UNDERSTANDING REASONING IN THINKING LANGUAGE MODELS VIA STEERING VECTORS

Constantin Venhoff*
University of Oxford
United Kingdom
constantin@robots.ox.ac.uk

Iván Arcuschin*
University of Buenos Aires
Argentina
iarcuschin@dc.uba.ar

Philip Torr
University of Oxford
United Kingdom

Arthur Conmy

Neel Nanda

ABSTRACT

Recent advances in large language models (LLMs) have led to the development of *thinking* language models that generate extensive internal reasoning chains before producing responses. While these models achieve improved performance, the underlying mechanisms enabling their reasoning capabilities remain poorly understood. This work studies the particular reasoning processes of thinking LLMs by analyzing DeepSeek-R1 and comparing it with non-thinking models like GPT-4o. Through a systematic experiment on 300 tasks across 10 diverse categories, we identify key behavioral patterns that characterize thinking models, including expressing their own uncertainty, coming up with examples for validating their working hypothesis, and backtracking in reasoning chains. We demonstrate that these behaviors are mediated by linear directions in the model's activation space and can be controlled using steering vectors. By extracting and applying these vectors, we provide a method to modulate specific aspects of the model's reasoning process, such as its tendency to backtrack or express uncertainty. Our findings not only advance the understanding of how thinking models reason but also offer practical tools for steering their reasoning processes in a controlled and interpretable manner. We validate our approach using two DeepSeek-R1-Distill models, showing consistent results across different model architectures.

1 INTRODUCTION

A recent trend in the development of large language models (LLMs) has been *thinking* LLMs, which generate extensive internal reasoning chains before producing responses (Reynolds & McDonell, 2021; Nye et al., 2021; Wei et al., 2022). Examples include OpenAI's o1 (OpenAI, 2024) and DeepSeek's R1 (DeepSeek-AI, 2025). These models have achieved remarkable improvements in performance (Chollet, 2024), yet it remains unclear which internal mechanisms they are learning to efficiently achieve this performance, and how these mechanisms differ from non-thinking models.

To shed light on this, we investigate the unique reasoning mechanisms of thinking LLMs by first analyzing their reasoning chains and comparing them to non-thinking models like GPT-4o. Using DeepSeek-R1-Distill models, we identify distinctive mechanisms that contribute to their efficiency, most notably their ability to express uncertainty, backtrack in their reasoning chain, and self-correct mistakes.

We investigate whether these unique reasoning mechanisms of thinking LLMs can be directly influenced using *steering vectors*, a method that has been shown to allow precise behavioral control in LLMs (Subramani et al., 2022; Turner et al., 2023; Zou et al., 2023; Panickssery et al., 2023; Templeton et al., 2024; Arditi et al., 2024a). By extracting and applying steering vectors, we provide a means to modulate and better understand the internal reasoning dynamics of thinking LLMs.

*Equal contribution

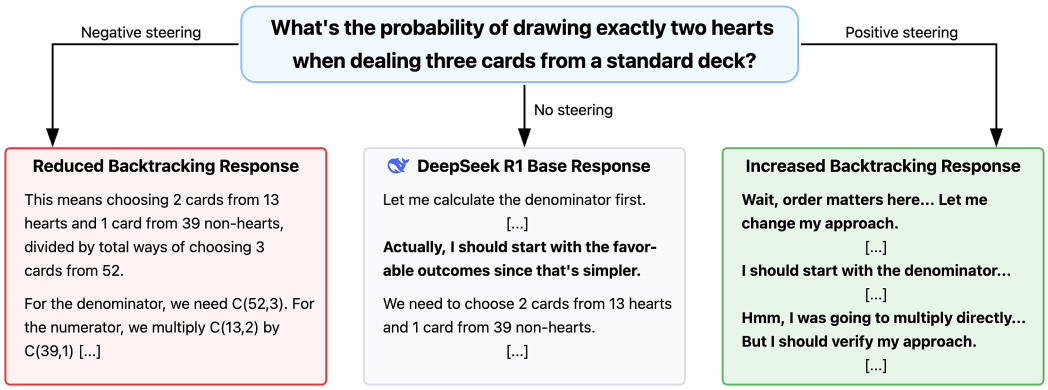

Figure 1: *Steering* on DeepSeek-R1's *backtracking* feature vector changes the model's behavior. Depending on whether we add or subtract this vector to the activations at inference time, the model increases or decreases its tendency to abandon its current approach and explore alternative strategies for the task at hand. Highlighted sections indicate instances of this behavior.

In summary, our work presents the following key contributions:

1. We characterize the reasoning mechanisms in DeepSeek-R1 by comparing it against GPT-4o, finding key differences: DeepSeek-R1 states its own uncertainty and performs self-correction, while GPT-4o follows mostly a linear reasoning process
2. We extract steering vectors for DeepSeek-R1's reasoning mechanisms, allowing us to precisely control its thinking process, e.g. the rate of *backtracking*.
3. We demonstrate the effectiveness of our steering vector by conducting an empirical study on over 300 tasks across 10 diverse categories using two DeepSeek-R1-Distill models.

Our results not only enhance our understanding of how thinking models operate but also provides a practical tool for steering their reasoning processes in a controlled and interpretable manner.

2 Preliminaries

2.1 Attribution Patching

A fundamental challenge in analyzing the behavior of large language models (LLMs) is identifying the specific components and layers responsible for a given behavior. A widely used technique for addressing this challenge is *activation patching* (Meng et al., 2022). Activation patching works by replacing the activations of a specific model component with those from a *counterfactual* example, which differs only in a specific aspect of the behavior being analyzed. If this intervention significantly alters the model's output with respect to the observed behavior, the modified component can be attributed as playing a key role in implementing this behavior. The *patching effect* is quantified as the change in a relevant output metric:

$$\Delta L = L(\mathbf{x}_{\text{clean}} \mid \text{do}(\mathbf{a} = \mathbf{a}_{\text{patch}})) - L(\mathbf{x}_{\text{clean}}),$$

where L is a metric measuring the difference in model outputs (e.g. KL-divergence), $\mathbf{a}$ is the original activation, and $\mathbf{a}_{\text{patch}}$ is the counterfactual activation.

Since activation patching is computationally expensive, a more efficient linear approximation known as *attribution patching* (Nanda, 2023; Syed et al., 2023) is often used, which utilizes the gradients of the model's activations with respect to the metric:

$$\Delta L \approx (\mathbf{a}_{\text{patch}} - \mathbf{a}_{\text{clean}})^T \cdot \frac{\partial}{\partial \mathbf{a}_{\text{clean}}} L(\mathbf{x}_{\text{clean}} \mid \text{do}(\mathbf{a} = \mathbf{a}_{\text{clean}})).$$

2.2 Computing Steering Vectors

The *Difference of Means* method is a widely used technique for extracting steering vectors in LLMs (Turner et al., 2024; Arditi et al., 2024b). This technique is based on constructing contrastive datasets

that differ in a specific concept and computing the difference in their mean activations of a model. Formally, let D_+ and D_- be two datasets where samples in D_+ exhibit a given concept, while samples in D_- do not. Given a model component, we compute the *Difference of Means* vector as:

$$\mathbf{u} = \frac{1}{|D_+|} \sum_{p_i \in D_+} \mathbf{a}(p_i) - \frac{1}{|D_-|} \sum_{p_j \in D_-} \mathbf{a}(p_j)$$

where $a(p_i)$ and $a(p_j)$ represent the activations of the model components over the prompts from the respective datasets. This vector u captures the primary direction in activation space that differentiates the two datasets with respect to the target concept. In cases where explicitly matched counterfactuals are unavailable, a common heuristic is to define D_+ as the set of all samples exhibiting the target behavior, while D_- consists of the full dataset. In this scenario, the *Difference of Means* vector is computed by subtracting the overall mean activation from the mean activation of the behavior-associated examples. This isolates the direction in the activation space most associated with the target behavior while reducing the influence of general model biases.

2.3 DeepSeek Thinking Models

As mentioned in the introduction, *thinking* models are a type of language model designed to generate long chains of internal reasoning before arriving at a final answer. Examples of this type of models include QwQ (Qwen Team, 2024), Gemini 2.0 Flash Thinking (GDM, 2024), o1 (OpenAI, 2024), and DeepSeek R1 (DeepSeek-AI, 2025).

In this work, we focus on characterizing the thinking mechanisms of DeepSeek-R1, a recent *thinking* model that has achieved a similar performance to o1-preview on the ARC-AGI-Pub dataset (Knoop, 2025; Chollet, 2024). DeepSeek-R1 is a language model trained through a multi-stage process that combines large-scale reinforcement learning (RL) with strategic use of supervised fine-tuning (SFT). The model's architecture uses a Mixture-of-Experts (MoE) approach with 37B activated parameters and 671B total parameters.

Additionally, the DeepSeek team has distilled R1's reasoning capabilities into smaller dense models ranging from 1.5B to 70B parameters, based on both Qwen and Llama architectures. These distilled models achieve similar or better performance than frontier production models like GPT-4o and Claude 3.5 Sonnet at several math and coding benchmarks (DeepSeek-AI, 2025). We use the `Qwen-14B` and `Llama-8B` distilled models of DeepSeek R1 for our analysis.

3 Characterizing Thinking Mechanisms in DeepSeek-R1

To characterize the reasoning process of DeepSeek-R1, we manually examined 30 reasoning chains generated by this model and 30 answers produced by a non-thinking model, GPT-4o, using tasks across a diverse set of categories (see Section 4.1). Consistent with prior findings (DeepSeek-AI, 2025), we observe that one of the key differences between the two models is DeepSeek-R1's tendency to explore many different approaches when solving a task, whereas GPT-4o follows a more linear reasoning trajectory. Notably, the thinking model can express uncertainty about its current approach, generate examples or scenarios to test hypotheses and backtrack to intermediate steps when revising its reasoning.

Based on these observations, we categorize DeepSeek-R1's reasoning process into the following behavioral patterns:

- **Initialization**: The model rephrases the task and articulates initial thoughts, typically at the beginning of the reasoning chain.
- **Deduction**: The model derives conclusions based on its current approach and assumptions.
- **Knowledge Augmentation**: The model incorporates external knowledge to refine its reasoning.
- **Example Testing**: The model generates examples or scenarios to validate its working hypothesis.
- **Uncertainty Estimation**: The model explicitly states its confidence or uncertainty regarding its reasoning.
- **Backtracking**: The model abandons its current approach and explores an alternative strategy.

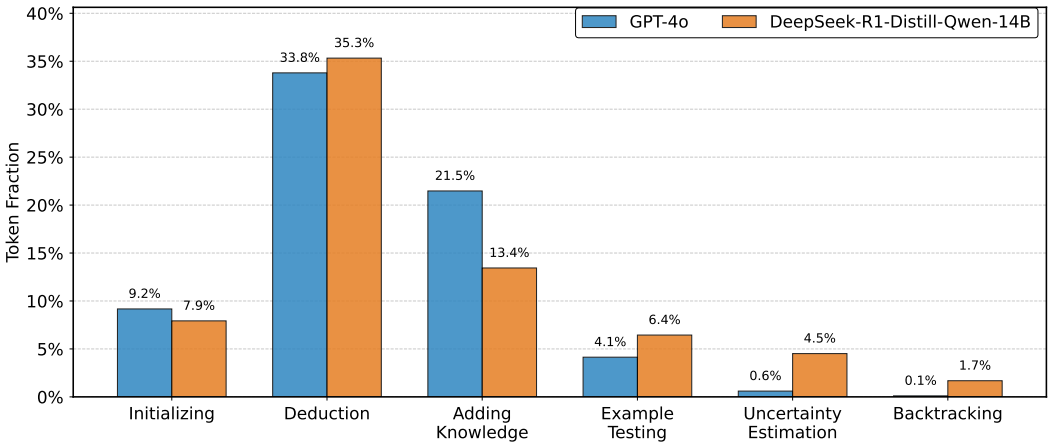

Figure 2: Distribution of behavioral patterns for the Qwen-14B distill-model of DeepSeek-R1 and GPT-4o, on 100 randomly selected tasks from our dataset (cf. Section 4.1). The y-axis shows the percentage of tokens annotated as each behavioral category. We can see that "Initialization" and "Deduction" are behaviors shared by both models. The "Adding Knowledge" behavior is predominant in GPT-4o, while "Example Testing" is predominant in the DeepSeek-R1-Distill. Finally, the "Uncertainty Estimation" and "Backtracking" are behaviors almost exclusive to the thinking model.

An annotated example of these behaviors is provided in Section B. To systematically annotate reasoning chains for our experiments, we employ GPT-4o (see Section A for details). To quantify the prevalence of these reasoning behaviors, we generate reasoning chains using DeepSeek-R1-Distill-Qwen-14B and answers using GPT-4o across 300 diverse tasks (see Section 4.1). We then automatically annotate each reasoning chain and compute the fraction of tokens annotated with each behavioral category. Results are presented in Figure 2.

First, we note that both models have a similar fraction of tokens dedicated to articulating initial thoughts about the task and performing deductive reasoning, indicating that these are the main shared mechanisms between them. Furthermore, we can see that GPT-4o almost never expresses its uncertainty or performs backtracking. It also generates examples significantly less often than the DeepSeek-R1 model. Instead, GPT-4o recalls its stored factual knowledge more often. This suggests that uncertainty estimation and backtracking are the main distinctive features of DeepSeek-R1's reasoning process.

Given that the models are not distinguishable in their initialization and deduction behavior, steering on these seems conceptually ill-defined; initialization occurs once at the start and deduction is fundamental to any LLM, therefore we omit them from the empirical study on steering vectors.

4 Extracting and Evaluating Steering Vectors

In this section, we demonstrate that the reasoning behaviors of the DeepSeek-R1 model characterized in Section 3 are mediated by linear directions, the *steering vectors*. We assess the causal effect of these vectors by comparing the model's original reasoning chains to those generated under positive and negative steering (adding and subtracting the steering vectors). Our findings indicate that DeepSeek-R1 learns distinct mechanisms to achieve its reasoning process. Additionally, our steering vectors provide an efficient way to influence DeepSeek-R1's reasoning behavior, for example increasing its tendency to backtrack or modulate its inherent uncertainty in its own reasoning.

4.1 Experimental Setup

For our experiments, we generate a dataset of 300 tasks across 10 categories using Claude 3.5 Sonnet (see Table 1). We conduct our experiments on two DeepSeek-R1-Distill models: `Qwen-14B` and `Llama-8B`. When generating a reasoning chain we use greedy decoding and 500 max tokens per response.

Category	Description
Mathematical Logic	Problems requiring formal logical operations, mathematical proofs, and numerical reasoning
Spatial Reasoning	Tasks involving visualization, geometric manipulation, and understanding spatial relationships
Verbal Logic	Problems focused on language-based reasoning, syllogisms, and verbal analogies
Pattern Recognition	Questions requiring identification and continuation of sequences or abstract patterns
Lateral Thinking	Problems that require creative, non-linear approaches to reach unconventional solutions
Causal Reasoning	Tasks involving understanding cause-and-effect relationships and making causal inferences
Probabilistic Thinking	Problems requiring reasoning about uncertainty, probability, and statistical concepts
Systems Thinking	Questions about complex systems, interconnected components, and emergent behaviors
Creative Problem Solving	Open-ended problems requiring novel approaches and innovative solutions
Scientific Reasoning	Tasks involving hypothesis formation, experimental design, and evidence evaluation

Table 1: Task categories used to analyze reasoning behaviors.

4.2 Locating Causally Relevant Activations

To extract robust steering vectors, we first identify the activations where these vectors are linearly represented within the model. We focus on the *residual stream activations*, i.e., the outputs of each transformer layer. Given a reasoning chain generated by a DeepSeek-R1 model, we identify both the *token positions* and *layers* where the steering vector is active. This process consists of two key steps:

1. **Identifying relevant token positions**: Determine which tokens in the reasoning chain correspond to a specific behavioral category.
2. **Determining causally relevant layers**: Use *attribution patching* (Section 2) to evaluate which layers contribute causally to the behavior in question.

To obtain token positions associated with each behavioral category, we generate 300 reasoning chains with the tasks introduced in Section 4.1, using both DeepSeek-R1 models and then annotate them automatically with GPT-4o. Since DeepSeek-R1 is an autoregressive model, we consider for each category both the token position **preceding** the start of a token-sequence annotated with the current category and the annotated sequence itself as the causally relevant token positions. This ensures that we capture both the decision point where the model transitions into the behavior and the behavior's execution phase.

To identify the causally relevant layers for each behavioral category, we first extract a steering vector candidate from every layer using the *Difference of Means* method (Section 2.2):

$$\mathbf{u}_\ell^c = \frac{1}{|D_+|} \sum_{p_i \in D_+} \bar{\mathbf{a}}_\ell^c(p_i) - \frac{1}{|D_-|} \sum_{p_j \in D_-} \mathbf{a}_\ell^c(p_j), \quad \text{with} \quad \bar{\mathbf{a}}_\ell^c(p_i) = \frac{1}{|\text{seq}_c(p_i)|} \sum_{t \in \text{seq}_c(p_i)} \mathbf{a}_\ell(t).$$

where:

- $\mathbf{a}_\ell(t)$ represents the residual stream activation at layer ℓ for token position t.
- $\text{seq}_c(p)$ is the set of all token sequences within prompt p that are annotated with category c, including the preceding token position.
- $\bar{\mathbf{a}}_\ell^c(p_i)$ denotes the *mean activation* across all token positions within the annotated sequences of category c at layer ℓ.

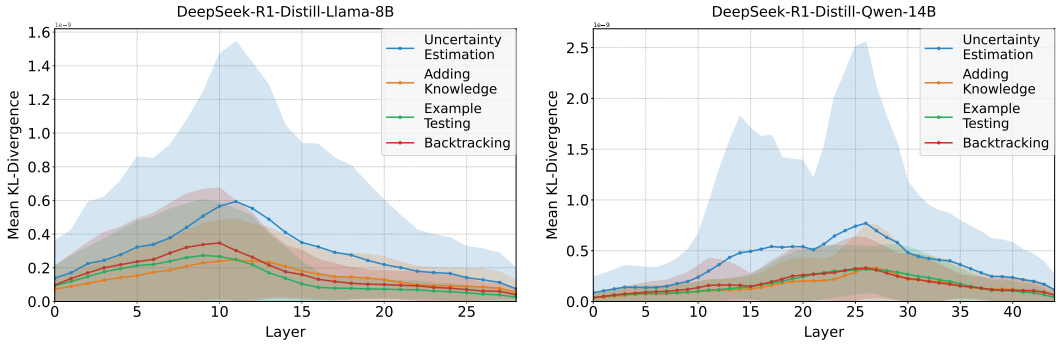

Figure 3: Causal impact of candidate steering vectors across model layers. The y-axis represents the absolute mean KL-divergence for the next-token logit distribution when removing the steering vector at each layer. The left plot shows results for DeepSeek-R1-Distill-Llama-8B, while the right plot corresponds to DeepSeek-R1-Distill-Qwen-14B. The steering vectors for all reasoning mechanisms have similar peaks in the middle layers of the respective models.

- D_+ consists of prompts containing at least one sequence labeled with category c, while D_- represents the full dataset.

The resulting vector $\mathbf{u}_\ell^c$ serves as a candidate steering vector for each layer.

4.3 Selecting Final Steering Vectors

To determine the final steering vectors, we apply *attribution patching* (Section 2.1) to quantify the causal relevance of each vector in its respective layer. Specifically, we consider the following patching experiment: Given a candidate steering vector $\mathbf{u}_\ell^c$ for a specific behavioral category, we add it to the residual stream activation preceding a token-sequence annotated with one of the relevant behaviors. Therefore, we define the patched activation as:

$$\mathbf{a}_\ell^{\text{patched}} = \mathbf{a}_\ell + \mathbf{u}_\ell^c.$$

If this intervention leads to a significant change in the KL divergence of the next-token prediction, then the steering vector in layer ℓ is *causally relevant* for the given behavior. We approximate the patching effect for this experiment with:

$$\Delta L \approx (\mathbf{u}_\ell^c)^T \cdot \frac{\partial}{\partial \mathbf{a}_\ell} L(\mathbf{x}_{\text{clean}} \mid \text{do}(\mathbf{a}_\ell = \mathbf{a}_{\text{clean}})),$$

where $\mathbf{u}_\ell^c = (\mathbf{a}_\ell^{\text{patched}} - \mathbf{a}_\ell)$. We average the absolute patching effect for each category over all category-sequences in all 300 reasoning chains. The results are shown in Figure 3. Based on these results, we can select the causally most relevant steering vectors from the layers where the patching scores are highest.

4.4 Evaluation of Steering Vectors

To evaluate the effectiveness of our extracted steering vectors, we apply them at the selected layers (see Table 2) and observe their influence on the model's reasoning process. Steering is implemented by *adding or subtracting* the extracted steering vectors $\mathbf{u}_\ell^c$ to the residual stream activations at inference time. By applying this intervention, we can increase or decrease behaviors such as backtracking, uncertainty estimation, and example testing, providing a direct mechanism for manipulating the model's reasoning process.

4.5 Results of Steering

We apply each steering vector to 30 unseen reasoning tasks and analyze how the model's reasoning behavior changes. The results, presented in Figure 4, demonstrate that our extracted vectors effectively

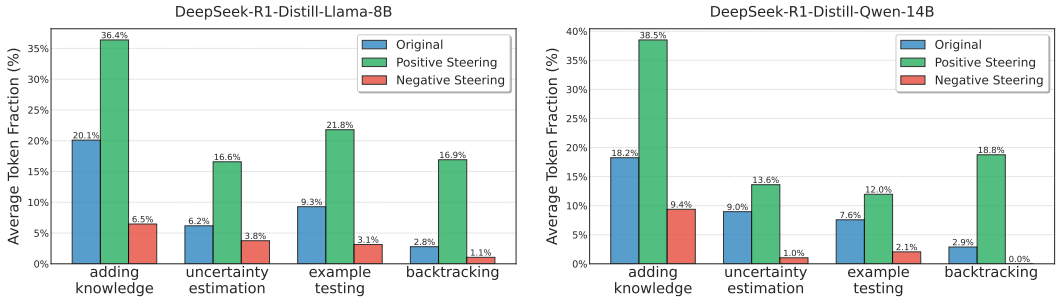

Figure 4: Effect of applying the steering vector for each reasoning behavior. The y-axis shows the change in the fraction of tokens exhibiting each behavior when applying positive or negative steering. Results for DeepSeek-R1-Distill-Llama-8B are shown on the left, and results for DeepSeek-R1-Distill-Qwen-14B are on the right. Positive steering increases behaviors such as backtracking and uncertainty estimation, while negative steering suppresses or significantly reduces them, confirming the causal influence of our extracted vectors.

Behavioral Category	DeepSeek-R1-Distill-Llama-8B	DeepSeek-R1-Distill-Qwen-14B
Uncertainty Estimation	9, 10, 11	14, 24, 25, 26
Example Testing	8, 9, 10, 11	22, 23, 28, 29
Backtracking	9, 10, 11	11, 24, 25, 26
Adding Knowledge	11, 12, 13	24, 25, 26, 27

Table 2: Selected layers for each behavioral category based on attribution patching results. For the Llama model, we select the layer with the maximum score and the two preceding layers. For ”Example Testing” we include an additional layer, as the curve is flatter compared to the other categories. For the Qwen model, we choose the layer with the maximum value and the two preceding layers, and if there is a second peak we include the layer of that peak too.

control the model's reasoning patterns. Section C includes a full example of positive and negative steering on the “Adding Knowledge” vector for the DeepSeek-R1-Distill-Qwen-14B model.

As shown in Figure 4, positive steering increases behaviors such as backtracking, uncertainty estimation, and example testing, while negative steering reduces them. These effects are consistent across both DeepSeek-R1-Distill models, reinforcing the hypothesis that Thinking LLMs encode these reasoning mechanisms as linear directions in their activation space. Our findings confirm that steering vectors provide a reliable and efficient method for interpreting and controlling the internal reasoning dynamics of thinking large language models.

5 Related Work

Recent work has explored methods for steering and interpreting language models by identifying meaningful directions or features within their internal representation spaces. Subramani et al. (2022) show that extracting latent steering vectors from pretrained language models can systematically alter generated text. Similarly, Turner et al. (2023) propose *activation engineering*, modifying model activations at inference time to control outputs, in contrast to prompt engineering or fine-tuning.

Extending this line of research, Panickssery et al. (2023) introduce Contrastive Activation Addition (CAA), which derives a “contrastive” vector by averaging differences in residual stream activations between positive and negative examples of a target behavior. Adding this vector to a model's activations elicits more desirable outputs without retraining. Beyond these methods, Zou et al. (2023) propose *Representation Engineering*, offering a top-down approach to refining and analyzing internal representations for greater transparency. Additionally, Arditi et al. (2024a) has shown that refusal behavior can be localized to a single direction in latent space, enabling targeted interventions that minimally affect other capabilities.

In parallel, some work has examined how inference-time manipulations can improve factual correctness. For instance, Li et al. (2023) propose an *inference-time intervention* to encourage more truthful responses without additional training. Overall, these approaches highlight an emerging trend toward fine-grained control of language models through systematic manipulation of their internal representations.

6 Conclusion and Future Work

This work presents an analysis of the unique mechanisms underlying the reasoning process of thinking LLMs, with a specific focus on DeepSeek-R1. Our study reveals distinct behavioral patterns that differentiate thinking models from traditional LLMs, including expressing their own uncertainty, backtracking, and example testing. Furthermore, we demonstrate that these behaviors can be effectively captured and controlled using steering vectors.

Our key findings indicate that:

- Thinking models, such as DeepSeek-R1, implement fundamentally different reasoning mechanisms compared to non-thinking models, characterized by explicit uncertainty handling and self-correction capabilities.
- These reasoning behaviors can be isolated to specific directions in the model's activation space, allowing precise control through steering vectors.
- The effectiveness of our steering approach is validated across a diverse set of 300 tasks, demonstrating robust control over the model's reasoning process.

These results not only enhance our understanding of how thinking models reason but also provide practical tools for modulating their reasoning capabilities. The ability to adjust specific aspects of the reasoning process through steering vectors opens new possibilities for adapting these models to different tasks and requirements.

Despite these promising results, our work has several limitations that suggest directions for future research. The current automated annotation process using GPT-4o, while efficient, has occasionally produced false positives and false negatives in identifying reasoning patterns. Future work should focus on developing more robust annotation methods, potentially incorporating multiple models or human validation to improve accuracy. Additionally, while our analysis centers on DeepSeek-R1 and its distilled variants, the generalization of these findings to other thinking models, such as OpenAI's o1 or Qwen's QwQ, remains an open question. Extending this research to a broader range of models would provide deeper insights into the universality of thinking mechanisms and their practical applications. By addressing these limitations, future research can further advance the understanding and control of thinking models, paving the way for more reliable and adaptable AI systems.

REFERENCES

Andy Arditi, Oscar Obeso, Aaquib Syed, Daniel Paleka, Nina Panickssery, Wes Gurnee, and Neel Nanda. Refusal in Language Models Is Mediated by a Single Direction. *arXiv*, June 2024a. doi: 10.48550/arXiv.2406.11717.

Andy Arditi, Oscar Obeso, Aaquib Syed, Daniel Paleka, Nina Panickssery, Wes Gurnee, and Neel Nanda. Refusal in language models is mediated by a single direction. *arXiv*, abs/2406.11717, 2024b. URL https://arxiv.org/abs/2406.11717.

François Chollet. OpenAI o3 Breakthrough High Score on ARC-AGI-Pub, December 2024. URL https://arcprize.org/blog/oai-o3-pub-breakthrough.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement 'learning, 2025.

GDM. Gemini flash thinking: Gemini 2.0 Flash Thinking Experimental, 2024. URL https://deepmind.google/technologies/gemini/flash-thinking/.

Mike Knoop. R1-Zero and R1 Results and Analysis, January 2025. URL https://arcprize.org/blog/r1-zero-r1-results-analysis.

Kenneth Li, Oam Patel, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Inference-Time Intervention: Eliciting Truthful Answers from a Language Model. *arXiv*, June 2023. doi: 10.48550/arXiv.2306.03341.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 17359–17372. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/6f1d43d5a82a37e89b0665b33bf3a182-Paper-Conference.pdf.

Neel Nanda. Attribution patching: Activation patching at industrial scale. 2023. https://www.neelnanda.io/mechanistic-interpretability/attribution-patching.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show your work: Scratchpads for intermediate computation with language models, 2021.

OpenAI. Learning to reason with LLMs, 9 2024. URL https://openai.com/index/learning-to-reason-with-llms/.

Nina Panickssery, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Matt Turner. Steering Llama 2 via Contrastive Activation Addition. *arXiv*, December 2023. doi: 10.48550/arXiv.2312.06681.

Qwen Team. Qwq: Reflect deeply on the boundaries of the unknown, 11 2024. URL https://qwenlm.github.io/blog/qwq-32b-preview/.

Laria Reynolds and Kyle McDonell. Prompt programming for large language models: Beyond the few-shot paradigm, 2021.

Nishant Subramani, Nivedita Suresh, and Matthew E. Peters. Extracting Latent Steering Vectors from Pretrained Language Models. *arXiv*, May 2022. doi: 10.48550/arXiv.2205.05124.

Aaquib Syed, Can Rager, and Arthur Conmy. Attribution patching outperforms automated circuit discovery. In *NeurIPS Workshop on Attributing Model Behavior at Scale*, 2023. URL https://openreview.net/forum?id=tiLbFR4bJW.

Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, Hoagy Cunningham, Nicholas L Turner, Callum McDougall, Monte MacDiarmid, C. Daniel Freeman, Theodore R. Sumers, Edward Rees, Joshua Batson, Adam Jermyn, Shan Carter, Chris Olah, and Tom Henighan.

Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. *Transformer Circuits Thread*, 2024. URL https://transformer-circuits.pub/2024/scaling-monosemanticity/index.html.

Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J. Vazquez, Ulisse Mini, and Monte MacDiarmid. Steering Language Models With Activation Engineering. *arXiv*, August 2023. doi: 10.48550/arXiv.2308.10248.

Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J. Vazquez, Ulisse Mini, and Monte MacDiarmid. Activation addition: Steering language models without optimization. *arXiv*, abs/2308.10248, 2024. URL https://arxiv.org/abs/2308.10248.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, Shashwat Goel, Nathaniel Li, Michael J. Byun, Zifan Wang, Alex Mallen, Steven Basart, Sanmi Koyejo, Dawn Song, Matt Fredrikson, J. Zico Kolter, and Dan Hendrycks. Representation Engineering: A Top-Down Approach to AI Transparency. *arXiv*, October 2023. doi: 10.48550/arXiv.2310.01405.

A DETAILS ON THE ANNOTATION PROCESS

We use the following prompt to automatically annotate LLM responses:

```
Please split the following reasoning chain of an LLM into
annotated parts using labels and the following format ["label
"]...["end-section"]. A sentence should be split into multiple
parts if it incorporates multiple behaviours indicated by the
labels.

Available labels:
0. initializing -> The model is rephrasing the given task and
states initial thoughts.
1. deduction -> The model is performing a deduction step based on
its current approach and assumptions.
2. adding-knowledge -> The model is enriching the current approach
 with recalled facts.
3. example-testing -> The model generates examples to test its
current approach.
4. uncertainty-estimation -> The model is stating its own
uncertainty.
5. backtracking -> The model decides to change its approach.

The reasoning chain to analyze:
{thinking_process}

Answer only with the annotated text. Only use the labels outlined
above. If there is a tail that has no annotation leave it out.
```

B ANNOTATED EXAMPLE

Riddle: What has cities, but no houses; forests, but no trees; and rivers, but no water?

DeepSeek R1 Response:

```
Okay, so I came across this riddle. At first glance, it seems
tricky, but I can break it down.
Let me analyze each part separately. The first part: "What has
cities, but no houses." The answer must be something that contains
 cities but lacks houses. Cities have buildings, roads, and
infrastructure, but without houses, it must be something that
represents a city rather than an actual one.
What about the second part: "forests, but no trees"? Forests are
made of trees, so if there are no trees, it isn't really a forest.
 But perhaps it's something that symbolizes a forest without
actually being one.
Finally, "rivers, but no water." Rivers are defined by flowing
water, so if there's no water, it might be referring to a
representation of a river rather than a real one.
Considering all three parts together, the answer must be something
 that includes symbolic representations of cities, forests, and
rivers but lacks their real-world components. A map fits this
description-it has cities, forests, and rivers marked on it, but
they are not real.
```

Automatically annotated response:

```
["initializing"] Okay, so I came across this riddle. At first
glance, it seems tricky, but I can break it down. ["end-section"]
```

```
["deduction"] Let me analyze each part separately. The first part:
 "What has cities, but no houses." The answer must be something
that contains cities but lacks houses. ["end-section"]
["adding-knowledge"] Cities have buildings, roads, and
infrastructure, but without houses, it must be something that
represents a city rather than an actual one. ["end-section"]
["example-testing"] Maybe it's a city without residential
buildings, but that doesn't fit perfectly. ["end-section"]
["backtracking"] Wait, perhaps it refers to something that *
displays* cities rather than being a real city. ["end-section"]
["deduction"] Moving on: "forests, but no trees." ["end-section"]
["adding-knowledge"] Forests are made of trees, so if there are no
 trees, it isn't really a forest. ["end-section"]
["example-testing"] Unless it's an area that used to be a forest
but was deforested-but that wouldn't be called a forest anymore.
["end-section"]
["uncertainty-estimation"] That might be a stretch, though. ["end-
section"]
["deduction"] The third part: "rivers, but no water." ["end-
section"]
["adding-knowledge"] Rivers are defined by flowing water. If there
's no water, it might be referring to a representation of a river
rather than a real one. ["end-section"]
["example-testing"] Maybe it's a dry riverbed, but then it's not
technically a river anymore. ["end-section"]
["deduction"] Putting it all together, the answer must be
something that includes symbolic representations of cities,
forests, and rivers but lacks their real-world components. ["end-
section"]
["deduction"] A map fits this description-it has cities, forests,
and rivers marked on it, but they are not real. ["end-section"]
```

C STEERED EXAMPLE (ADDING KNOWLEDGE)

Task:

```
What's the probability of drawing exactly two hearts when dealing
three cards from a standard deck?
```

Original Response:

```
["initializing"]
Okay, so I need to figure out the probability of drawing exactly
two hearts when dealing three cards from a standard deck.
["end-section"]

["uncertainty-estimation"]
Hmm, let's see.
["end-section"]

["adding-knowledge"]
I remember that probability problems often involve combinations,
["end-section"]

["deduction"]
so maybe I should start by recalling how combinations work.
["end-section"]

["adding-knowledge"]
```

```
First, a standard deck has 52 cards, right? And there are 13
hearts in the deck, one for each rank.
["end-section"]

["deduction"]
So, if I'm drawing three cards, I want exactly two of them to be
hearts and one to be a non-heart. That makes sense.
["end-section"]

["deduction"]
I think the way to approach this is to calculate the number of
favorable outcomes (i.e., drawing exactly two hearts and one non-
heart) and then divide that by the total number of possible
outcomes when drawing three cards.
["end-section"]

["adding-knowledge"]
So, probability is generally favorable over total.
["end-section"]

["initializing"]
Let me write that down:
["end-section"]

["deduction"]
Probability = (Number of ways to choose 2 hearts and 1 non-heart)
/ (Total number of ways to choose 3 cards from the deck). Okay, so
 I need to compute both the numerator and the denominator.
["end-section"]

["deduction"]
Starting with the denominator, the total number of ways to choose
3 cards from 52.
["end-section"]

["adding-knowledge"]
That's a combination problem because the order doesn't matter when
 you're dealing cards.
["end-section"]

["adding-knowledge"]
The formula for combinations is C(n, k) = n! / (k! * (n - k)!),
where n is the total number, and k is the number you're choosing.
["end-section"]

["deduction"]
So, the denominator is C(52, 3). Let me compute that:
["end-section"]

["deduction"]
C(52, 3) = 52! / (3! * (52 - 3)!) = (52 * 51 * 50) / (3 * 2 * 1) =
 (52 * 51 * 50) / 6
["end-section"]

["initializing"]
Let me calculate that:
["end-section"]

["uncertainty-estimation"]
```

```
52 divided by 6 is approximately 8.666..., but maybe I should
compute it step by step.
["end-section"]

["deduction"]
First, 52 * 51 = 2652
["end-section"]

["deduction"]
Then, 2652 * 50 = 132600
["end-section"]

["deduction"]
Now, divide by 6: 132600 / 6 = 22100
["end-section"]

["deduction"]
So, the total number of ways is 22,100. Got that.
["end-section"]
```

Positively Steered Response

```
["initializing"]
Okay, so I need to figure out the probability of drawing exactly
two hearts when dealing three cards from a standard deck.
["end-section"]

["uncertainty-estimation"]
Hmm, probability problems can be tricky,
["end-section"]

["adding-knowledge"]
but I remember that probability often involves combinations and
the concept of favorable outcomes over total possible outcomes.
["end-section"]

["adding-knowledge"]
Let me start by recalling that a standard deck has 52 cards,
divided into four suits: hearts, diamonds, clubs, and spades.
["end-section"]

["adding-knowledge"]
Hearts are one of the four suits, and there are 13 cards in each
suit.
["end-section"]

["deduction"]
So, in a standard deck, there are 13 hearts, and the rest of the
cards are non-hearts, which would be 52 minus 13, so 39 non-heart
cards.
["end-section"]

["adding-knowledge"]
When dealing with probabilities involving card draws, especially
without replacement, the concept of combinations is really
important.
["end-section"]

["adding-knowledge"]
```

```
Combinations are used to determine the number of ways to choose a
subset of items from a larger set, and the formula for
combinations is C(n, k) = n! / (k! * (n - k)!), where n is the
total number of items, and k is the number of items you're
choosing.
["end-section"]

["adding-knowledge"]
In this problem, we're dealing with hypergeometric distribution,
which deals with successes and failures in draws without
replacement.
["end-section"]

["adding-knowledge"]
The hypergeometric probability formula is used to calculate the
probability of k successes (in this case, drawing hearts) in n
draws (here, n is 3 because we're dealing three cards).
["end-section"]

["adding-knowledge"]
The formula is P(X = k) = (C(K, k) * C(N - K, n - k)) / C(N, n),
where:
- N is the total number of items in the population, which in this
case is the deck of 52 cards.
- K is the number of success states in the population, which here
is the number of hearts, so K = 13.
- n is the number of draws, which is 3 for this problem.
- k is the number of observed successes, which is the number of
hearts we want to draw. In this case, we're interested in exactly
two hearts, so k = 2.
["end-section"]

["deduction"]
So, for exactly two hearts in three card draws, we need to
calculate the probability of drawing exactly 2 hearts and 1 non-
heart card.
["end-section"]

["deduction"]
The formula for this specific case would be: P(X = 2) = [C(13, 2)
* C(39, 1)] /
["end-section"]
```

Negatively Steered Response

```
["initializing"]
First, I need to determine the number of ways to draw exactly two
hearts from the deck.
["end-section"]

["adding-knowledge"]
There are 13 hearts in the deck, and I want to choose 2 of them.
["end-section"]

["deduction"]
The number of ways to do this is 13 choose 2.
["end-section"]

["adding-knowledge"]
```

```
Then, I need to choose 1 card from the remaining 27 non-heart
cards.
["end-section"]

["deduction"]
The number of ways to do this is 27 choose 1.
["end-section"]

["deduction"]
Finally, I multiply these two numbers together to get the total
number of favorable outcomes.
["end-section"]

["deduction"]
Then, I divide by the total number of possible ways to draw 3
cards from the deck, which is 52 choose 3.
["end-section"]

["deduction"]
This gives me the probability of drawing exactly two hearts.
["end-section"]
```