Look Before You Leap: A Universal Emergent Decomposition of Retrieval Tasks in Language Models

March 5, 2024

Look Before You Leap: A Universal Emergent Decomposition of Retrieval Tasks in Language Models

Alexandre Variengien and Eric Winsor

https://arxiv.org/abs/2312.10091

Look Before You Leap: A Universal Emergent Decomposition of Retrieval Tasks in Language Models

Alexandre Variengien*
École Normale Supérieure de Lyon
École Polytechnique Fédérale de Lausanne

Eric Winsor
Conjecture

Abstract

When solving challenging problems, language models (LMs) are able to identify relevant information from long and complicated contexts. To study how LMs solve retrieval tasks in diverse situations, we introduce ORION, a collection of structured retrieval tasks spanning six domains, from text understanding to coding. Each task in ORION can be represented abstractly by a request (e.g. a question) that retrieves an attribute (e.g. the character name) from a context (e.g. a story). We apply causal analysis on 18 open-source language models with sizes ranging from 125 million to 70 billion parameters. We find that LMs internally decompose retrieval tasks in a modular way: middle layers at the last token position process the request, while late layers retrieve the correct entity from the context. After causally enforcing this decomposition, models are still able to solve the original task, preserving 70% of the original correct token probability in 98 of the 106 studied model-task pairs. We connect our macroscopic decomposition with a microscopic description by performing a fine-grained case study of a question-answering task on Pythia-2.8b. Building on our high-level understanding, we demonstrate a proof of concept application for scalable internal oversight of LMs to mitigate prompt-injection while requiring human supervision on only a single input. Our solution improves accuracy drastically (15.5% → 97.5% on Pythia-12b). This work presents evidence of a universal emergent modular processing of tasks across varied domains and models and is a pioneering effort in applying interpretability for scalable internal oversight of LMs. Code available at https://github.com/aVariengien/causal-checker.

1 Introduction

Recent advances in language models (LMs) (Vaswani et al., 2017) have demonstrated their flexible problem-solving abilities and their expert-level knowledge in a wide range of fields (Bubeck et al., 2023; OpenAI, 2023). Researchers have developed a series of techniques such as fine-tuning (Ouyang et al., 2022) and Reinforcement Learning from Human Feedback (RLHF) (Ouyang et al., 2022) to ensure models output honest and helpful answer. However, as their abilities reach human level, supervision from human feedback becomes costly and even impossible. This necessitates more efficient or automated methods of supervision, known generally as *scalable oversight*.

Moreover, existing methods only control for the output of the model while leaving the internals of the model unexamined (Casper et al., 2023; Ngo et al., 2023). This is a critical limitation as many internal processes can elicit the same output while using trustworthy or untrustworthy mechanisms. For instance, we would like to know whether a model answers faithfully based on available information or simply gives a users' preferred answer (Perez et al., 2022). We call this problem *internal oversight*.

*Work done during an internship at Conjecture. Correspondence to alexandre.variengien@gmail.com.

Look Before You Leap overview/outline [1]

- Studied macroscopic level properties of retrieval tasks in LLMs
- Retrieval tasks can be represented abstractly by:
  - a request (e.g. a question)
  - that retrieves an attribute (e.g. the character name)
  - from a context (e.g. a story)
- First, create a dataset of 15 different tasks in 6 different domains, with varied details for each example
- Baseline performance on 18 different models from 4 LLM families
  - GPT-2, Pythia, Falcon, and Llama 2
  - From 125M up to 70B parameters

Look Before You Leap overview/outline [2]

- Perform *residual stream patching* on the last token
  - Try every possible layer, one at a time
  - Replace the residual stream vector of values with those from a separate inference run, which had a different context and a different request
- Find that LMs internally decompose retrieval tasks in a modular way:
  - Middle layers at the last token position process the request
  - Late layers retrieve the correct entity from the context
- Middle layer residual stream patching works on all models
- Middle layer residual stream patching works on all domains/tasks
  - Only abstract induction (much easier task) on large models was weak

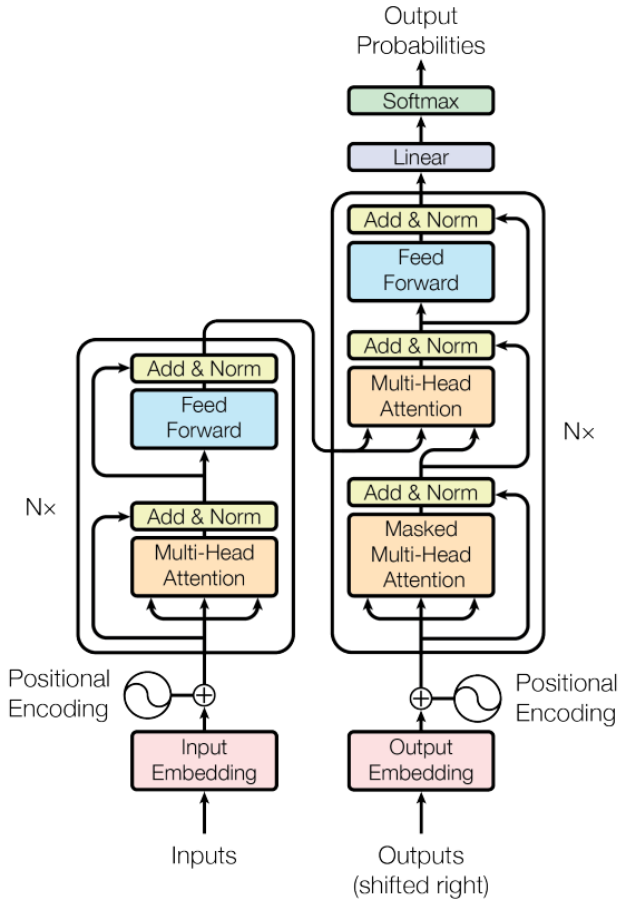
Look Before You Leap overview/outline [3]

- Further experiment (Pythia 2.8b) showed that microscopic analysis of individual attention heads could *not* produce similar consistent patterns
  - For example, a few attention heads seemed to answer requests about cities only for a particular city
  - Although multiple different microscopic mechanisms seem to work in parallel, they seem to have a common portion of the LLM where they were located (perhaps akin to an organ in an animal's body) that performs the request and the retrieval functions
- Also perform a proof of concept of oversight of retrieval against prompt injection attacks using residual stream patching

Look Before You Leap & Attention data flow

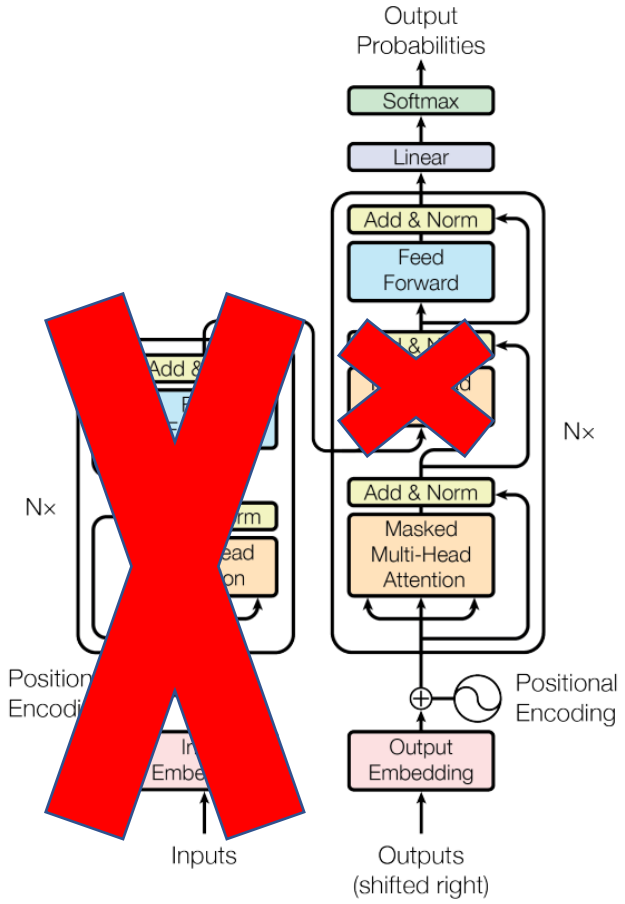
- We try to crystallize understanding of the flow of information across token positions and across layers
- Starting with the original attention diagram, we step-by-step make small modifications until we have a diagram with the same orientation as the diagrams used in this paper
- To understand residual stream patching, it is critical to understand how information can only flow from earlier tokens to later tokens, and from earlier layers to later layers

Original Transformer

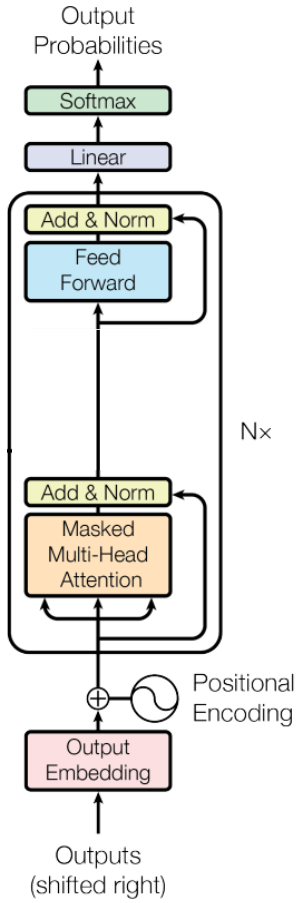


"*Attention Is All You Need*," Vaswani et al., https://arxiv.org/abs/1706.03762

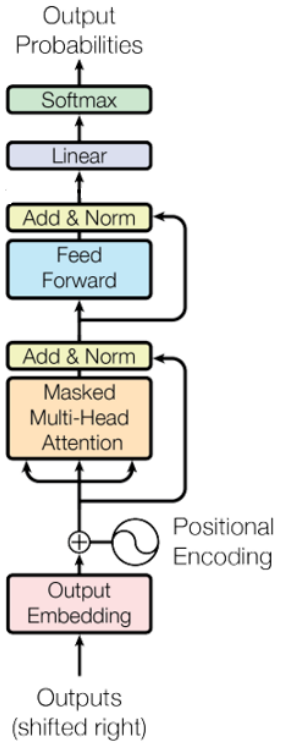
Decoder-only Transformer – 1



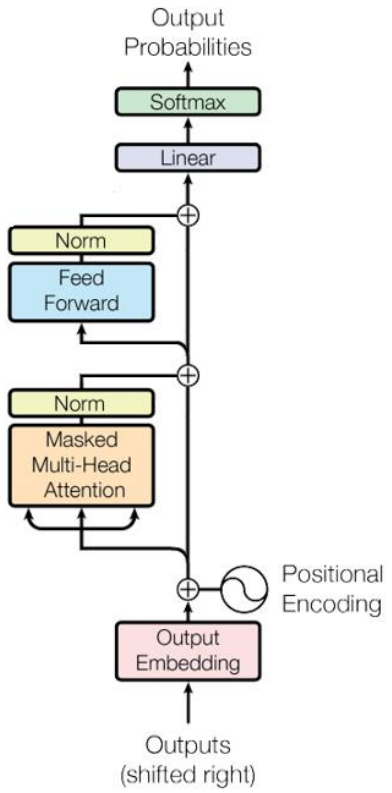
Decoder-only Transformer – 2



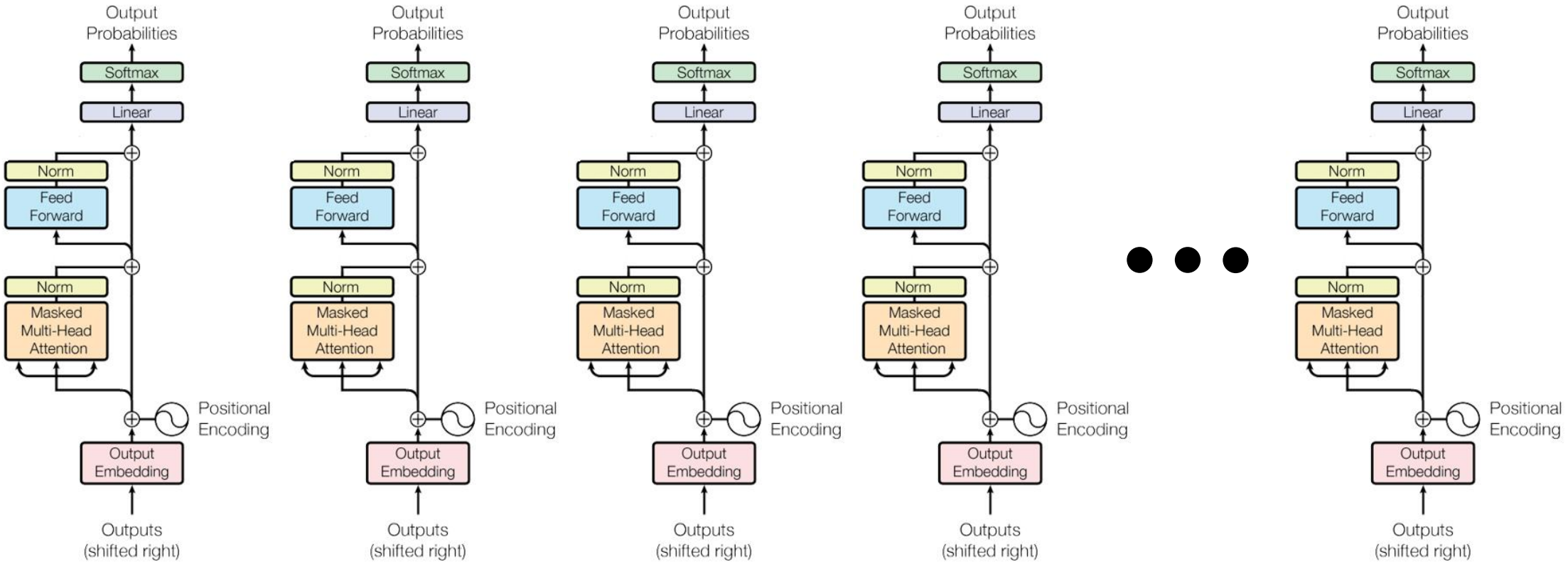
Decoder-only Transformer, Compact



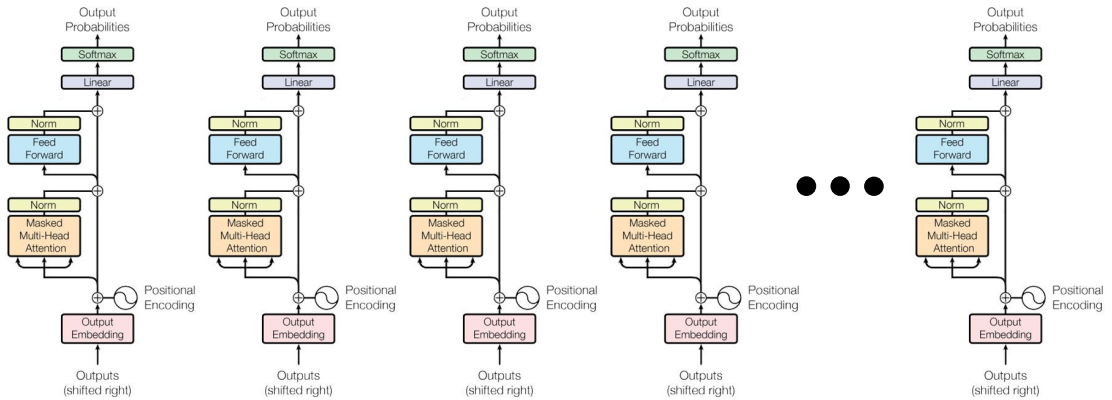
Decoder-only Transformer, Straight Through



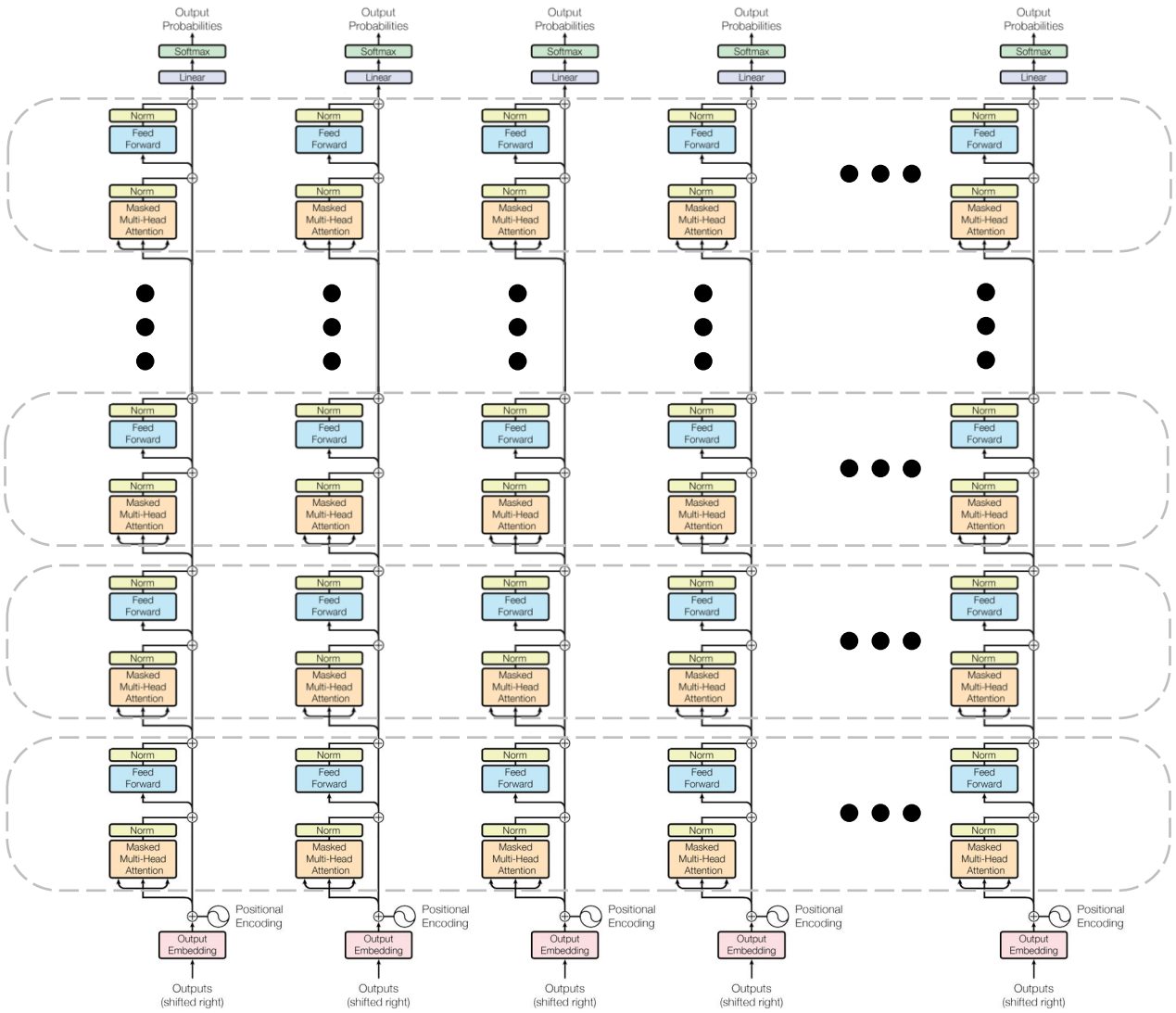
Decoder, Multiple Tokens



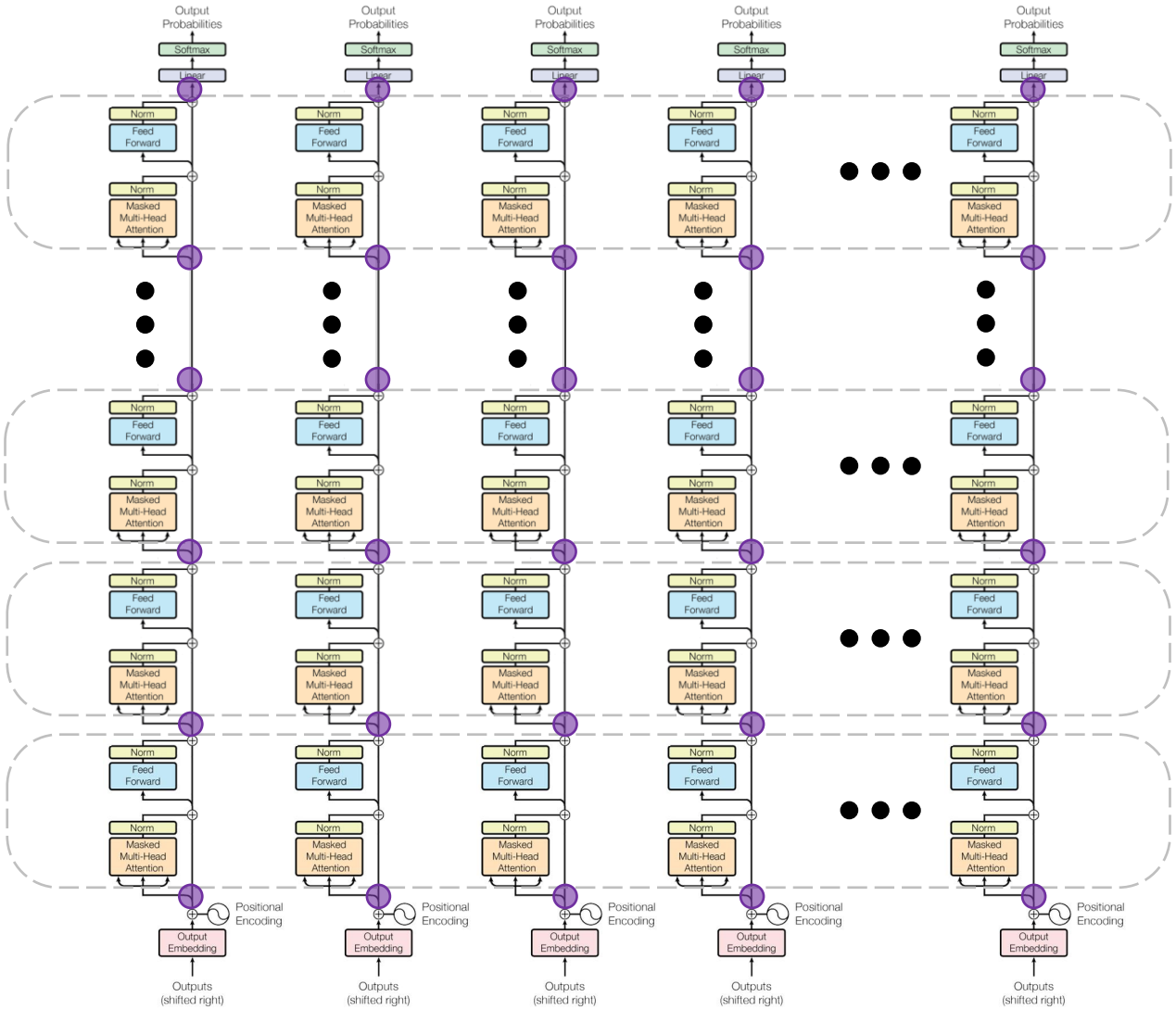
Decoder, Multiple Tokens, Small



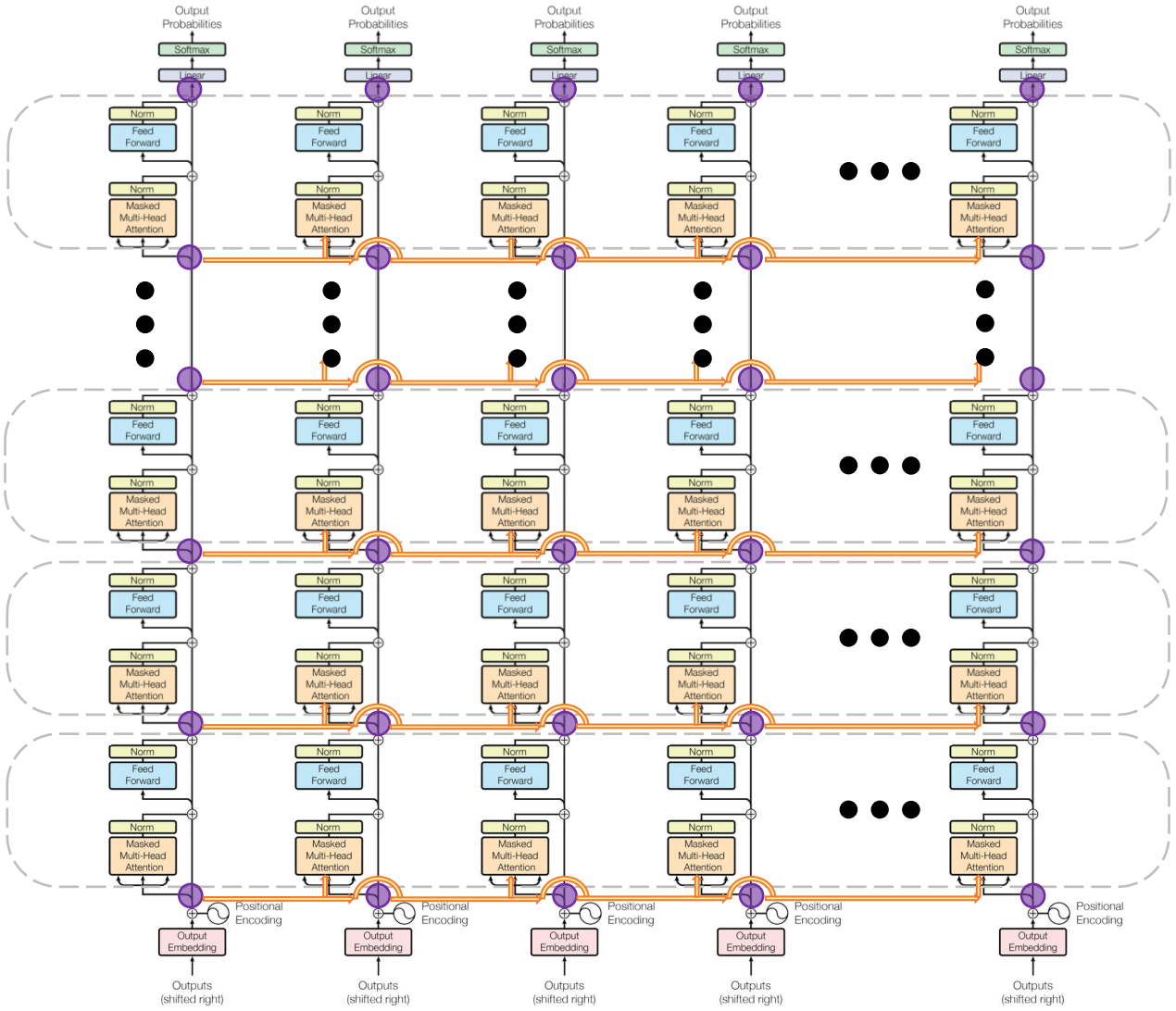
Decoder, Adding Multiple Layers



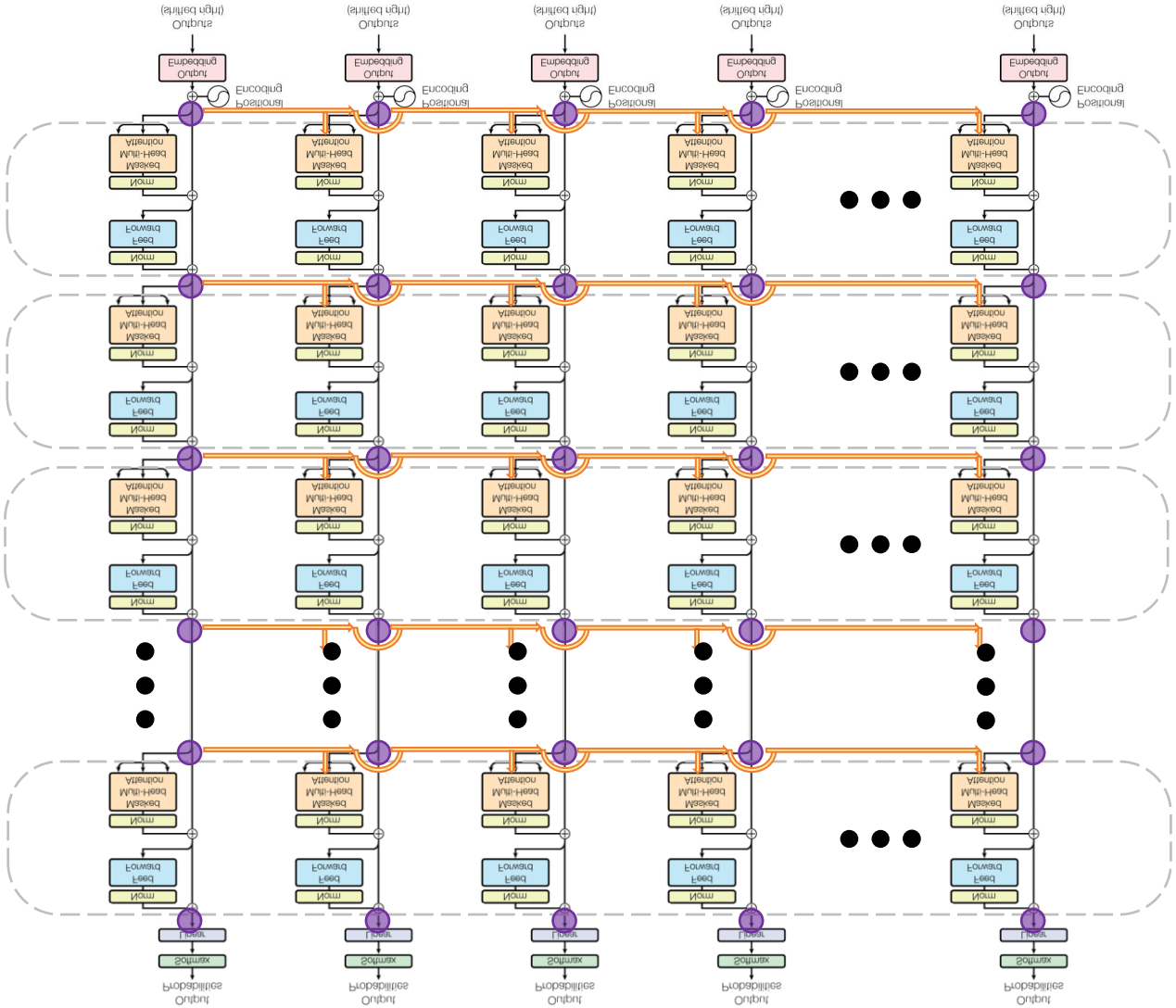
Decoder, Adding Hidden States



Decoder, Adding Attention Paths



Decoder with Attention Paths, Flipped Top to Bottom



Look Before You Leap

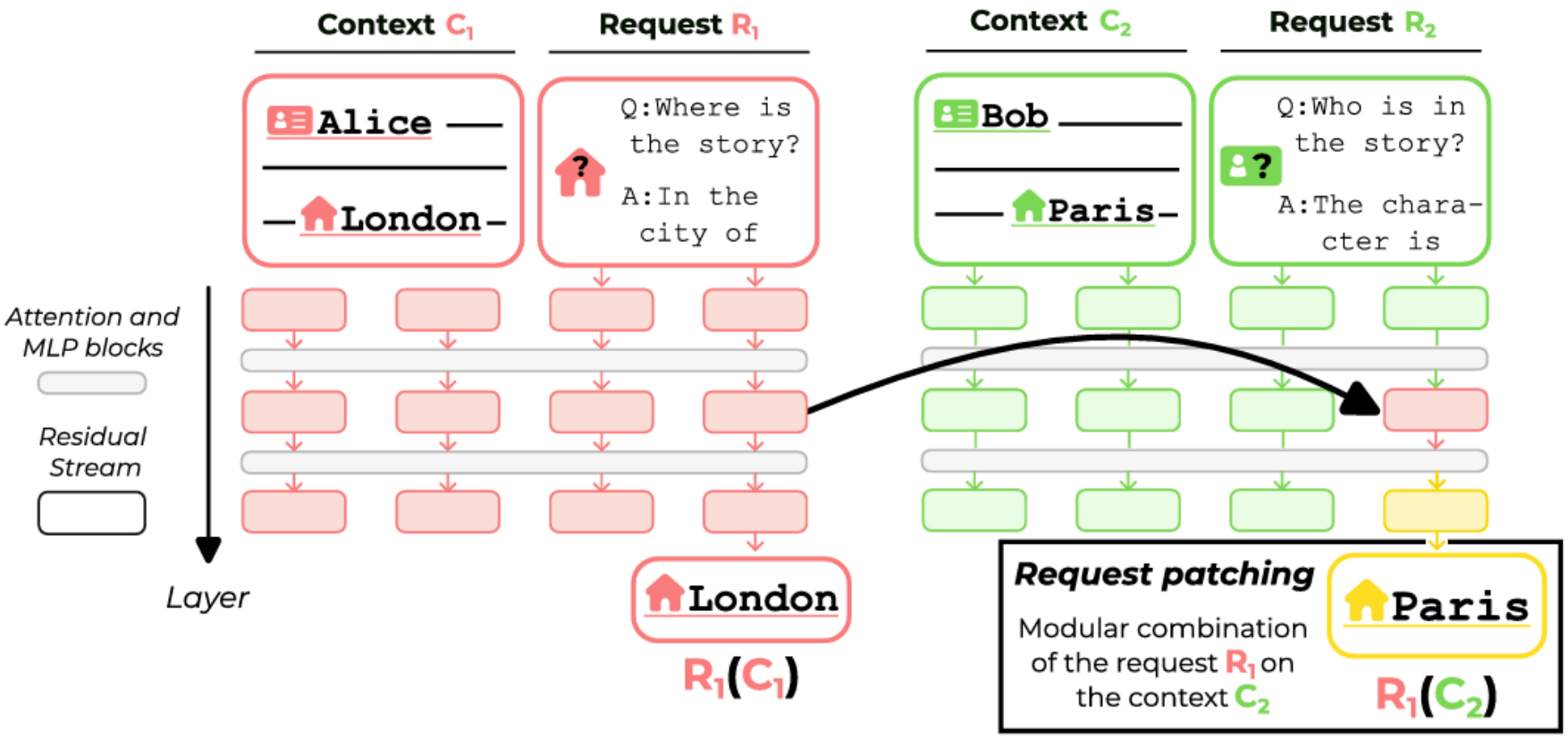


Figure 1: Illustration of our main experimental discovery. Patching the mid-layer residual stream on a retrieval task from ORION causes the language model to output a modular combination of the request from x_1 (asking for the city) and the context from x_2 (a story about Bob in Paris). We call this phenomenon *request-patching*.

Look Before You Leap

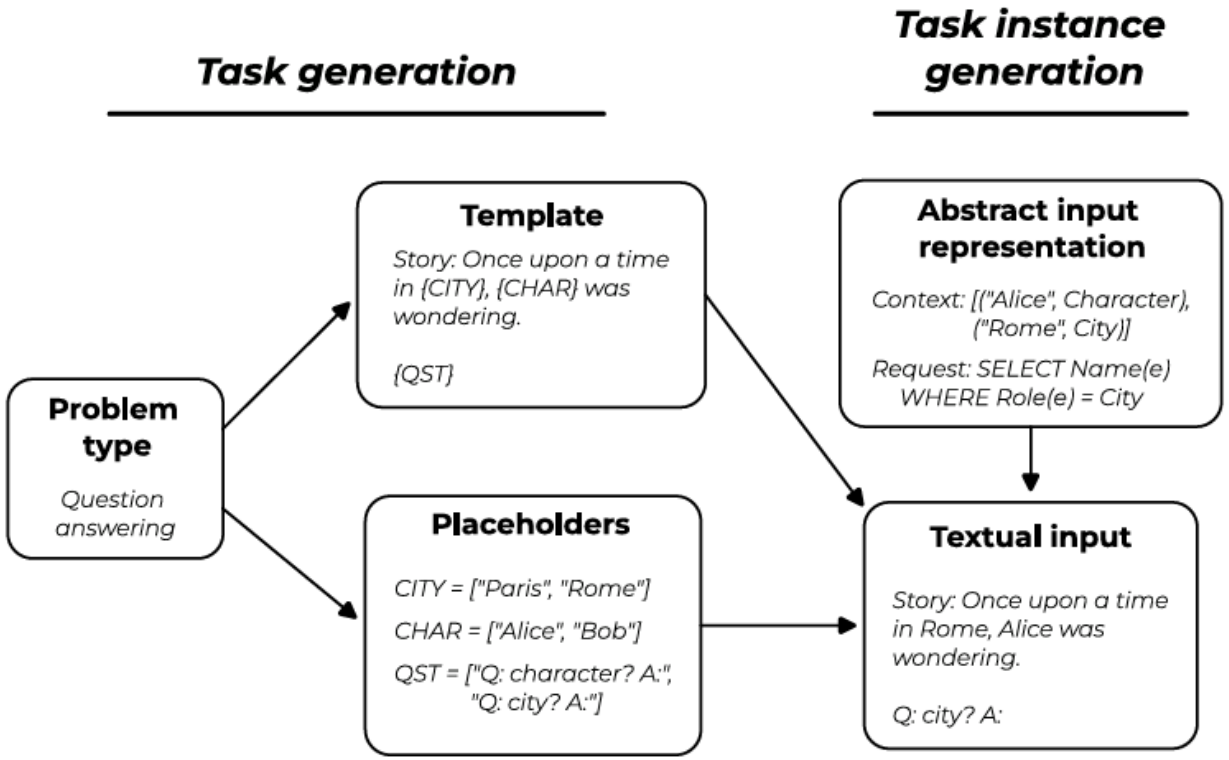


Figure 4: The semi-automatic task generation process used to create ORION. We use ChatGPT to create a template and values for the placeholders given a problem type. To generate an instance from the task, we start by randomly selecting placeholder values to create an abstract input representation. Then, we use a format string to fill the template. When we need more flexibility, we use GPT-4 to incorporate the placeholder values into the template.

Look Before You Leap

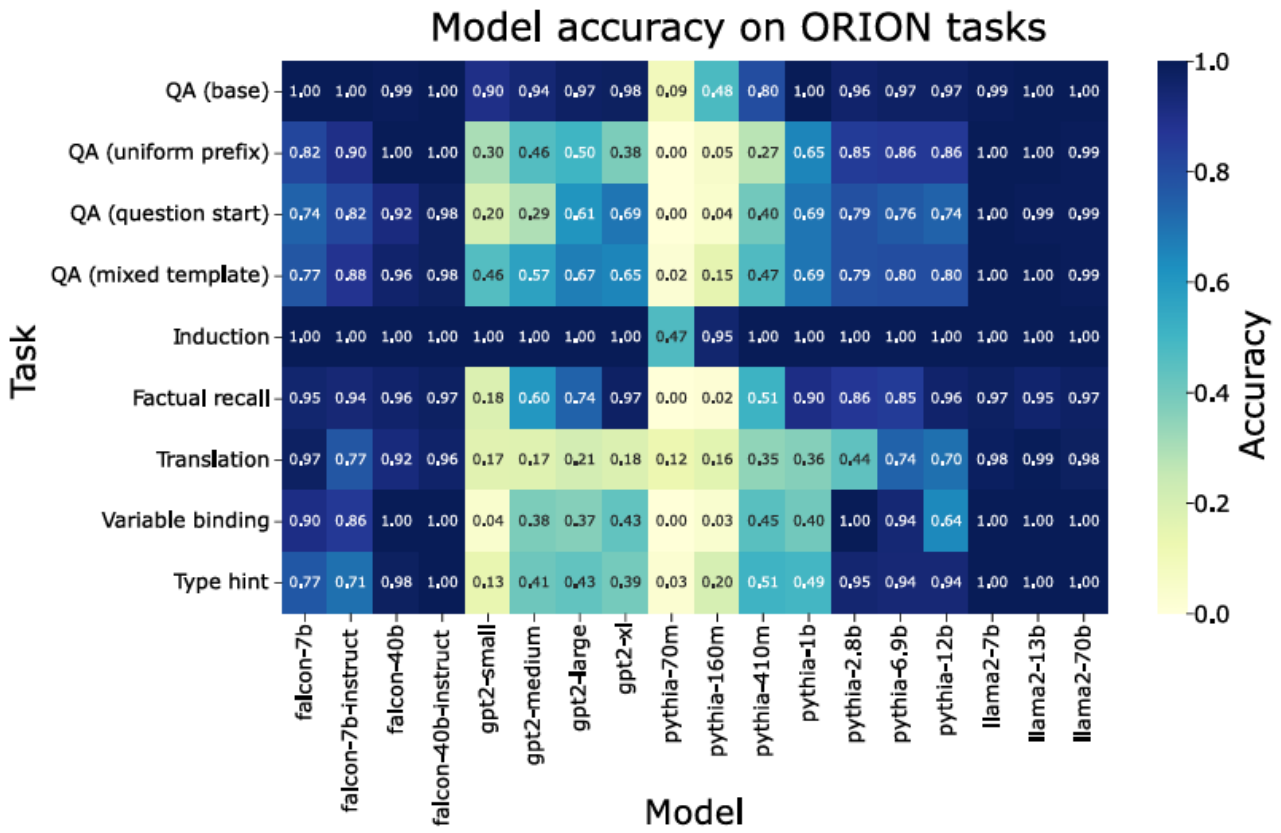


Figure 5: Accuracy of 18 models on the ORION task collection. Models with more than 7 billion parameters are able to robustly solve every task. However, simple tasks such as the base question-answering can be solved by models as small as GPT-2 small (125 million parameters), enabling comparative studies across a wide range of model scales.

Look Before You Leap

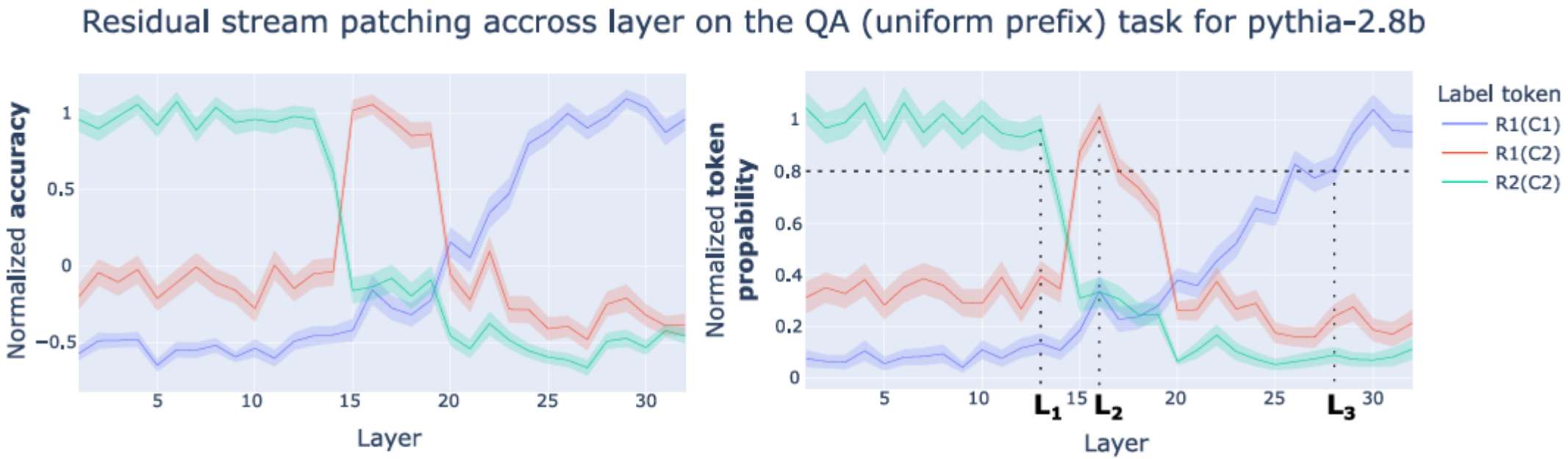


Figure 6: Normalized token probability and accuracy for the label tokens $R_1(C_1)$, $R_1(C_2)$ and $R_2(C_2)$ after patching the residual stream across all layers. Patching early (before $L_1 = 13$) and late (after $L_3 = 27$) leads to the expected results, respectively no change in output and patching the output from x_1. However, intervening on the middle layer ($L_2 = 16$) leads to the model confidently outputting the token $R_1(C_2)$, a modular combination of the request from x_1 and the context from x_2.

Look Before You Leap

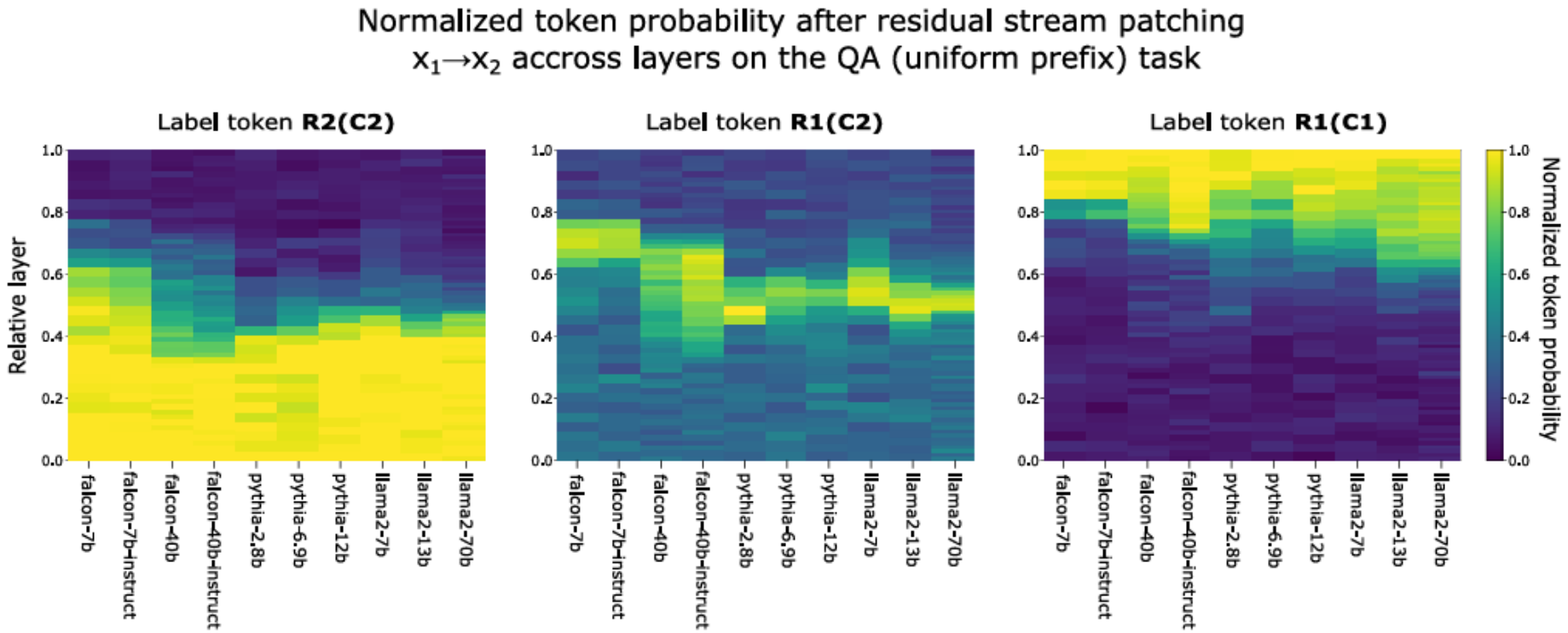


Figure 7: Normalized probability of the label tokens after residual stream patching across all layers on the question-answering task with uniform prefix. To enable comparison across models, we use the relative layer with 0 as the first and 1 as the last layer. Request-patching is general across models: mid-layer residual stream patching causes the model to output $R_1(C_2)$ with more than 80% normalized token probability.

Look Before You Leap

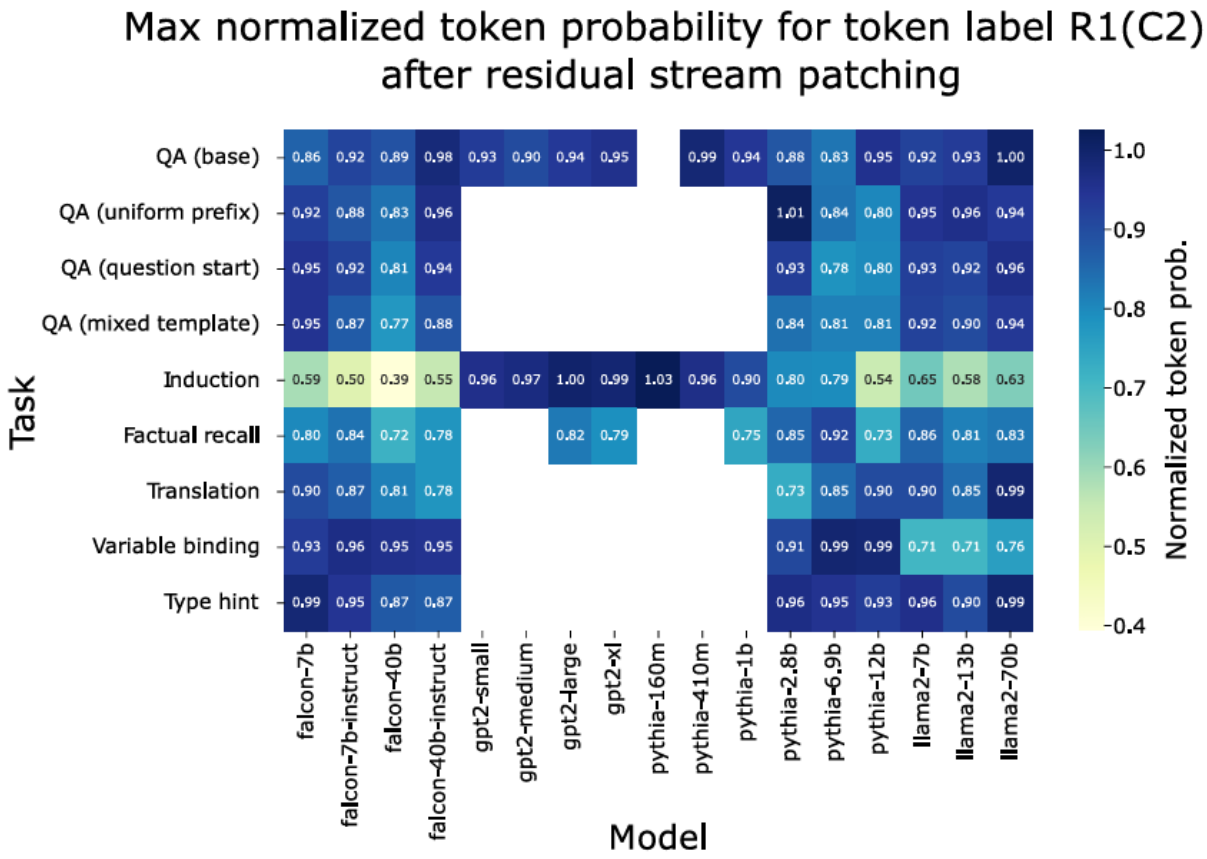


Figure 8: Maximal normalized probability of the $R_1(C_2)$ label token after residual stream patching on all models and tasks from ORION. Request-patching generalizes to the vast majority of tasks and models studied. White regions correspond to settings where the model is unable to robustly solve the task.

Look Before You Leap

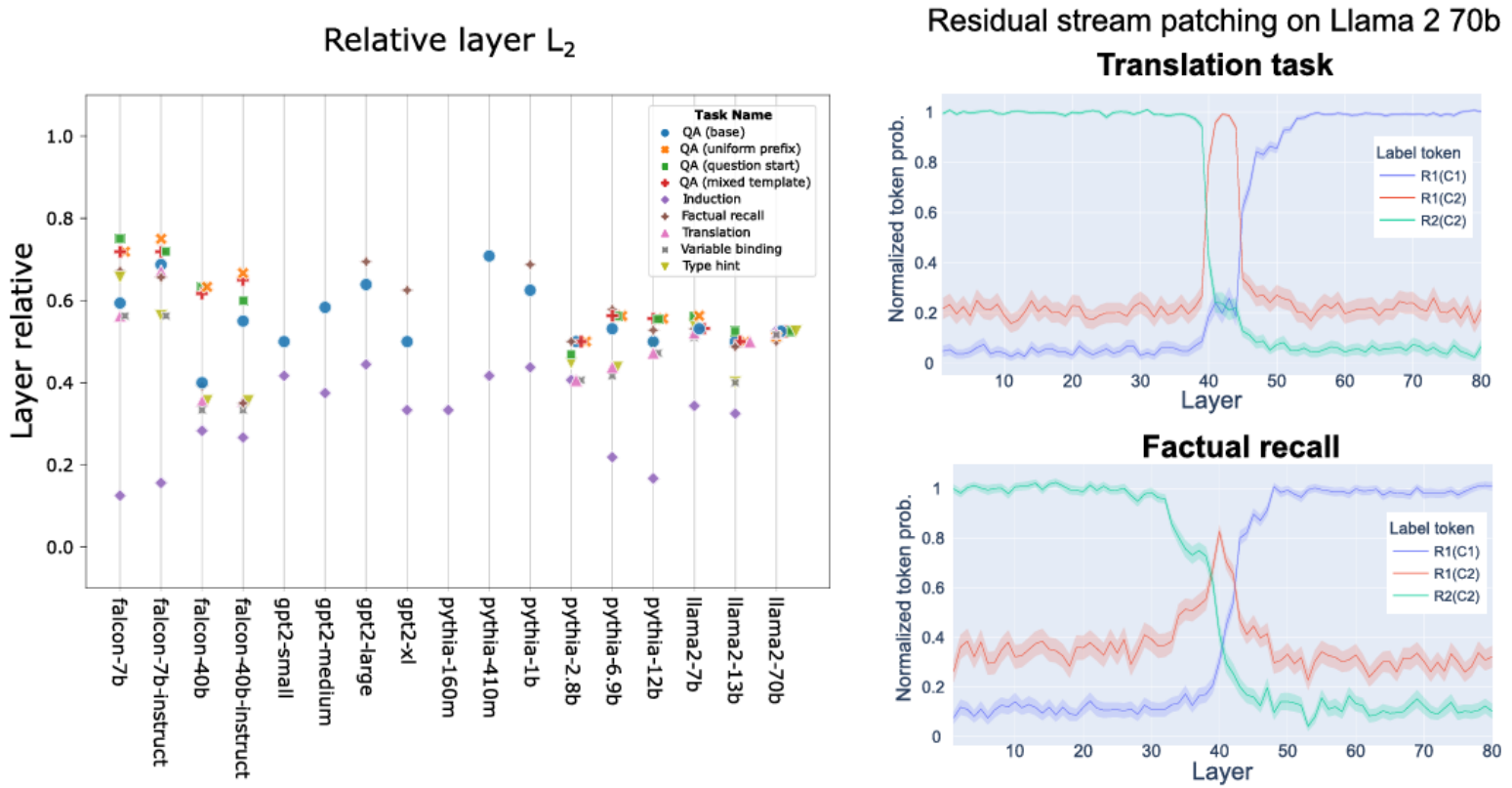


Figure 9: **Left:** Layer of maximal request-patching performance L_2 for different models and tasks. While the L_2 layers for most tasks are concentrated at similar layers, the processing of the request in the induction task seems to happen at earlier layers. **Right:** results of residual stream patching on Llama 2 70b. Request-patching is most performant in a narrow range of layers centered around layer 42 and does not depend on the nature of the task.

Look Before You Leap

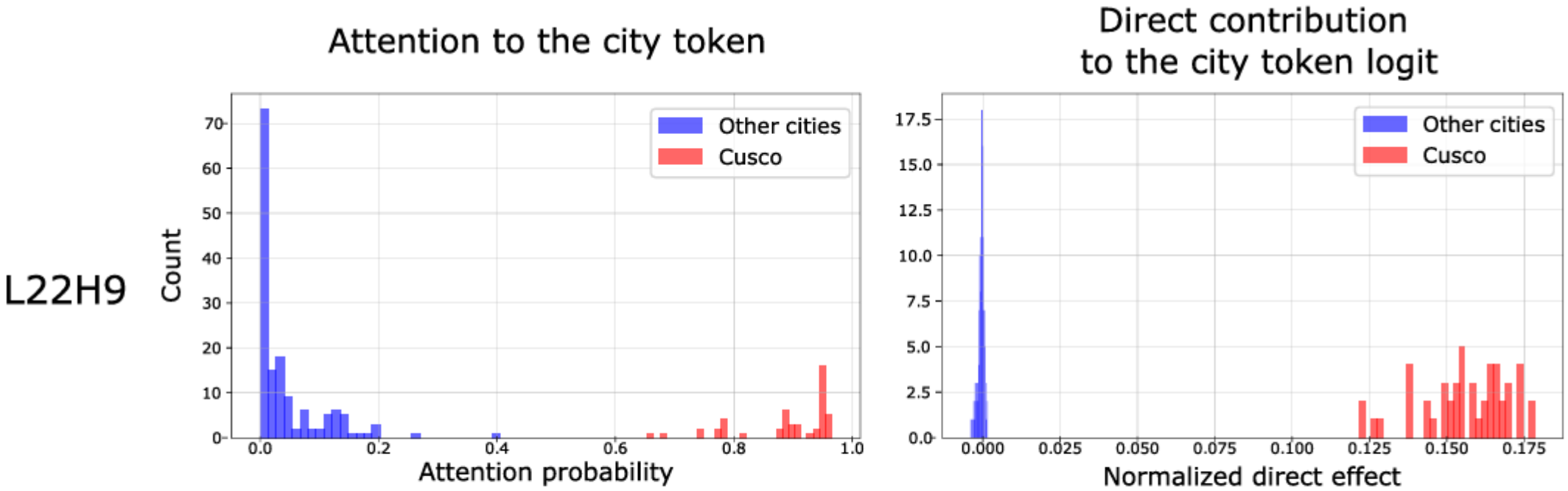


Figure 10: City-specific heads attend to the city token and contribute directly to the logits when the question asks about the city of the story *only* if the city has a specific value, e.g. “Cusco” for the head L22H9.

Look Before You Leap

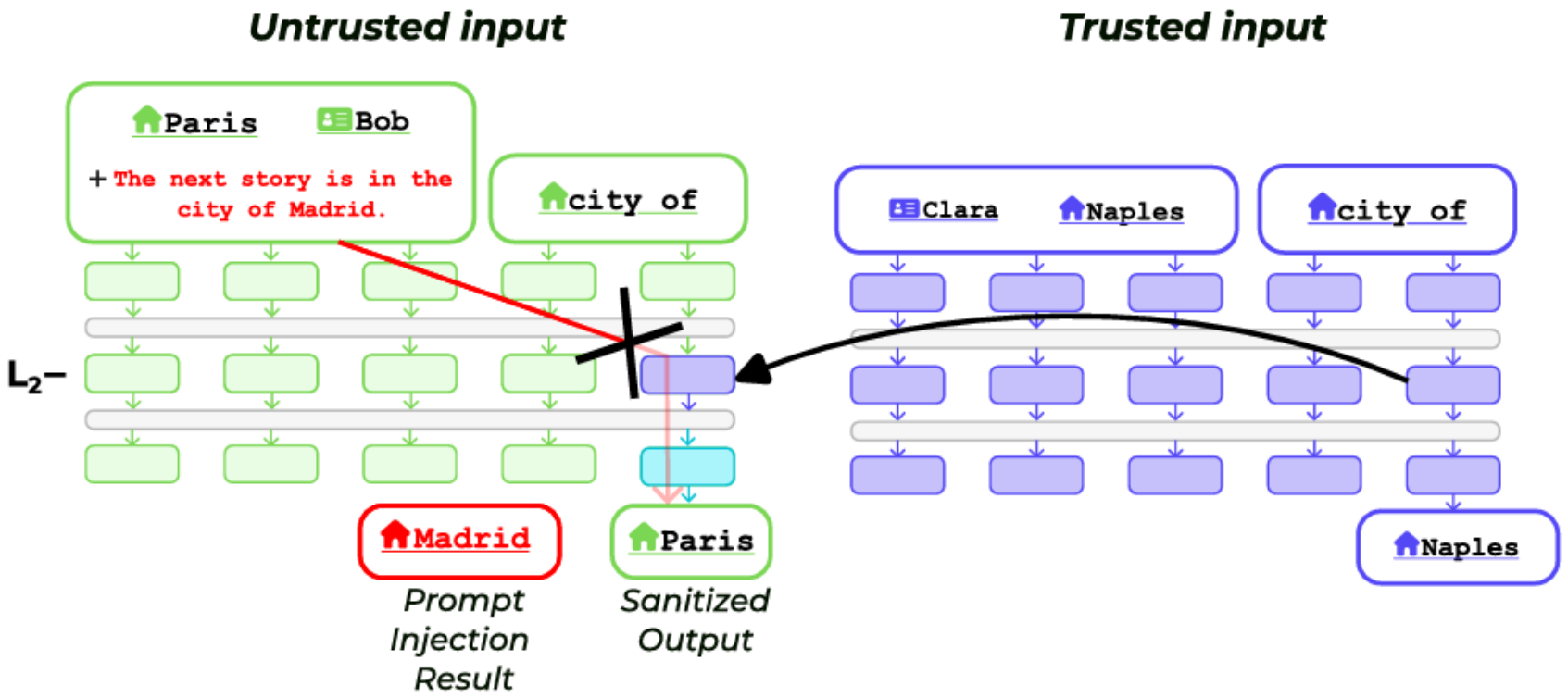


Figure 11: Our scalable internal oversight technique relies on request-patching to remove the influence of the distractor, a string of text crafted to make the model output an arbitrary city (red) instead of answering the question. We patch the residual stream at layer L_2 from an input inspected by a human (blue) to a model processing an untrusted input (green). A single trusted input is used throughout all experiments.

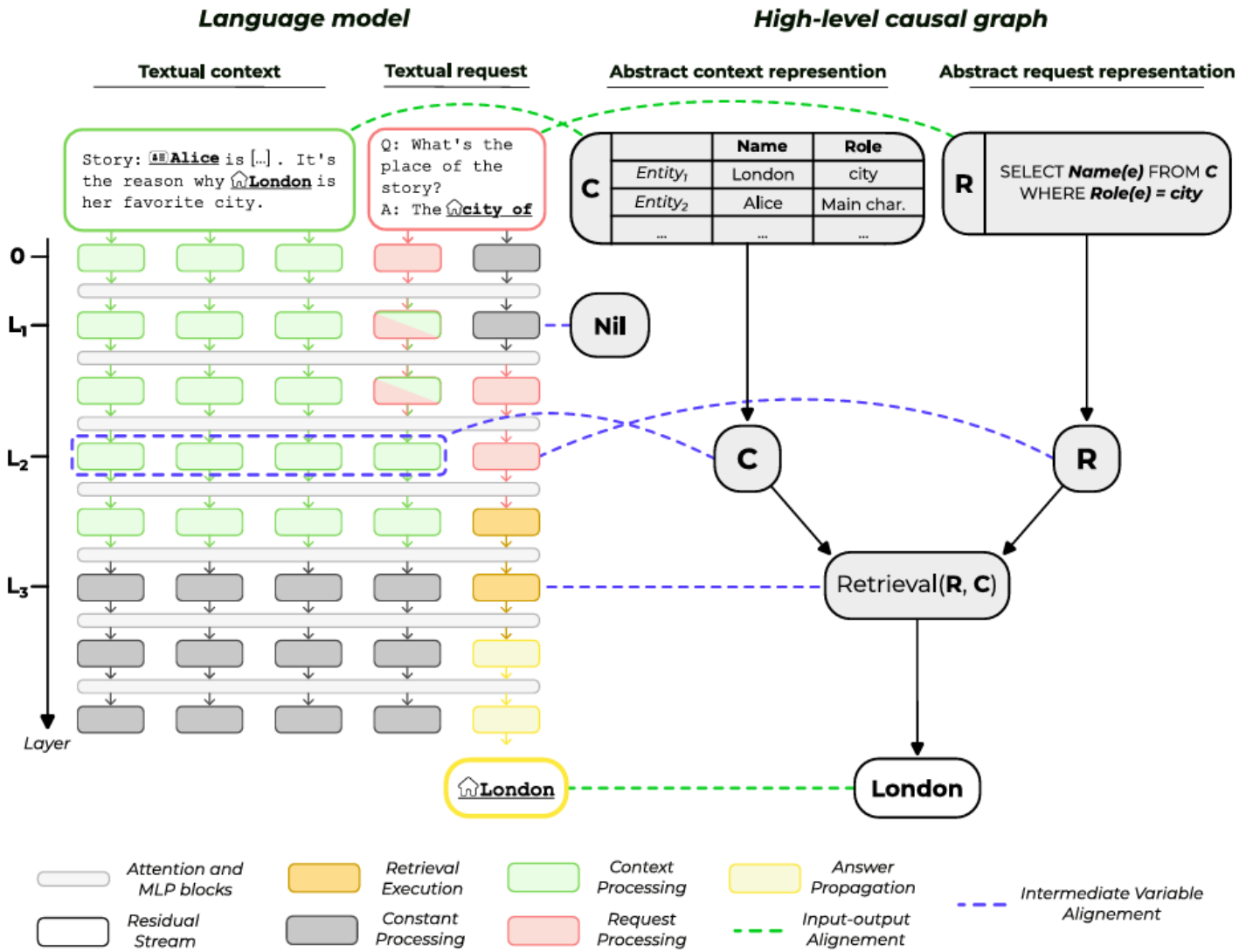


Figure 18: Alignment between a high-level causal graph that uses abstract representations of inputs, and a language model running on the textual representation of the inputs for a retrieval task. The alignment bounds the position where request processing (in red) and context processing (in green) are located in the intermediate layers of the model. The Nil node is isolated in the high-level causal graph. It does not influence the output of the causal graph and thus can be interchanged freely.

References

- Alexandre Variengien's Master Thesis: https://drive.google.com/file/d/1PKQQnQH2pyXfkKJBFqPsATNnG0FhZ2sc/view
- Code: https://github.com/aVariengien/causal-checker
- The NNsight library for similar NN experimentation (and more): https://nnsight.net/