How to think step-by-step: A mechanistic understanding of chain-of-thought reasoning

Paper Presentation for Seminar Foundation Model Frontiers

Subhabrata Dutta *et al.*
IIT Delhi, India

Presented by Zixuan Cao

CIS, LMU Munich

20.06.2025

Table of Contents

Think Step by Step to Boost Reasoning

- LLMs are autoregressive next-word-predictors. With a direct question prompt, the model may be able to generate a correct answer, but also can struggle with complex reasoning required by the question.

Think Step by Step to Boost Reasoning

- LLMs are autoregressive next-word-predictors. With a direct question prompt, the model may be able to generate a correct answer, but also can struggle with complex reasoning required by the question.
- Prompting the model with instructions such as *Let's think step by step* (zero-shot) or demonstrations (few-shot) enables it to generate intermediate computations (thoughts).

Think Step by Step to Boost Reasoning

- LLMs are autoregressive next-word-predictors. With a direct question prompt, the model may be able to generate a correct answer, but also can struggle with complex reasoning required by the question.
- Prompting the model with instructions such as *Let's think step by step* (zero-shot) or demonstrations (few-shot) enables it to generate intermediate computations (thoughts).
- Empirically, the intermediate computations called called **Chain of Thought** (CoT) can improve the reasoning performance.

Example: How many Cakes?

Prompt without CoT

If a baker makes 30 cakes and sells 2/3 of them, how many cakes did he sell?

- Answer: The baker sold 20 cakes.

Example: How many Cakes?

Prompt without CoT

If a baker makes 30 cakes and sells 2/3 of them, how many cakes did he sell?

- Answer: The baker sold 20 cakes.

Prompt with CoT

If a baker makes 30 cakes and sells 2/3 of them, how many cakes did he sell? Let's break this down step-by-step.

- Step 1: Identify the total number of cakes: 30.
- Step 2: Identify the fraction sold: 2/3.
- Step 3: Calculate 2/3 of 30. (2/3) * 30 = 20.
- Final Answer: The baker sold 20 cakes.

Intuition, Beyond Intuition

- Intuitively, we imagine that a model works like humans, so engaging CoT is to break a task into smaller subtasks, which relieves the reasoning pressure at each step.

Intuition, Beyond Intuition

- Intuitively, we imagine that a model works like humans, so engaging CoT is to break a task into smaller subtasks, which relieves the reasoning pressure at each step.
- However, what really happens under the hood, in those transformer blocks and layers?

Intuition, Beyond Intuition

- Intuitively, we imagine that a model works like humans, so engaging CoT is to break a task into smaller subtasks, which relieves the reasoning pressure at each step.
- However, what really happens under the hood, in those transformer blocks and layers?
- Plenty of works had been done before to prove that CoT does equip models with additional reasoning abilities, but they:
  - only looked at the model's behavior, instead of the internal mechanism
  - studied toy models instead of real LLMs
  - assumed CoT was used
  - lacked causality evidence

Table of Contents

The Aim of the Work

This paper studied the reasoning mechanism of a real LLM (`Llama-2 7B`). It specifically includes:

- identifying which attention heads are responsible for a certain subtask

The Aim of the Work

This paper studied the reasoning mechanism of a real LLM (`Llama-2 7B`). It specifically includes:

- identifying which attention heads are responsible for a certain subtask
- checking whether the model grasped relations between entities

The Aim of the Work

This paper studied the reasoning mechanism of a real LLM (`Llama-2 7B`). It specifically includes:

- identifying which attention heads are responsible for a certain subtask
- checking whether the model grasped relations between entities
- exploring which part of knowledge the model attends to

The Aim of the Work

This paper studied the reasoning mechanism of a real LLM (`Llama-2 7B`). It specifically includes:

- identifying which attention heads are responsible for a certain subtask
- checking whether the model grasped relations between entities
- exploring which part of knowledge the model attends to
- tracing the information flow through attention heads

Table of Contents

Quick Recap: Attention Head

- In the self-attention mechanism, for an input X, the attention scores are calculated by:

Quick Recap: Attention Head

- In the self-attention mechanism, for an input X, the attention scores are calculated by:
  - $Q = XW_Q, K = XW_K, V = XW_V$

Quick Recap: Attention Head

- In the self-attention mechanism, for an input X, the attention scores are calculated by:
  - $Q = XW_Q, K = XW_K, V = XW_V$
  - $\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_K}})V$

Quick Recap: Attention Head

- In the self-attention mechanism, for an input X, the attention scores are calculated by:
  - $Q = XW_Q, K = XW_K, V = XW_V$
  - $\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_K}})V$
- A set of matrices Q, K and V captures certain information from the input. However, in order to sufficiently represent it, we train multiple sets of matrices, where each captures a certain aspect of the input information. Each set of computed scores is called an **attention head**.

Quick Recap: Attention Head

- In the self-attention mechanism, for an input X, the attention scores are calculated by:
  - $Q = XW_Q, K = XW_K, V = XW_V$
  - $\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_K}})V$
- A set of matrices Q, K and V captures certain information from the input. However, in order to sufficiently represent it, we train multiple sets of matrices, where each captures a certain aspect of the input information. Each set of computed scores is called an **attention head**.
  - $\text{head}_h = \text{Attention}(XW_Q^{(h)}, XW_K^{(h)}, XW_V^{(h)})$

Quick Recap: Attention Head

- In the self-attention mechanism, for an input X, the attention scores are calculated by:
  - $Q = XW_Q, K = XW_K, V = XW_V$
  - $\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_K}})V$
- A set of matrices Q, K and V captures certain information from the input. However, in order to sufficiently represent it, we train multiple sets of matrices, where each captures a certain aspect of the input information. Each set of computed scores is called an **attention head**.
  - $\text{head}_h = \text{Attention}(XW_Q^{(h)}, XW_K^{(h)}, XW_V^{(h)})$
- After concatenation, the attention heads will be projected back to the token space.
  - $\text{Multihead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h, ..., \text{head}_n)W_O$

Quick Recap: Residual Stream

- Models with deep layers often suffer from vanishing gradients and being unable to preserve information. In order to solve that, we can add the original input to the layer output after each forward computation, called residual connection, or **residual stream**.
  - $\text{Output}(X) = X + \text{Layer}(X)$, where X is the input from the last layer.

Quick Recap: Residual Stream

- Models with deep layers often suffer from vanishing gradients and being unable to preserve information. In order to solve that, we can add the original input to the layer output after each forward computation, called residual connection, or **residual stream**.
  - $\text{Output}(X) = X + \text{Layer}(X)$, where X is the input from the last layer.
- Apart from benefiting training, an advantage that comes in handy is that we can extract the token representations after each layer, especially after each transformer block. This is extremely useful in investigating as well as manipulating the information flow at a given layer.

Fictional and False Ontology

- LLMs are trained with huge datasets that contain rich knowledge.

Fictional and False Ontology

- LLMs are trained with huge datasets that contain rich knowledge.
- This important feature, however, will disturb the experiment, because we want to exclude the possibility that the model answers based on the pretraining knowledge, instead of the context.

Fictional and False Ontology

- LLMs are trained with huge datasets that contain rich knowledge.
- This important feature, however, will disturb the experiment, because we want to exclude the possibility that the model answers based on the pretraining knowledge, instead of the context.
- To achieve this, we invent some entity names that don't make sense, called **Fictional Ontology**, so that the model won't be able to cheat with pretraining knowledge.

Fictional and False Ontology

- LLMs are trained with huge datasets that contain rich knowledge.
- This important feature, however, will disturb the experiment, because we want to exclude the possibility that the model answers based on the pretraining knowledge, instead of the context.
- To achieve this, we invent some entity names that don't make sense, called **Fictional Ontology**, so that the model won't be able to cheat with pretraining knowledge.

A fictional ontology question

Tumpuses are bright. Lempuses are tumpuses. Max is a lempus.
True or False: Max is bright. $\rightarrow$ True

Fictional and False Ontology

- Similarly, **False Ontology** is a statement that is false in reality, but formally valid.

A false ontology question

Mammals can fly. Pigs are mammals. Peppa is a pig.
True or False: Peppa can fly. $\rightarrow$ True

Activation Patching

- During mechanism study, we want to figure out for which function a certain transformer block at a layer is responsible. **Activation Patching** can give us clue.

Activation Patching

- During mechanism study, we want to figure out for which function a certain transformer block at a layer is responsible. **Activation Patching** can give us clue.
- The general idea is as follows:

Activation Patching

- During mechanism study, we want to figure out for which function a certain transformer block at a layer is responsible. **Activation Patching** can give us clue.
- The general idea is as follows:
  - Given a task, e.g., *John and Mary went to the park. John passed the bottle to [?]*, a working model should answer correctly: Mary.

Activation Patching

- During mechanism study, we want to figure out for which function a certain transformer block at a layer is responsible. **Activation Patching** can give us clue.
- The general idea is as follows:
  - Given a task, e.g., *John and Mary went to the park. John passed the bottle to [?]*, a working model should answer correctly: Mary.
  - **Hypothesis: block$_j$ is responsible for moving the name information.**

Activation Patching

- During mechanism study, we want to figure out for which function a certain transformer block at a layer is responsible. **Activation Patching** can give us clue.
- The general idea is as follows:
  - Given a task, e.g., *John and Mary went to the park. John passed the bottle to [?]*, a working model should answer correctly: Mary.
  - **Hypothesis: block$_j$ is responsible for moving the name information.**
  - Copy the token representation x_j^{Mary} at that layer.

Activation Patching

- During mechanism study, we want to figure out for which function a certain transformer block at a layer is responsible. **Activation Patching** can give us clue.
- The general idea is as follows:
  - Given a task, e.g., *John and Mary went to the park. John passed the bottle to [?]*, a working model should answer correctly: Mary.
  - **Hypothesis: block$_j$ is responsible for moving the name information.**
  - Copy the token representation x_j^{Mary} at that layer.
  - Corrupt the input: Change Mary to Anne in the prompt. The model answer becomes Anne.

Activation Patching

- During mechanism study, we want to figure out for which function a certain transformer block at a layer is responsible. **Activation Patching** can give us clue.
- The general idea is as follows:
  - Given a task, e.g., *John and Mary went to the park. John passed the bottle to [?]*, a working model should answer correctly: Mary.
  - **Hypothesis: block$_j$ is responsible for moving the name information.**
  - Copy the token representation x_j^{Mary} at that layer.
  - Corrupt the input: Change Mary to Anne in the prompt. The model answer becomes Anne.
  - Replace x_j^{Anne} with x_j^{Mary}.

Activation Patching

- During mechanism study, we want to figure out for which function a certain transformer block at a layer is responsible. **Activation Patching** can give us clue.
- The general idea is as follows:
  - Given a task, e.g., *John and Mary went to the park. John passed the bottle to [?]*, a working model should answer correctly: Mary.
  - **Hypothesis: block$_j$ is responsible for moving the name information.**
  - Copy the token representation x_j^{Mary} at that layer.
  - Corrupt the input: Change Mary to Anne in the prompt. The model answer becomes Anne.
  - Replace x_j^{Anne} with x_j^{Mary}.
  - **If the model outputs Mary again, then our hypothesis is correct!**

Knockout

- **Knockout** is to eliminate the function of a node by assigning a non-discriminative value to it.

Knockout

- **Knockout** is to eliminate the function of a node by assigning a non-discriminative value to it.
- A naive way of knockout is to set the value of corresponding nodes to zero, but this can be destructive to other computation.

Knockout

- **Knockout** is to eliminate the function of a node by assigning a non-discriminative value to it.
- A naive way of knockout is to set the value of corresponding nodes to zero, but this can be destructive to other computation.
- Using the **False Ontology** mentioned above, this paper constructed:
  - $x_j^{Knock} = \text{Mean}_i(\{x_j^i | s_i \in S \in D_{False}\})$, where D_{False} denotes the set of false ontologies.
    We can imagine this as information from a reversed world, which will confuse the model.

Knockout

- **Knockout** is to eliminate the function of a node by assigning a non-discriminative value to it.
- A naive way of knockout is to set the value of corresponding nodes to zero, but this can be destructive to other computation.
- Using the **False Ontology** mentioned above, this paper constructed:
  - $x_j^{Knock} = \text{Mean}_i(\{x_j^i | s_i \in S \in D_{False}\})$, where D_{False} denotes the set of false ontologies.
    We can imagine this as information from a reversed world, which will confuse the model.
  - To insert this false information:
    $s_i^{Knock} = \underset{x_{logit}^i}{\text{argmax}} \text{LM}_{j,k}^l(x_{logit}^i | S, y_{j,k}^l = x_j^{Knock})$, where j,k and l denote the lth token at the kth head of the jth layer.
    Don't panic. The intuition: Push the node to be eliminated toward the wrong/unreal direction as possible.

Table of Contents

Three Types of Subtasks

- From here, we will dive into the experiments. First of all, in order to make the experiments understandable, we need to construct a human-friendly logic framework in **Fiction Ontology**.

Three Types of Subtasks

- From here, we will dive into the experiments. First of all, in order to make the experiments understandable, we need to construct a human-friendly logic framework in **Fiction Ontology**.
- In other words, subtasks are categorized into three types: **Decision-Making**, **Copying** and **Induction**.

Three Types of Subtasks

- From here, we will dive into the experiments. First of all, in order to make the experiments understandable, we need to construct a human-friendly logic framework in **Fiction Ontology**.
- In other words, subtasks are categorized into three types: **Decision-Making**, **Copying** and **Induction**.
  - **Decision-Making** - The model decides on the path of reasoning to follow.

Three Types of Subtasks

- From here, we will dive into the experiments. First of all, in order to make the experiments understandable, we need to construct a human-friendly logic framework in **Fiction Ontology**.
- In other words, subtasks are categorized into three types: **Decision-Making**, **Copying** and **Induction**.
  - **Decision-Making** - The model decides on the path of reasoning to follow.
  - **Copying** - The LM needs to copy key information given in the input to the output.

Three Types of Subtasks

- From here, we will dive into the experiments. First of all, in order to make the experiments understandable, we need to construct a human-friendly logic framework in **Fiction Ontology**.
- In other words, subtasks are categorized into three types: **Decision-Making**, **Copying** and **Induction**.
  - **Decision-Making** - The model decides on the path of reasoning to follow.
  - **Copying** - The LM needs to copy key information given in the input to the output.
  - **Induction** - The LM uses a set of statements to infer new relations.

Three Types of Subtasks

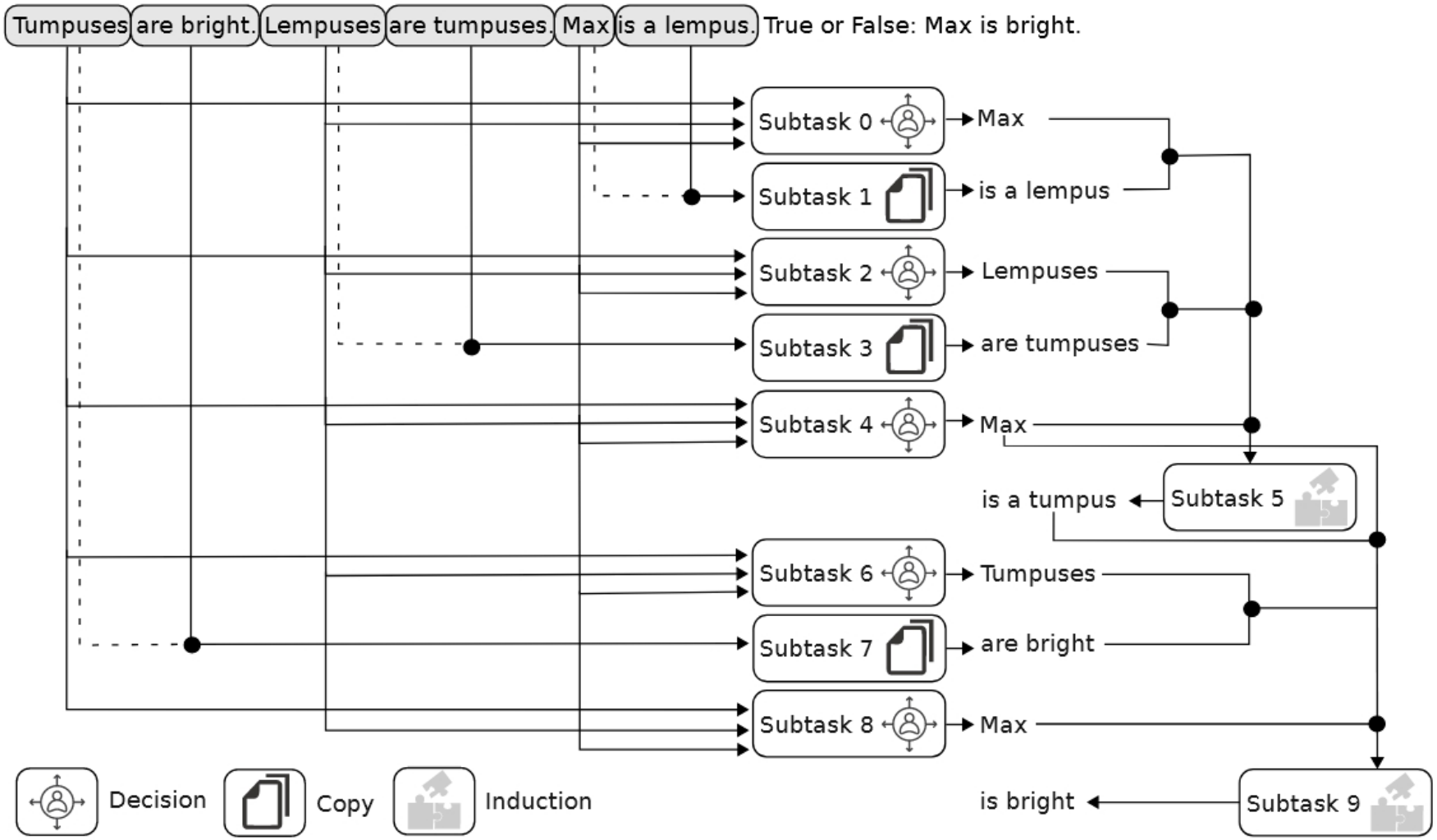

Figure: Task Composition of a Fictional Ontology Question

Identify the Working Heads

- Since we have defined each token as a type of subtask, it is now possible to split heads into three subsets respectively responsible for each type.
- More specifically, we will calculate a responsibility score for each subtask type across heads, using techniques including **Activation Patching**.
- ***Missing math will be completed later.***

Identify the Working Heads

- Ideally, we would be glad to see heads clearly differentiated by subtask types. However, the reality rejects our hypothesis.

Figure: Accuracy (when the rest of heads are knocked out) and Fraction of Heads Involved over all subtasks of Different Sets of Heads

Identify the Working Heads

- Ideally, we would be glad to see heads clearly differentiated by subtask types. However, the reality rejects our hypothesis.
- It is worth noting that:
  - Subtask type—responsibility is shared across many heads.
  - Induction subtasks require the fewest number of heads.
    (Argument: Induction inherently includes decision and copying)
  - Task 4 engages nearly all attention heads, indicating a high degree of distributed processing.

Figure: Accuracy (when the rest of heads are knocked out) and Fraction of Heads Involved over all subtasks of Different Sets of Heads

Table of Contents

Mixed Tokens as Representation of Relations

- Now we want to know whether the model really grasps the relations between entities.

Mixed Tokens as Representation of Relations

- Now we want to know whether the model really grasps the relations between entities.
- First of all, we define the value of a **relation** as negative/-1 (A is not B), neutral/0 (no relation) and positive/1 (A is B).

Mixed Tokens as Representation of Relations

- Now we want to know whether the model really grasps the relations between entities.
- First of all, we define the value of a **relation** as negative/-1 (A is not B), neutral/0 (no relation) and positive/1 (A is B).
- Since we are using fictional ontology, the model is not likely to retrieve a relation from its pretraining knowledge.

Mixed Tokens as Representation of Relations

- Now we want to know whether the model really grasps the relations between entities.
- First of all, we define the value of a **relation** as negative/-1 (A is not B), neutral/0 (no relation) and positive/1 (A is B).
- Since we are using fictional ontology, the model is not likely to retrieve a relation from its pretraining knowledge.
- According to attention mechanism, token representations will be mixed during self-attention.

Mixed Tokens as Representation of Relations

- Now we want to know whether the model really grasps the relations between entities.
- First of all, we define the value of a **relation** as negative/-1 (A is not B), neutral/0 (no relation) and positive/1 (A is B).
- Since we are using fictional ontology, the model is not likely to retrieve a relation from its pretraining knowledge.
- According to attention mechanism, token representations will be mixed during self-attention.
- If there exist three unique patterns in two token representations that are distinguishable given a residual stream, it means the model has successfully learned three above-mentioned relations.

Classification as a Metric

- With the rationales above, we define the model's level of capturing relations as the accuracy of a well-trained classifier with two concatenated mixed tokens.
- The classifier is built with stacked linear layers and a RELU activation layer. We can assume that it will capture any distinct signal given a setup.

Classification as a Metric

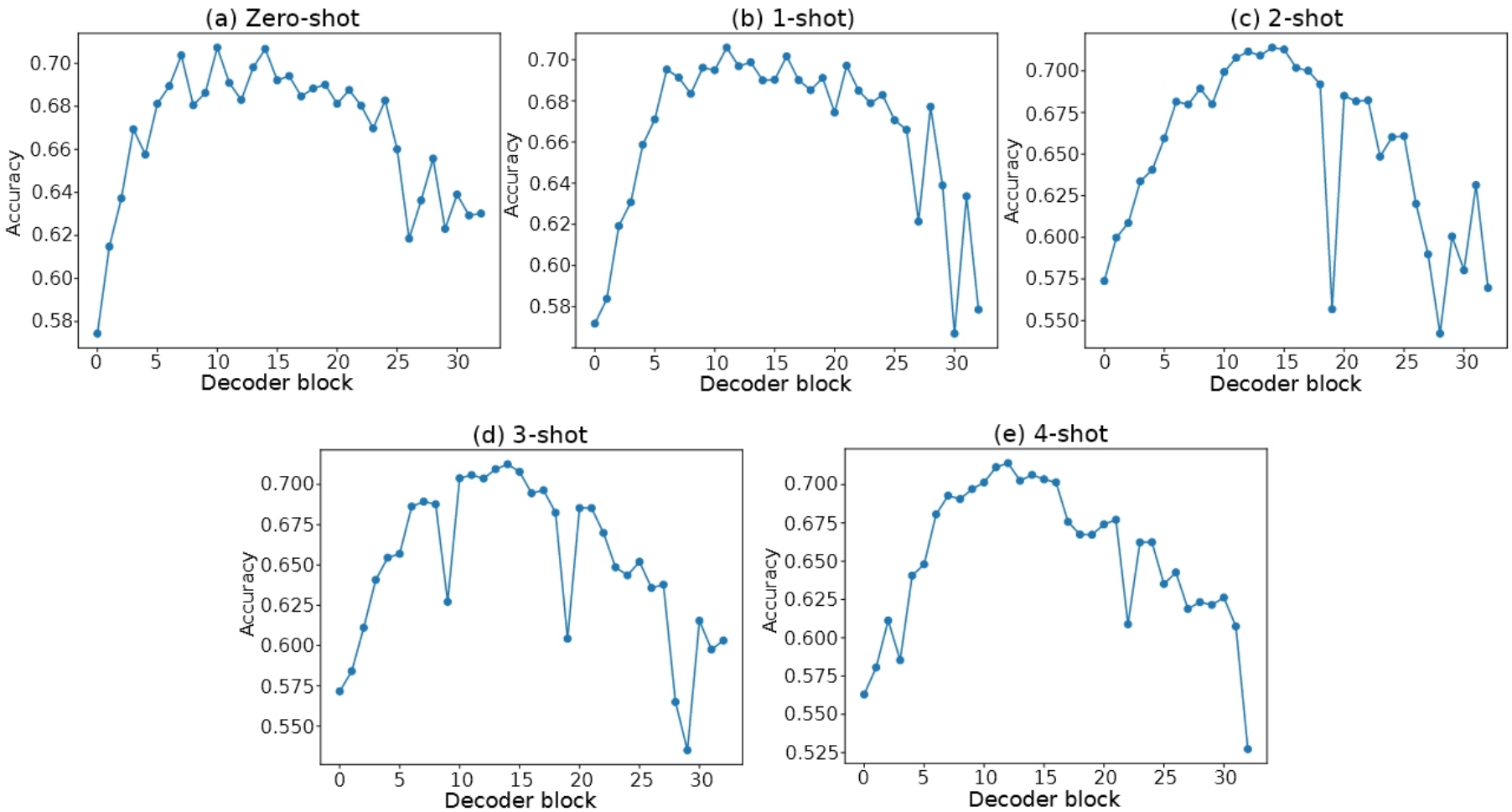

Figure: How does the LLM mix information among tokens according to ontology?

Classification as a Metric

- Notably, the mixing level increases with the depth of decoder blocks until the end, where it drops drastically due to the final decoding.
- Few-shot learning tends to be less stable with respect to token mixing.

Table of Contents

When does the Model rely on Context?

- Now we will explore which layer does the model start attending to the context instead of just relying on patterns from pretraining.

When does the Model rely on Context?

- Now we will explore which layer does the model start attending to the context instead of just relying on patterns from pretraining.
- For a token s_i in the context and an attention head $h_{j,k}$, we can unembed the head to find the predicted token $\hat{s}_i^{j,k}$. If $< s_i, \hat{s}_i^{j,k} >$ is a bigram in the context, then we consider the head as attending to the context.

  It should be noted that applying unembedding projection typically corresponds to Bigram modeling (Elhage et al., 2021). Therefore, when we map any intermediate representation (attention output, residual stream, etc.), we essentially retrieve the token that is most likely to follow.

This is why we use s_i instead of s_{i-1}!

When does the Model rely on Context?

- Now we will explore which layer does the model start attending to the context instead of just relying on patterns from pretraining.
- For a token s_i in the context and an attention head $h_{j,k}$, we can unembed the head to find the predicted token $\hat{s}_i^{j,k}$. If $< s_i, \hat{s}_i^{j,k} >$ is a bigram in the context, then we consider the head as attending to the context.

  It should be noted that applying unembedding projection typically corresponds to Bigram modeling (Elhage et al., 2021). Therefore, when we map any intermediate representation (attention output, residual stream, etc.), we essentially retrieve the token that is most likely to follow.

  This is why we use s_i instead of s_{i-1}!
- Naturally, a **context-abidance score** is defined as $c_{j,k} = \frac{\text{Number of tokens where } <s_i, \hat{s}_i^{j,k}> \text{ is a bigram in } S}{\text{Number of tokens considered}}$.

When does the Model rely on Context?

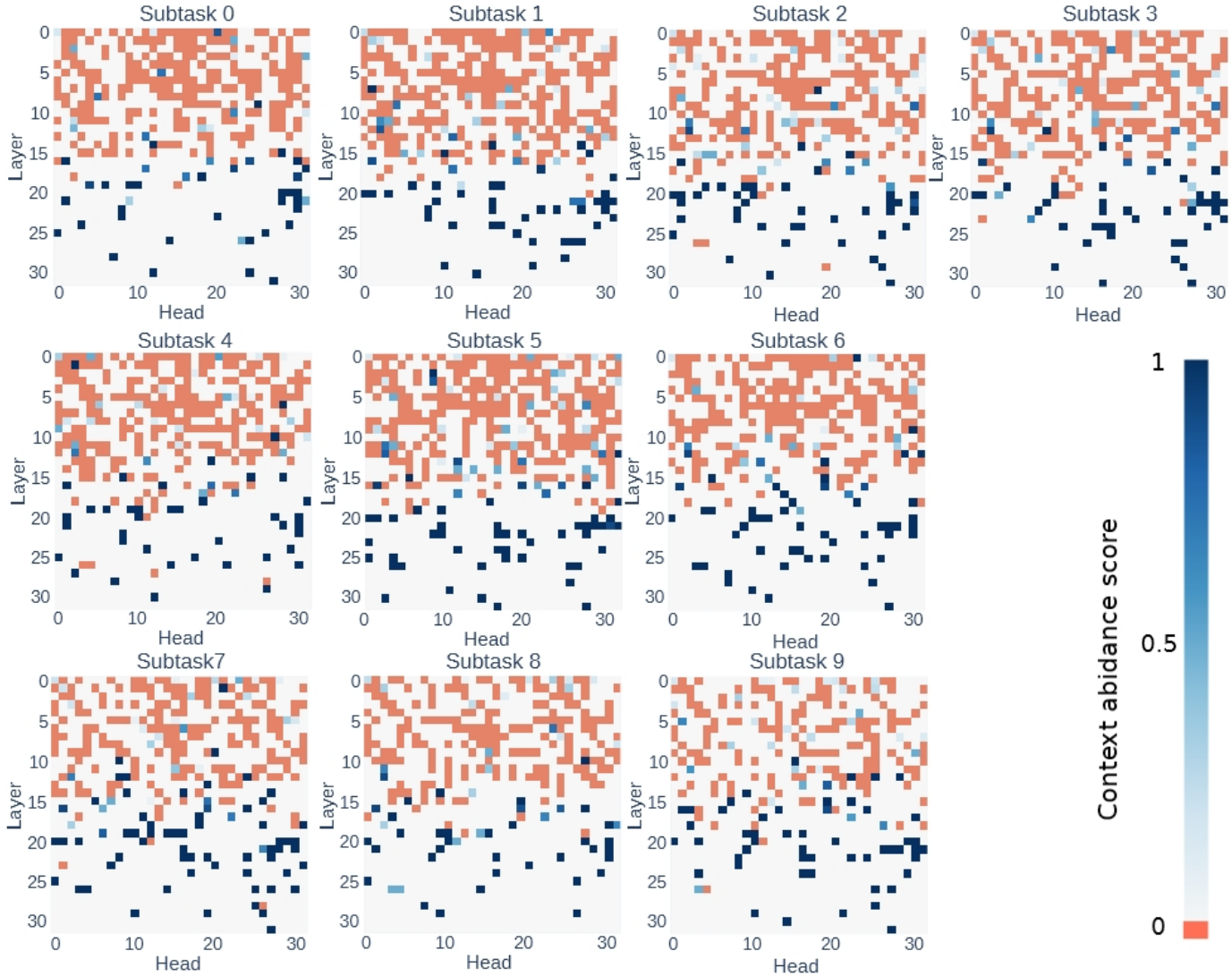

Figure: When does the LLM start following the context?

When does the Model rely on Context?

- Fictional ontology ensures that most bigrams come from the context.
- Therefore, we are convinced that the model starts attending to contextual information only at deeper depths (starting from the 16th layer in this setup).
- No correlation is observed between context abidance and subtasks.

How is the answer written?

- Similarly, we can apply head unembedding to investigate which heads write the answer to a given subtask.

How is the answer written?

- Similarly, we can apply head unembedding to investigate which heads write the answer to a given subtask.
- Here, **the head that writes the answer** is defined as the head whose unembeded token is the answer token. Probabilities will also be recorded.

How is the answer written?

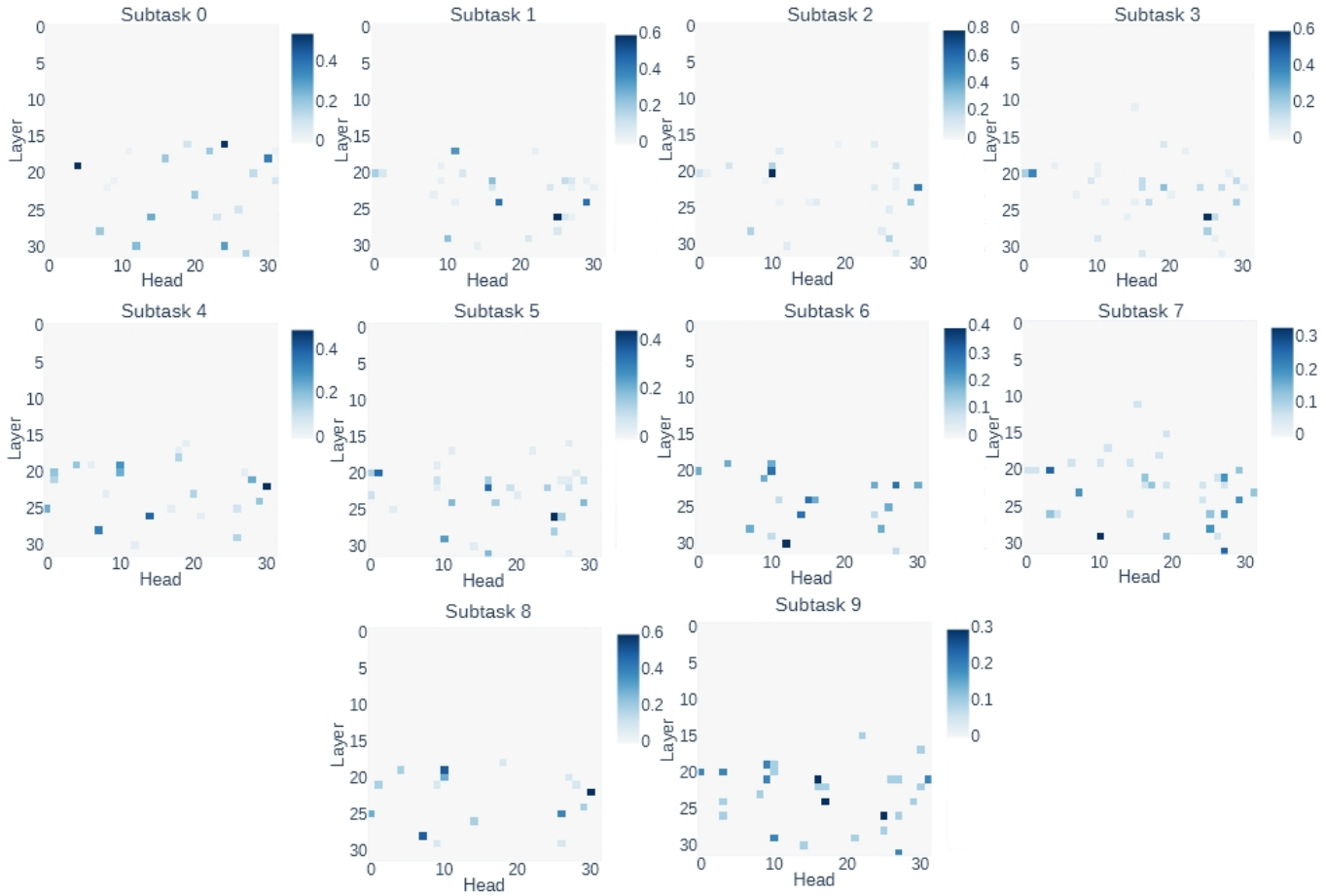

Figure: Which heads are writing the answer?

How is the answer written?

- There are multiple heads writing to the same answer at the same layer, which indicates redundancy in the model.
- Most answer writing starts from the 16th layer.
- Again, subtasks and heads' answer writing are irrelevant.

Where does the answer come from?

- Given the existence of multiple answer writers, we want to know whether they come from the same source.

Where does the answer come from?

- Given the existence of multiple answer writers, we want to know whether they come from the same source.
- To trace the information flow, a recursive strategy is used:
  - Start at an answer-writing head
  - Look at where this head is attending
  - Project those residual streams back into tokens
  - Find out which earlier head wrote that content
  - Repeat recursively
  - Until one of the following cases is met:
    - Reach a head in the first decoder layer (i.e., bottom of the network)
    - Reach the first token in the prompt

Where does the answer come from?

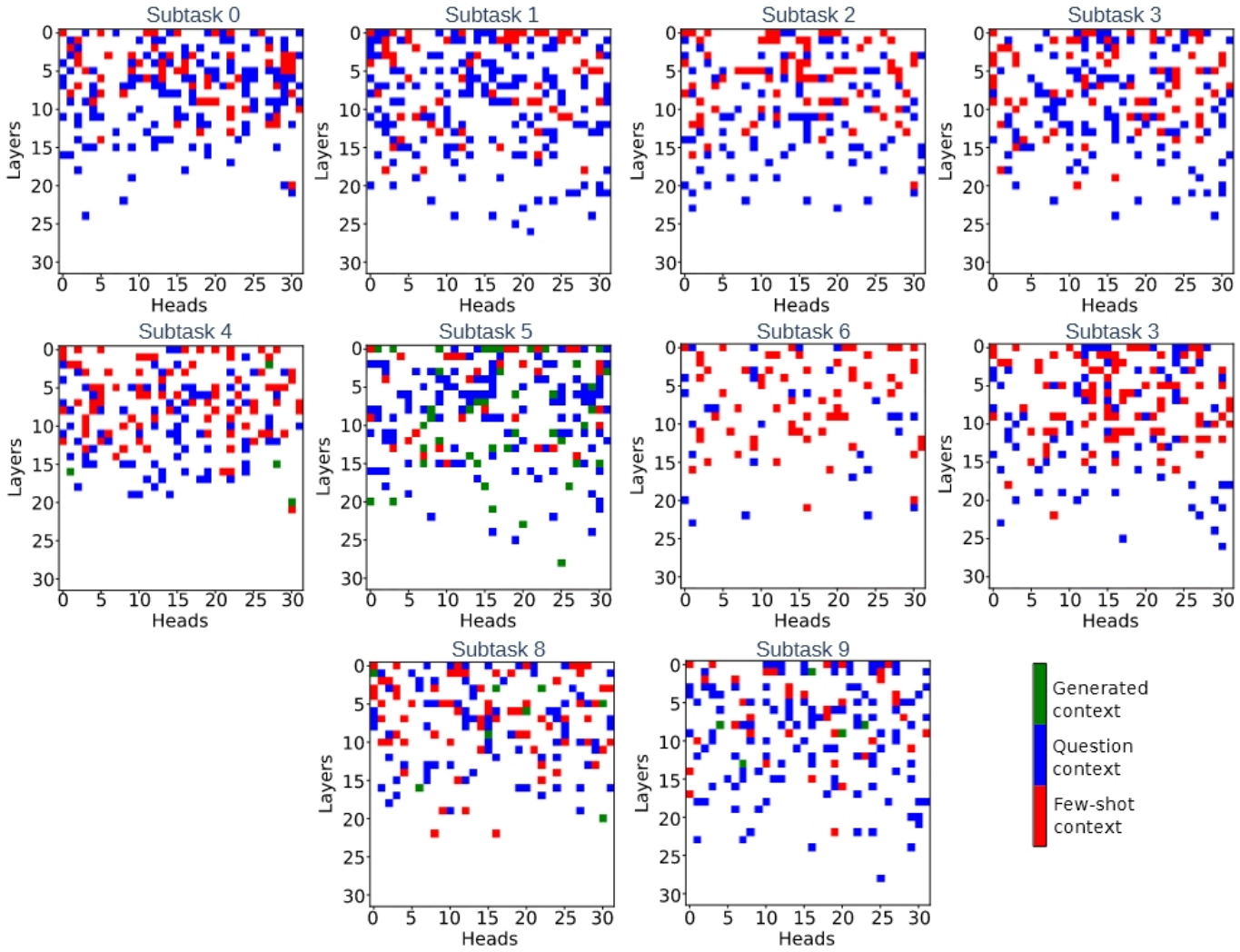

Figure: Where do the answer-writing heads collect their answers from?

Where does the answer come from?

- There are different heads writing answers from different sources
- Answer source is subtask-sensitive.
  - No generated-context-source in subtask 0-3, because no answer token is present in this stage.
  - Similarly, subtask 6 and 7[1] does not include any question context.
  - For subtasks 4, 5, 8, and 9, there are noticeable heads writing from a generated context source. This is a proof that the model is actually referring to CoT.

[1] Note that there is a typo in the graph (Subtask 3 $\rightarrow$ Subtask 7).

Table of Contents

Findings

- Despite varying reasoning needs across CoT stages, the model reuses similar functional components, structured like induction circuits (networks of attention heads)

Findings

- Despite varying reasoning needs across CoT stages, the model reuses similar functional components, structured like induction circuits (networks of attention heads)
- Attention heads facilitate information flow between related tokens, forming distinct representations—starting early and influenced by in-context examples.

Findings

- Despite varying reasoning needs across CoT stages, the model reuses similar functional components, structured like induction circuits (networks of attention heads)
- Attention heads facilitate information flow between related tokens, forming distinct representations—starting early and influenced by in-context examples.
- Multiple neural pathways generate answers in parallel, with different heads contributing to the final token in the residual stream.

Findings

- Despite varying reasoning needs across CoT stages, the model reuses similar functional components, structured like induction circuits (networks of attention heads)
- Attention heads facilitate information flow between related tokens, forming distinct representations—starting early and influenced by in-context examples.
- Multiple neural pathways generate answers in parallel, with different heads contributing to the final token in the residual stream.
- These pathways source answers from the question, few-shot, and generated CoT contexts—confirming that LLMs actively use CoT-generated context when reasoning.

Findings

- Despite varying reasoning needs across CoT stages, the model reuses similar functional components, structured like induction circuits (networks of attention heads)
- Attention heads facilitate information flow between related tokens, forming distinct representations—starting early and influenced by in-context examples.
- Multiple neural pathways generate answers in parallel, with different heads contributing to the final token in the residual stream.
- These pathways source answers from the question, few-shot, and generated CoT contexts—confirming that LLMs actively use CoT-generated context when reasoning.
- A functional rift occurs mid-model (i.e., at the 16th layer in LLaMA-2 7B), marking a shift from bigram-based reasoning to in-context processing; token-mixing and erroneous few-shot answer retrieval happen before this rift, while answer-writing heads emerge after it.

Limitations and Future Research

- This research focused mostly on few-shot learning CoT, while zero-shot learning CoT needs to be studied separately.
- With fictional ontology, this work skiped the function of MLP, which mainly serves as a pretraining knowledge retriever, and only explored transformer blocks. However, the mechanism of MLP needs further research as well.