LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
CENTRUM FÜR INFORMATIONS- UND SPRACHVERARBEITUNG
STUDIENGANG COMPUTERLINGUISTIK

Thesis

for the Seminar:

Foundation Model Frontiers

CIS, LMU Munich

A Recurrent Depth Approach to Latent Reasoning

from

Adrian Mülthaler

Date: April 30, 2026

Abstract

This thesis examines how large language models (LLMs) can be improved in their reasoning abilities by comparing two paradigms: explicit reasoning through Chain-of-Thought (CoT) and latent reasoning. CoT enables interpretability by making intermediate steps explicit, but it is computationally costly and limited to reasoning that can be verbalized. Latent reasoning, in contrast, operates entirely within hidden states, offering greater efficiency and flexibility but less transparency. A particular focus is placed on the recurrent depth approach, which introduces recurrence into transformer architectures. This method allows iterative refinement of internal representations and shows promising results in domains such as mathematics and coding, achieving competitive performance with fewer parameters. The findings suggest that CoT and latent reasoning should be viewed as complementary rather than competing strategies, and that future research may benefit from integrating the interpretability of explicit reasoning with the efficiency of latent computation.

Contents

1 Introduction

This thesis explores a novel model architecture for large language models (LLMs)that incorporates recurrence to enable latent reasoning, based on the framework introduced by Geiping et al. (2025). It begins with a concise overview of large reasoning models (LRMs), and continues by introducing verbose reasoning strategies and then the recurrent depth approach to latent reasoning.

The thesis is organized in the following way: Section 1 presents briefly the overall motivation and background for reasoning in LLMs. Section 2 introduces the principles and mechanisms of CoT prompting and CoT in general. Section 3 presents the concept of latent reasoning and summarizes the methodology proposed by Geiping et al. (2025). Finally, Section 4 offers concluding remarks and reflections.

1.1 Reasoning

Reasoning is the process by which one draws conclusions from information. At its core, it is about moving from what we know (or think we know) to new knowledge, beliefs, or actions. In cognitive science, reasoning is studied as a fundamental aspect of human thought. In AI, the focus is on formalizing this process and replicating it in computational systems.

Early AI systems approached reasoning through symbolic methods, in which explicit rules and logical inference engines were used to derive conclusions. While these methods offered transparency and formal guarantees, they struggled to scale to complex real-world data. In contrast, neural models like LLMs do not rely on explicit symbolic rules. Instead, they learn reasoning patterns from vast amounts of data. Research suggests that certain forms of reasoning can **emerge** as a byproduct of scaling model size and training data (Wei et al., 2022b). However, since further scaling is becoming increasingly infeasible due to energy and resource costs, there is a need to develop smaller, specialized models, which are tailored specifically for reasoning.

1.2 Explicit vs. Latent Reasoning

One broad scope of this thesis is to compare two paradigms of creating large reasoning models: explicit and latent.

Explicit reasoning refers to the way that LLMs explain their intermediate steps in natural language by generating reasoning text. This means that models are encouraged to produce tokens explaining how they arrive at their final answer. A method of doing this is by letting the model produce a chain of thought. It is worth noting that in this thesis, we see reasoning as explicit, even if the generated reasoning tokens are hidden from the user.

Latent reasoning refers to the way LLMs do reasoning in a latent space, which is not directly shown in the output. Latent reasoning in LLMs can be achieved through adjustments to the architecture, or by targeted fine-tuning or distillation. Zhu et al. (2025) divide latent reasoning into two types: **activation-based** and **hidden state-based** methods.

Activation-based methods (vertical recurrent) follow the principle of iteratively refining representations by creating a recurrent computational flow. A foundational approach to this is a loop-based architecture, to which the recurrent depth approach (Geiping et al., 2025) can be assigned.

Hidden state-based methods (horizontal recurrent), on the other hand, operate along the temporal dimension. In contrast to activation-based methods, which expand the amount of computation (the layer depth) for each time step, hidden-state methods try to compress previous information into a single vector or matrix. This is done natively by the standard transformer via the KV-cache, which stores all previous input token. However, it faces the bottleneck of growing linearly with sequence length, leading to very high memory consumption for long sequences.

1.3 Test-Time Compute

An important concept for this thesis is **test-time compute**. It refers to the amount of computation a model performs when generating an answer to a specific input, as opposed to the computation that is needed to train the model. In the context of reasoning, this concept is important because allocating more compute can improve performance on complex, multi-step tasks without changing the model's parameters. In other words, increasing the amount of compute at inference time, also called **test-time scaling (TTS)**, leads the model to "think" for a longer time or reason more thoroughly before producing an answer.

As stated earlier, TTS can be employed in an **explicit, verbose way**, or an **implicit, latent way**. In the former, TTS is done by increasing the number of tokens processed and generated during inference, either by encouraging models to explain their thought process or even just by letting the model repeat a simple thinking token like '<T>' several times (Herel and Mikolov, 2024). In the latter, TTS can be achieved by increasing the amount of calculations done internally, without outputting tokens.

1.4 Motivation

LLMs have shown remarkable improvements in reasoning ability, yet how we might make that reasoning more effective remains an open question. While explicit approaches like CoT provide transparency, they are computationally expensive and limited to reasoning that can be verbalized. At the same time, emerging methods like latent reasoning suggest that powerful internal computation can occur without ever producing intermediate text, potentially enabling more efficient and flexible forms of reasoning.

2 Chain-of-Thought

Chain-of-thought (CoT) refers to the general idea of large language models producing **explicit reasoning** steps before arriving at a final answer. In practice, this means the model generates additional tokens to allocate more computation at test time and to decompose complex problems into intermediate steps.

Although the term was originally introduced in the context of chain-of-thought prompting, in current usage, CoT is often used more broadly to describe this explicit reasoning process, regardless of how it is triggered.

2.1 Chain-of-Thought Prompting

In contrast to CoT, chain-of-thought prompting has a more specific meaning: it provides the model with examples (shots) of reasoning steps in the prompt to encourage similar behavior. It was originally proposed by Wei et al. (2023). Here, the authors propose a few-shot prompting method consisting of triplets of the form $\langle \text{input}, \textit{chain of thought}, \text{output} \rangle$. In each triple, the *chain-of-thought* is a series of intermediate steps that lead to the desired output.

This combines the strengths of two ideas that have been previously studied in the context of reasoning: **1.** in-context few-shot learning via prompting, which showed poor improvements on reasoning tasks (Brown et al., 2020; Rae et al., 2022) (this also holds for a zero-shot setup on instrctuion tuned models (Wei et al., 2022a)) and **2.** training models from scratch (Ling et al., 2017) or finetuning pretrained models (Cobbe et al., 2021) to give them the ability to generate natural language intermediate steps benefiting reaoning capabilities. For the latter, it is very costly to create large, curated datasets because they need to contain complex rationales instead of simple input-output pairs. Wei et al. (2023) propose to show the model only a few high-quality rationale examples at inference time, avoiding the need for a large fine-tuning dataset and potentially enabling the model to use natural language rationales to arrive at an answer.

This method has several properties that are advantageous for reasoning. First, it allows models to break down multi-step problems into smaller, intermediate steps. This means that the model can devote more computational resources at test time. Second, CoT offers a more interpretable view of how the model is "thinking". By examining the reasoning steps, we can get a sense of how it arrived at its answer and even identify where it might have gone wrong. Third, this technique is applicable for a variety of tasks, such as solving math problems, applying commonsense reasoning, and performing symbolic manipulations. Finally, CoT reasoning can often be triggered in large pre-trained language models simply by including example reasoning sequences in the few-shot prompts provided to the model.

2.2 Chain-of-Thought Reasoning Models

While CoT prompting can encourage language models to reason more effectively, it relies on the assumption that the prompt includes high-quality CoT examples. In real-world settings, however, users typically provide only a short question or instruction, without any examples containing reasoning steps. One way to address this is to fine-tune models so they learn to produce useful reasoning steps even without being explicitly prompted with examples. This is often done by instruction-tuning a model using reinforcement learning (RL) with human preferences. By optimizing for reasoning quality directly, these CoT-trained models can achieve better results on complex tasks in a more realistic, example-free setting. Three examples of such models are presented in the following paragraphs.

OpenAI o1 (OpenAI, 2024), for example, is such a CoT LRM. For this model, OpenAI chose to hide the CoT, justifying their decision by stating that it improves user experience and gives the model a competitive advantage. In addition, they want to monitor the model's CoT and understand

its thought process in an unaltered form, which is why they have not trained the CoT itself on user preferences.

DeepSeek-R1 (DeepSeek-AI et al., 2025) would be another example. Here, the authors apply RL directly to a base model, without relying on supervised fine-tuning (SFT) as a preliminary step, to develop **DeepSeek-R1-Zero**. Trained purely via RL, the authors claim it allows the model to explore CoT for solving complex problems. Building on this, they introduce a pipeline for **DeepSeek-R1** that includes two RL stages and two SFT stages. Outputs from R1-Zero are used to generate preference data, which is then used to align R1 with human preferences.

Gemini-2.5 Thinking (Comanici et al., 2025) is trained using RL to increase inference-time compute during a "thinking" stage in which the model can perform tens of thousands of forward passes before responding to a query. The model determines the duration of this hidden CoT process itself. The authors also introduced a "thinking budged", which constrains the model to a set number of thinking tokens and allows the user to trade off performance for cost.

2.3 Limitations

While LRMs fine-tuned on CoT data show promising results, this reasoning paradigm has notable limitations. First, effective CoT reasoning requires a model to be trained on very long domain-specific demonstrations. This requires the model to work on extremely long context windows, which results in high memory and energy costs. Second, CoT has the constraint that the reasoning must always be projected down to a single verbalized next token. This has the drawback that it can only model human thinking, potentially overlooking aspects of human thought that are less easily verbalized.

3 Latent Reasoning: A Recurrent Depth Approach

Latent reasoning poses an alternative to the explicit step-by-step approach used in CoT. Rather than verbalizing internal reasoning steps, models are trained to natively "think" directly within their continuous latent space. This allows them to capture facets of human cognition that defy verbalization, such as physical intuition, spatial thinking, or long-term planning.

In this section, we summarize the work of Geiping et al. (2025) as an example of how latent reasoning can be implemented. Their approach includes modifying the transformer architecture by introducing a recurrent unit, which iteratively updates the model's internal state, and creates a high-dimensional vector space enabling the deep exploration of multiple directions simultaneously, leading to a system capable of exhibiting novel and complex reasoning behavior.

3.1 Related Work

The architecture designed by Geiping et al. (2025) falls into the category of an activation-based, architectural latent reasoning model (Zhu et al., 2025). As later shown in Section 3.2 it follows a Pre/Loop/Coda structure, which seperates input encoding, iterative reasoning, and output decoding, and enables a modular and more interpretable computation flow. The following paragraphs will introduce three related architectures.

Universal Transformer (UT) (Dehghani et al., 2019) builds on the standard encoder–decoder transformer architecture but modifies it by introducing recurrence over depth. Instead of stacking a fixed number of distinct layers as in the vanilla transformer, the UT applies the same self-attention and transition block repeatedly across steps. In this way, each position in the input or output sequence is iteratively refined, giving the model both the parallelism of Transformers and the iterative processing bias of RNNs.

Relaxed Recursive Transformer (RRT) (Bae et al., 2025) extends the idea of parameter sharing in transformers by introducing a recursive architecture with layer-wise low-rank adaptations (LoRA). Instead of having each layer maintain its own full set of parameters, the RRT recursively reuses a shared core set of transformer parameters across layers, while allowing lightweight, trainable LoRA modules to adapt each layer's behavior. Conceptually, the recursion lets the network apply the same computation multiple times over the representations, similar in spirit to the UT, but more 'relaxed' because each step can still adapt via LoRA, making the model both lightweight and flexible.

AlgoFormer (Gao et al., 2025) introduces a structured transformer design based on three modules: TF_{pre}, TF_{loop}, and TF_{post}. An input sequence $\mathbf{X}$ is first processed by TF_{pre}, then iteratively refined by repeatedly applying the shared-weight TF_{loop} block, and finally passed through TF_{post} to produce the output. This Pre/Loop/Coda structure (which is also used by Geiping et al. (2025)) is flexible because of its modular design.

3.2 Model Architecture

The model is structured around decoder-only transformer blocks, which are structured into three functional groups: the *prelude* P, the *recurrent block* R, and the *Coda* C. This setup is shown in Figure 1. P is meant to embed the input data into the latent space. The core R block then iterates over the hidden state and gives the final output to C, which un-embeds the hidden state from the latent space and contains the prediction head of the model.

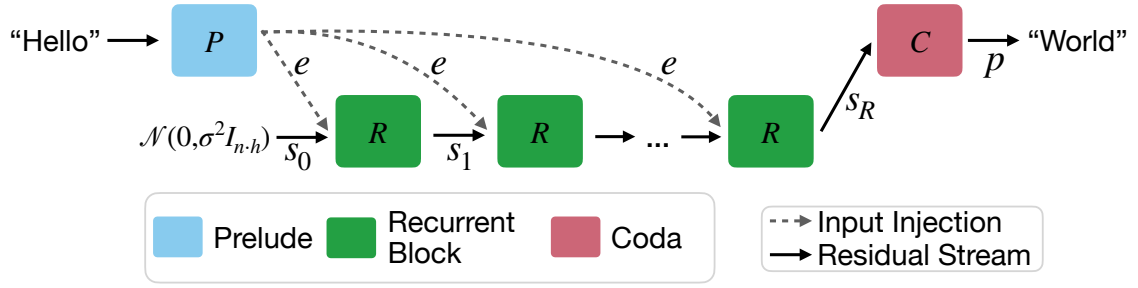


Figure 1: A visualization of the Architecture. (Geiping et al., 2025)

The Transformer Decoder *T* is part of every functional block, each containing multiple layers of *T*. It follows standard transformer design.

Each layer contains masked self-attention using Rotary Positional Embeddings (RoPE) (Su et al., 2023). RoPE encodes token positions by rotating query and key vectors in the attention mechanism rather than by adding learned or sinusoidal position vectors to token embeddings. Each pair of dimensions in Q/K is interpreted as a 2D vector and multiplied by a rotation matrix whose angle depends on the token's absolute position.

For the normalization function, they used Root Mean Square Layer Normalization (RMSNorm) (Zhang and Sennrich, 2019). RMSNorm normalizes activations using only the root mean square of the input ($\frac{1}{\sqrt{mean(x^2)+\epsilon}}$) and a learned per-channel scaling vector, but unlike LayerNorm does not subtract the mean. Removing the mean subtraction slightly reduces computation and can improve stability with large batch sizes or deep networks while preserving the benefits of scale-invariance.

And for the fully connected layer they used a gated SiLU Multi Layer Perceptron (MLP) (Shazeer, 2020). SiLU (also called swish) is a smooth, non-monotonic activation defined as $x \cdot sigmoid(x)$. It yields better gradient flow and empirical performance than ReLU in many transformer MLPs because of its smoother curvature and nonlinearity.

Putting this all together, this results in a transformer decoder block, which can be written as

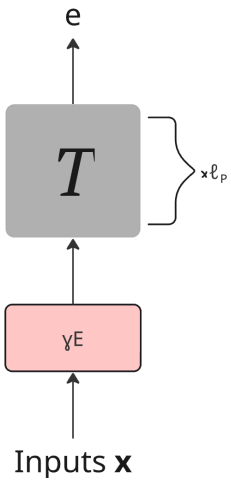
$$\hat{\mathbf{x}}_l = n_2\left(\mathbf{x}_{l-1} + \text{Attn}(n_1(\mathbf{x}_{l-1}))\right) \tag{1}$$

$$\mathbf{x}_l = n_4\left(\hat{\mathbf{x}}_l + \text{MLP}(n_3(\hat{\mathbf{x}}_l))\right). \tag{2}$$

where $\mathbf{x}_{l-1}$ and $\mathbf{x}_l$ denote the input and output hidden states of layer l, $n_1 \ldots n_4$ are the aforementioned layer normalization operators , Attn$(\cdot)$ is the masked multi-head self-attention, and MLP$(\cdot)$ aforementioned SiLU MLP.

Both attention and MLP sublayers operate on normalized inputs (pre-norm), and their outputs are added to the layer input via residual connections before a final normalization is applied.

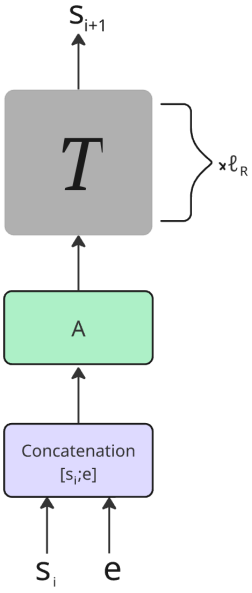
The following sections will explain each functional block of the architecture in a bit more detail. Each is accompanied by a self-created figure, visualizing the process of that block. Each figure contains a *T*-block which represents the decoder block described here.



Prelude

The *Prelude* P serves the purpose of encoding the inputs $\mathbf{x}$ into the latent space. It first encodes $\mathbf{x}$ as $\gamma E\mathbf{x}$, with E being the embedding matrix and γ being the embedding scale. Then the decoder block T is applied ℓ_P times. This results in the latent input $\mathbf{e}$.

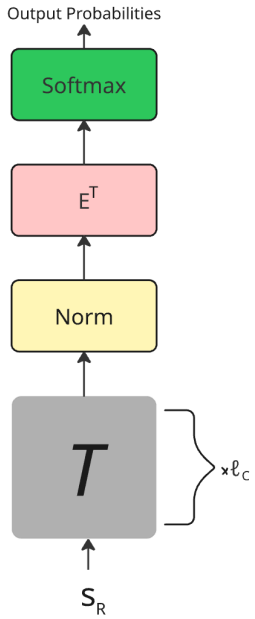
Recurrent Block



The *recurrent Block* R serves the purpose of refining the latent state. To create the new latent state $\mathbf{s}_{i+1}$, it starts by concatenating the latent input $\mathbf{e}$ with the last latent state $\mathbf{s_i}$ which are mapped back into the hidden dimension h using an adapter matrix $A : \mathbb{R}^{2h} \to \mathbb{R}^h$. The authors also tried different methods combining $\mathbf{e}$ and $\mathbf{s_i}$ but found that concatenation works best at scale. After the adapter matrix T is again applied ℓ_R times.

The initial state $\mathbf{s}_0$ is obtained by sampling randomly from a normal distribution, i.e., $\mathbf{s}_0 \sim \mathcal{N}(\mathbf{0}, \sigma^2 I_{n \cdot h})$.

It is important to note, that in this architecture the latent input $\mathbf{e}$ from P is **injected at each recurrent step**. This is the main point that sets this architecture apart from the Algoformer (mentioned in Section 3.1). The authors argue that by doing this and by randomly initializing $\mathbf{s}_0$ the recurrence is stabilized and achieves path independence (it leads to the same result independent of initialization). Also, with this the setup is less prone to the vanishing gradient problem compared to traditional recurrent networks. At backpropagation, P receives gradient signals at every recurrent step, since the latent input is injected every time.



Coda

The *Coda* C gets the last latent state $\mathbf{s}_R$ from the recurrent unit. Then it first applies ℓ_C decoder layers T, after which it normalizes again. Then it projects into the vocabulary by using tied embeddings E^T. The output probabilities for the next token can then be calculated by applying a softmax to these logits.

The size of a model can now be defined by (ℓ_P, ℓ_R, ℓ_C) and by the number of recurrences r, which may vary at each forward pass, i.e., at each generated token.

3.3 Training

During training, the number of recurrences for each step is sampled from a log-normal Poisson distribution Λ. This stochastic scheduling prevents the model from overfitting to a fixed recurrence depth and encourages it to generalize across different amounts of computation at test time. The training objective is optimizing the expectation of the loss function L:

$$\mathcal{L}(\theta) = \mathbb{E}_{\mathbf{x}\in X}\mathbb{E}_{r\sim\Lambda} L\left(m_\theta(\mathbf{x}, r), \mathbf{x}'\right), \tag{3}$$

where $m_\theta(\mathbf{x}, r)$ is the model output given input sequence $\mathbf{x}$ and recurrence depth r, and $\mathbf{x}'$ represents that same sequence shifted one step to the left, i.e., the next token prediction target. In other words, the model is trained to perform well regardless of how many recurrent steps it is allowed to take, making it more flexible at inference time.

To keep the computational cost low during training, the authors employed **truncated backpropagation**. Instead of backpropagating through the entire unrolled sequence of recurrent steps, which would be very expensive, they only propagate gradients through the last 8 steps of the recurrent unit. This strikes a balance between efficiency and learning stability, allowing the model to capture long-range dependencies without exploding memory requirements. With this method, training was typically run with a randomly chosen recurrence depth r averaging around 33, meaning that during inference the recurrent block effectively stacks up to dozens of transformer layers.

For their **pretraining data** Geiping et al. (2025) selected a dataset mixture that maximized the potential for emergent reasoning behaviors. Consequently, their dataset was heavily skewed towards code and mathematical reasoning data.

For their experiments, the authors first validated the approach with a smaller prototype model $(\ell_P = 1, \ell_R = 4, \ell_C = 1)$ and $h = 1024$, confirming that recurrence-based depth scaling behaved as expected. They then scaled up to a larger configuration $(\ell_P = 2, \ell_R = 4, \ell_C = 2)$ and $h = 5280$. This may look small compared to many big modern transformers, but when the recurrent block is unrolled, e.g., 33 times, the model ends up with $2 + 4 \cdot 33 + 2 = 136$ layers. This large model results in **3.5B parameters** and is used to evaluate their method.

3.4 Results

For comparability, the authors compared their model against other open-source models trained on fully public datasets of similar size. Namely, they compared against Amber (Liu et al., 2023), Pythia (Biderman et al., 2023) and some OLMo 1&2 variants (Groeneveld et al., 2024).

On **standard lm-eval-harness** (Biderman et al., 2024) benchmarks, the recurrent-depth model surpasses the older Pythia series and performs at a level broadly comparable to the first OLMo-7B model, despite being much smaller in parameter count. However, it falls short of the stronger results achieved by later OLMo models. The authors suggest that this could be the case because they benefited from larger, more carefully curated training datasets. Given its smaller size, novel architecture, and relatively modest training budget, these results are promising, suggesting that latent recurrence can yield competitive general-purpose reasoning performance without scaling parameter count alone.

In **mathematical reasoning** tasks such as GSM8k (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021), it outperforms all tested models except OLMo-2, showing particular gains in complex reasoning setups.

In **coding benchmarks** like MBPP (Austin et al., 2021) and HumanEval (Chen et al., 2021), it beats all other general-purpose open-source models but does not match the performance of specialized code-focused models such as StarCoder2, which have been trained on far more targeted

code data. These results highlight that while the recurrent-depth approach is not yet the best in absolute terms, it offers strong math and coding capabilities relative to its size and training scale.

3.5 Discussion

When analyzing their model Geiping et al. (2025) found two interesting aspects, which are briefly mentioned in the following.

3.5.1 Advanced LLM capabilities

Interestingly, the model demonstrates several advanced LLM capabilities, each achievable in a **zero-shot** manner:

- **Adaptive Compute at Test-Time** - The model can decide, without special training, how much recurrent computation to spend on each token. It uses a simple early exit rule: if the Kullback–Leibler (KL) divergence between two successive recurrent steps drops below a fixed threshold, the model stops iterating and outputs the token. This allows it to save compute on easy predictions while allocating more iterations to harder reasoning cases.
- **Continuous Chain-of-Thought** - Instead of reinitializing the latent state $\mathbf{s}_0$ at each new token, the model can warm-start it with the final latent state $\mathbf{s}_R$ from the previous token. This creates a continuous stream of latent reasoning across tokens, allowing the model to carry internal computations forward and converge more quickly on later steps.
- **KV-Cache Sharing** - By keeping only the last 16 recurrent steps in the KV-cache and overwriting older ones in a rolling fashion, the model reduces memory use without significantly harming performance.
- **Self-Speculative Decoding** - Traditional speculative decoding uses a smaller draft model to propose tokens quickly, which are then verified by a larger model. Here, the same recurrent model can act as both: running fewer recurrent iterations to propose tokens rapidly, then verifying them with more iterations.

3.5.2 Trajectories in the Latent Space

The authors analyzed the **trajectories in the latent space**, meaning the evolution of a model's hidden state s as it iterates through its high-dimensional latent representations. To make these dynamics interpretable, they projected the trajectories into a lower-dimensional space using **Principal Component Analysis (PCA)**. This visualization revealed several distinct dynamical patterns:

Fixed Points where the trajectory converges to a single point and stabilizes. Within the context of reasoning, fixed points may reflect situations where the model has reached a confident, internally consistent solution that no longer changes with further computation.

Orbits which are cyclical trajectories where the state revisits similar regions repeatedly. These may represent iterative refinement loops, in which the model revisits and re-evaluates intermediate hypotheses before stabilizing. Such behavior could be analogous to (human) cognitive processes, where various possible interpretations could be cycled through possible interpretations until a resolution emerges.

Directional drifts which are trajectories that steadily move across the latent space without settling into a cycle or a fixed point. These can be interpreted as progressive reasoning chains, where each iteration builds upon the previous state. This suggests that the model could use this to implement a mechanism to count.

The presence of these geometric structures indicates that latent reasoning might not just be an opaque black box, but rather exhibits identifiable dynamical behaviors. This suggests that latent reasoning in this case could be described as a structured process, opening pathways for both analysis and control of reasoning dynamics.

3.6 Other Directions in Latent Reasoning

In this section the survey paper by Zhu et al. (2025) is referenced. As mentioned in Section 1.2, they divide latent reasoning into activation-based and hidden state–based methods. The recurrent-depth approach belongs to the former and falls under the subcategory of loop-based recurrence methods. Below, some alternative categories are presented.

3.6.1 Activation with Explicit Hidden-State Feedback

Unlike loop-based recurrence, which repeatedly refines token embeddings within the same layers, this family of models explicitly **feeds hidden states back into the input sequence**. Here, internal activations are reintroduced as new tokens, allowing the model to attend to its own intermediate computations.

Two examples are Coconut (Hao et al., 2024), which inserts a continuous thought vector from the previous decoding step as an extra input position, and CoTFormer (Mohtashami et al., 2024), which interleaves preliminary embeddings back into the sequence before rerunning the shared block stack. Both approaches share these key properties, that distinguish them from pure activation-based methods: they **recycle the same parameters** rather than expanding the architecture, and they **treat hidden states as sequence elements** bridging recurrence and memory.

3.6.2 Linear-State Recurrence

Linear-state recurrence represents one of the main directions in hidden state–based reasoning methods. These models maintain a compact matrix-valued hidden state $\mathbf{S}_t$, which is updated step by step along the temporal dimension. At each time step, the state is globally decayed and then updated with new information from the current token. A prominent example of this approach is Mamba-2 (Dao and Gu, 2024). The hidden state is updated through recurrence:

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \mathbf{k}_t \mathbf{v}_t^\top, \tag{4}$$

with $(\cdot)$ being the and the associative operator (e.g., Hadamard product, matrix multiplication) and defining how updates are combined. The symbols $\mathbf{v}_t$, $\mathbf{k}_t$, $\mathbf{q}_t$ are functions of the current input $\mathbf{x}_t$. The predicted output token at time t can be obtained using

$$\mathbf{o}_t = \mathbf{S}_t \mathbf{q}_t. \tag{5}$$

Although current linear-state models have not yet demonstrated clear gains in reasoning ability, their theoretical properties make them a promising direction. By efficiently compressing temporal information into a recurrent state and updating it, these approaches hint at a unifying principle: latent reasoning may emerge not only by stacking depth but also by treating hidden states as continuously optimized representations.

3.6.3 Gradient-State Recurrence

While linear-state recurrence relies on fixed update–decay rules, gradient-state methods reinterpret the hidden state as a set of fast weights that are explicitly optimized at each step. In this view, every token update is treated as a lightweight gradient descent step on a local objective, steering the hidden state toward the current key–value target. This means the choice of optimizer (SGD, Adam-like updates, etc.) directly affects how the memory behaves over time.

Several models have been built around this idea, but they converge conceptually under a general update rule: the hidden state is updated step by step using gradients, rather than fixed formulas:

$$\mathbf{S}_t = \alpha_t \mathbf{S}_{t-1} - \eta_t \nabla_{\mathbf{S}} \ell(\mathbf{S}_{t-1}; \mathbf{k}_t, \mathbf{v}_t) \tag{6}$$

The downside is that this process is hard to parallelize. Unlike linear-state models that support efficient scan-based parallelization, the gradient $\nabla \ell$ requires the previous state $\mathbf{S}_{t-1}$ for its update, limiting hardware efficiency. To mitigate this, researchers use **chunk-wise parallelization**. Inside each chunk, updates can be done in parallel, but between chunks the model still passes the final state forward.

By treating the hidden state as an online-optimized memory, these models provide a framework for self-iteration and adaptation, potentially enabling reasoning dynamics even in the absence of explicit input tokens.

4 Conclusion

This thesis explored how LLMs can be pushed toward more effective reasoning. Two different paradigms have been at the center of this discussion: explicit reasoning, namely Chain-of-Thought (CoT), where a model makes its reasoning by spelling out intermediate steps, and latent reasoning, where the reasoning happens entirely inside the model's hidden states without ever being verbalized.

CoT has shown that giving models space to 'think out loud' often leads to better results on complex tasks. It offers interpretability and transparency, but at the cost of efficiency: it requires generating long strings of reasoning tokens, which is slow and resource-heavy. Furthermore, it can only capture thought processes that are easily expressed in language. Latent reasoning, on the other hand, aims to achieve the same or even better outcomes without producing explicit reasoning text. Instead, it leverages the model's internal dynamics, opening up the possibility of faster, more flexible, and less constrained forms of reasoning.

This thesis focused mainly on one concrete method for enabling latent reasoning: the recurrent-depth approach. This architecture introduces recurrence into the transformer itself, allowing the model to refine its internal representations step by step before producing an output.

The key takeaway is that latent reasoning is both a promising hypothesis and an open challenge. On one hand, it offers an appealing way to create a novel, efficient RM. On the other hand, results are still not on par with CoT-trained LRM and latent reasoning also lacks the interpretability provided by CoT.

4.1 Future Directions

In my opinion, both ways of doing reasoning in LLMs are promising, but as we have seen, each one has its advantages and disadvantages. However, I believe these two approaches should not be seen necessarily as rivals but could complement each other in the future. New systems may combine the clarity of explicit reasoning with the efficiency and depth of latent computation to achieve more adaptive and capable AI.

Latent reasoning offers several promising directions for improving reasoning capabilities, such as loop-based, linear-state, and gradient-state recurrence methods. While all of these directions are worth exploring, the loop-based technique is in my opinion the most interesting. I believe the best way to create specialized RMs is by altering the architecture; in particular, refining a state at each time step, i.e., increasing depth, has the greatest potential.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program Synthesis with Large Language Models. ArXiv:2108.07732 [cs].

Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. 2025. Relaxed Recursive Transformers: Effective Parameter Sharing with Layer-wise LoRA. ArXiv:2410.20672 [cs].

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. 2023. Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling. ArXiv:2304.01373 [cs].

Stella Biderman, Hailey Schoelkopf, Lintang Sutawika, Leo Gao, Jonathan Tow, Baber Abbasi, Alham Fikri Aji, Pawan Sasanka Ammanamanchi, Sidney Black, Jordan Clive, Anthony DiPofi, Julen Etxaniz, Benjamin Fattori, Jessica Zosa Forde, Charles Foster, Jeffrey Hsu, Mimansa Jaiswal, Wilson Y. Lee, Haonan Li, Charles Lovering, Niklas Muennighoff, Ellie Pavlick, Jason Phang, Aviya Skowron, Samson Tan, Xiangru Tang, Kevin A. Wang, Genta Indra Winata, François Yvon, and Andy Zou. 2024. Lessons from the Trenches on Reproducible Evaluation of Language Models. ArXiv:2405.14782 [cs].

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. ArXiv:2005.14165 [cs].

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. ArXiv:2107.03374 [cs].

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. ArXiv:2110.14168 [cs].

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, Luke Marris, Sam Petulla, Colin Gaffney, Asaf Aharoni, Nathan Lintz, Tiago Cardal Pais, Henrik Jacobsson, Idan Szpektor, Nan-Jiang Jiang, Krishna Haridasan, Ahmed Omran, Nikunj Saunshi, Dara Bahri, Gaurav Mishra, Eric Chu, et al. 2025. Gemini 2.5: Pushing the Frontier with Advanced Reasoning, Multimodality, Long Context, and Next Generation Agentic Capabilities. ArXiv:2507.06261 [cs] version: 1.

Tri Dao and Albert Gu. 2024. Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality. ArXiv:2405.21060 [cs].

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, et al. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. ArXiv:2501.12948 [cs].

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. 2019. Universal Transformers. ArXiv:1807.03819 [cs].

Yihang Gao, Chuanyang Zheng, Enze Xie, Han Shi, Tianyang Hu, Yu Li, Michael K. Ng, Zhenguo Li, and Zhaoqiang Liu. 2025. AlgoFormer: An Efficient Transformer Framework with Algorithmic Structures. ArXiv:2402.13572 [cs].

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. 2025. Scaling up Test-Time Compute with Latent Reasoning: A Recurrent Depth Approach. ArXiv:2502.05171 [cs].

Dirk Groeneveld, Iz Beltagy, Pete Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Harsh Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, Shane Arora, David Atkinson, Russell Authur, Khyathi Raghavi Chandu, Arman Cohan, Jennifer Dumas, Yanai Elazar, Yuling Gu, Jack Hessel, Tushar Khot, William Merrill, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew E. Peters, Valentina Pyatkin, Abhilasha Ravichander, Dustin Schwenk, Saurabh Shah, Will Smith, Emma Strubell, Nishant Subramani, Mitchell Wortsman, Pradeep Dasigi, Nathan Lambert, Kyle Richardson, Luke Zettlemoyer, Jesse Dodge, Kyle Lo, Luca Soldaini, Noah A. Smith, and Hannaneh Hajishirzi. 2024. OLMo: Accelerating the Science of Language Models. ArXiv:2402.00838 [cs].

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. 2024. Training Large Language Models to Reason in a Continuous Latent Space. ArXiv:2412.06769 [cs].

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring Massive Multitask Language Understanding. ArXiv:2009.03300 [cs].

David Herel and Tomas Mikolov. 2024. Thinking Tokens for Language Modeling. ArXiv:2405.08644 [cs] version: 1.

Philip Nicholas Johnson-Laird. 2006. *How we reason*. Oxford University Press, Oxford.

Nora Kassner, Oyvind Tafjord, Hinrich Schütze, and Peter Clark. 2021. BeliefBank: Adding Memory to a Pre-Trained Language Model for a Systematic Notion of Belief. ArXiv:2109.14723 [cs].

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program Induction by Rationale Generation: Learning to Solve and Explain Algebraic Word Problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 158–167, Vancouver, Canada. Association for Computational Linguistics.

Zhengzhong Liu, Aurick Qiao, Willie Neiswanger, Hongyi Wang, Bowen Tan, Tianhua Tao, Junbo Li, Yuqi Wang, Suqi Sun, Omkar Pangarkar, Richard Fan, Yi Gu, Victor Miller, Yonghao Zhuang, Guowei He, Haonan Li, Fajri Koto, Liping Tang, Nikhil Ranjan, Zhiqiang Shen, Xuguang Ren, Roberto Iriondo, Cun Mu, Zhiting Hu, Mark Schulze, Preslav Nakov, Tim Baldwin, and Eric P. Xing. 2023. LLM360: Towards Fully Transparent Open-Source LLMs. ArXiv:2312.06550 [cs].

Amirkeivan Mohtashami, Matteo Pagliardini, and Martin Jaggi. 2024. CoTFormer: A Chain-of-Thought Driven Architecture with Budget-Adaptive Computation Cost at Inference. ArXiv:2310.10845 [cs].

ALLEN NEWELL. 2023. *HUMAN PROBLEM SOLVING*, reprint edition. ECHO POINT BOOKS & MEDIA,, S.l. OCLC: 1501823286.

Allen Newell and Herbert A. Simon. 1976. Computer science as empirical inquiry: symbols and search. *Communications of the ACM*, 19(3):113–126.

OpenAI. 2024. Learning to reason with LLMs.

Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, Eliza Rutherford, Tom Hennigan, Jacob Menick, Albin Cassirer, Richard Powell, George van den Driessche, Lisa Anne Hendricks, Maribeth Rauh, Po-Sen Huang, Amelia Glaese, Johannes Welbl, Sumanth Dathathri, Saffron Huang, Jonathan Uesato, John Mellor, Irina Higgins, Antonia Creswell, Nat McAleese, Amy Wu, Erich Elsen, Siddhant Jayakumar, Elena Buchatskaya, David Budden, Esme Sutherland, Karen Simonyan, Michela Paganini, Laurent Sifre, Lena Martens, Xiang Lorraine Li, Adhiguna Kuncoro, Aida Nematzadeh, Elena Gribovskaya, Domenic Donato, Angeliki Lazaridou, Arthur Mensch, Jean-Baptiste Lespiau, Maria Tsimpoukelli, Nikolai Grigorev, Doug Fritz, Thibault Sottiaux, Mantas Pajarskas, Toby Pohlen, Zhitao Gong, Daniel Toyama, Cyprien de Masson d'Autume, Yujia Li, Tayfun Terzi, Vladimir Mikulik, Igor Babuschkin, Aidan Clark, Diego de Las Casas, Aurelia Guy, Chris Jones, James Bradbury, Matthew Johnson, Blake Hechtman, Laura Weidinger, Iason Gabriel, William Isaac, Ed Lockhart, Simon Osindero, Laura Rimell, Chris Dyer, Oriol Vinyals, Kareem Ayoub, Jeff Stanway, Lorrayne Bennett, Demis Hassabis, Koray Kavukcuoglu, and Geoffrey Irving. 2022. Scaling Language Models: Methods, Analysis & Insights from Training Gopher. ArXiv:2112.11446 [cs].

Noam Shazeer. 2020. GLU Variants Improve Transformer. ArXiv:2002.05202 [cs].

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. 2023. RoFormer: Enhanced Transformer with Rotary Position Embedding. ArXiv:2104.09864 [cs].

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. 2022a. Finetuned Language Models Are Zero-Shot Learners. ArXiv:2109.01652 [cs].

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022b. Emergent Abilities of Large Language Models. ArXiv:2206.07682 [cs].

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. ArXiv:2201.11903 [cs].

Biao Zhang and Rico Sennrich. 2019. Root Mean Square Layer Normalization. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Rui-Jie Zhu, Tianhao Peng, Tianhao Cheng, Xingwei Qu, Jinfa Huang, Dawei Zhu, Hao Wang, Kaiwen Xue, Xuanliang Zhang, Yong Shan, Tianle Cai, Taylor Kergan, Assel Kembay, Andrew Smith, Chenghua Lin, Binh Nguyen, Yuqi Pan, Yuhong Chou, Zefan Cai, Zhenhe Wu, Yongchi Zhao, Tianyu Liu, Jian Yang, Wangchunshu Zhou, Chujie Zheng, Chongxuan Li, Yuyin Zhou, Zhoujun Li, Zhaoxiang Zhang, Jiaheng Liu, Ge Zhang, Wenhao Huang, and Jason Eshraghian. 2025. A Survey on Latent Reasoning. ArXiv:2507.06203 [cs].