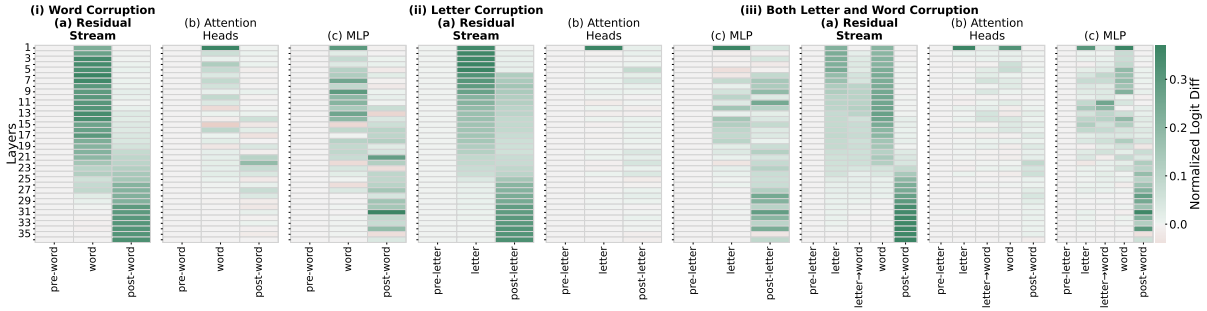
(a) Qwen2.5-7B



(b) Qwen2.5-3B

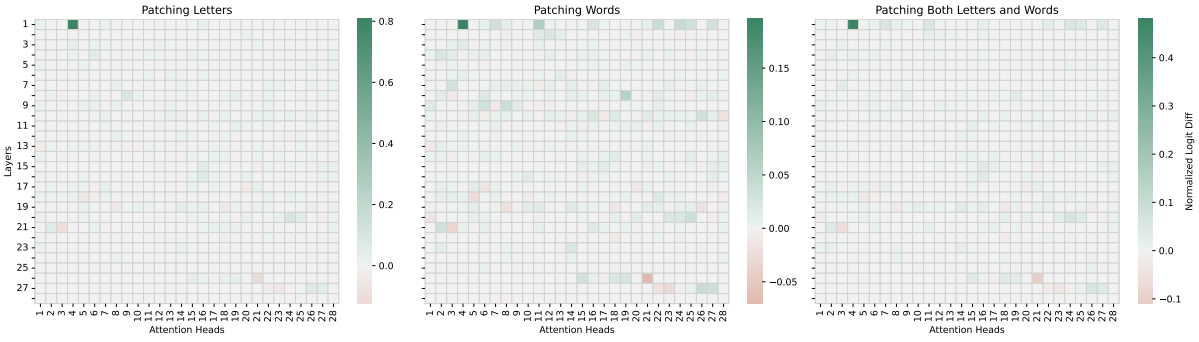


(c) Qwen2.5-7B-Instruct

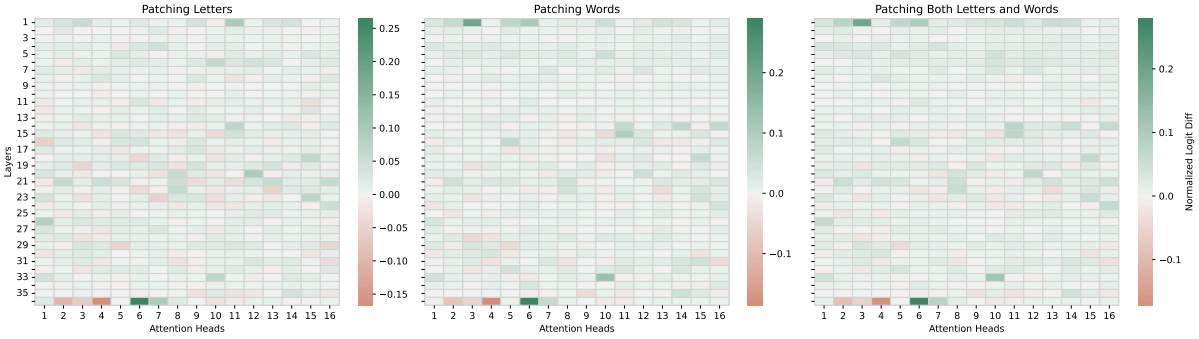


(d) Qwen2.5-3B-Instruct

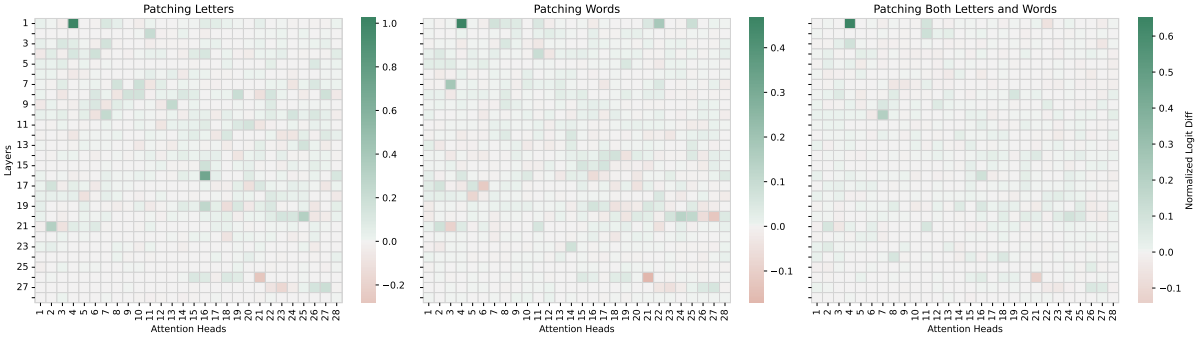
Figure 15: Activation patching heatmaps for Qwen Models averaged over 100 samples. Green regions indicate components where clean activations restore the correct behavior, helping identify localized circuits for character counting.



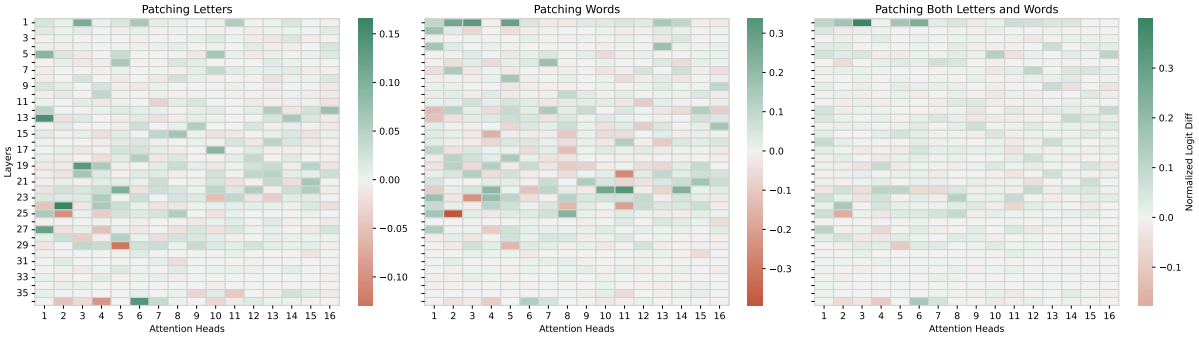
(a) Qwen2.5-7B



(b) Qwen2.5-3B



(c) Qwen2.5-7B-Instruct



(d) Qwen2.5-3B-Instruct

Figure 16: Activation patching heatmaps when corrupting the word, letter or both letter and word for Llama attention heads averaged over 100 samples. Green regions indicate components where clean activations restore the correct behavior, helping identify localized circuits for character counting. The Red regions indicate components which diminish performance when patched in.