Reasoning Beyond Language: A Comprehensive Survey on Latent Chain-of-Thought Reasoning

Taxonomy

- methodological advances
  - **token-wise strategies**
    - discrete tokens
    - continuous tokens
  - **Internal mechanisms**
    - structural forms
    - representational forms
- analysis and interpretability
- applications

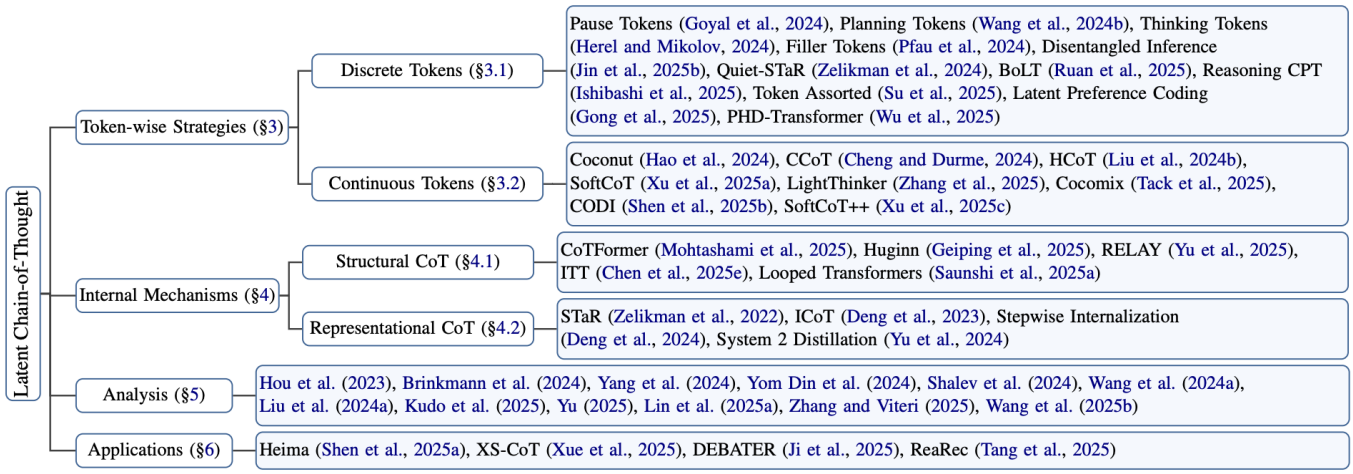
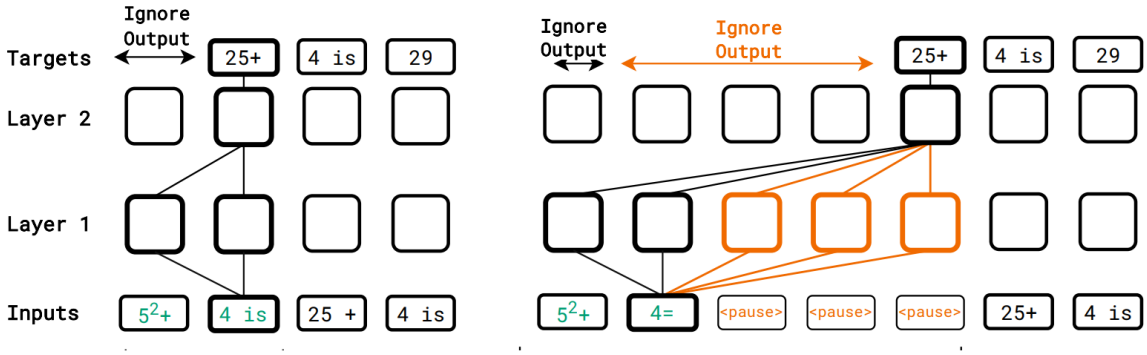


Figure 2: Taxonomy of Latent Chain-of-Thought (CoT) reasoning.

Token-wise strategies

Discrete tokens

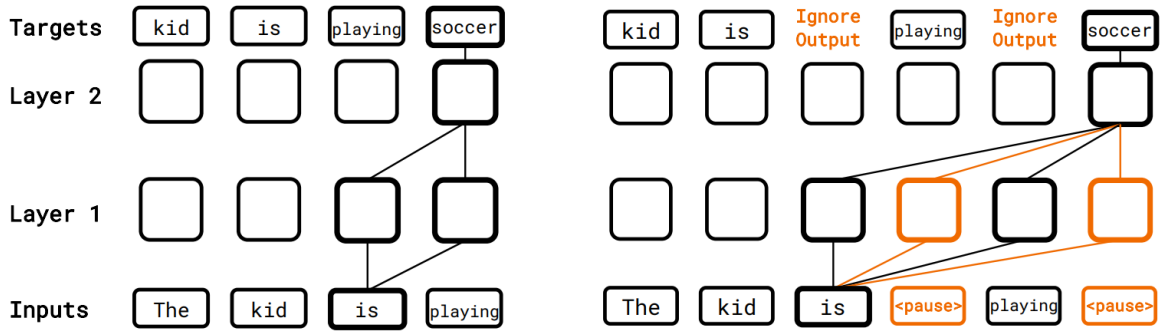
- Pause Tokens (Goyal et al., ICLR 2024)



(a) Standard inference and finetuning (b) Pause-inference and finetuning

Figure 1: **Standard vs. pause-inference (and finetuning)**. We consider a downstream task where, given a prefix, the decoder-only model (bidirectionally) attends to all of the prefix to generate its target answer. The rounded squares denote one Transformer operation (a self-attention and MLP) in a 2-layer Transformer. Any `Ignore Output` denotes that during inference, the corresponding output token is not extracted and thus, not fed back autoregressively; during finetuning, this output is not backpropagated through. The connecting lines denote some (not all) of the "computational pathways" within the model. Specifically, we visualize only those pathways that begin at a specific token in the prefix (here arbitrarily chosen to be "`4 is`") and end at an output token (here arbitrarily chosen to be "`25+`"). All differences between the two settings are highlighted in color. (a) In standard inference (finetuning), the model's output is extracted immediately upon seeing the last prefix token. (b) In pause-inference (and pause-finetuning), this is initiated only after appending a manually specified number of `<pause>` tokens. This introduces new computational pathways (the colored lines) between the prefix token and the output token of interest.

- pretraining: 在文本中随机插入pause token，不算pause token上的loss



(a) Standard pretraining (b) Pause-pretraining

Figure 2: **Standard vs. pause-pretraining**. We consider pretraining based on causal language modeling, where each token is predicted given all preceding tokens in the sequence, using unidirectional self-attention. Here, we visualize the computational pathways beginning from the token "`is`" on the input side of the decoder-only model, to a subsequent token "`soccer`" on the output side. Please see Figure 1 for a guide on how to follow this visualization. (a) In standard pretraining, we compute the model's loss at each output token, and backpropagate through it. (b) In pause-pretraining, we insert multiple copies of `<pause>` tokens at uniformly random locations in the input. However, we do not apply a loss on the model to predict these tokens, as indicated by each corresponding `Ignore Output` flags. This introduces new computational pathways connecting the input token and the output token of interest.

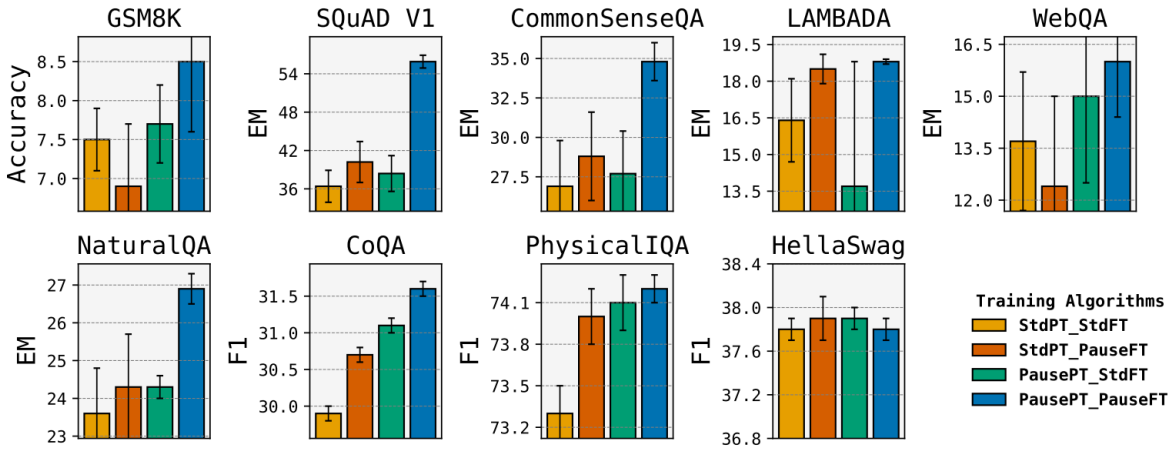


Figure 3: **Downstream performance for a 1B model.** Injecting delays in both stages of training (PausePT_PauseFT) outperforms the standard end-end training StdPT_StdFT on our wide variety of tasks (except HellaSwag). In contrast, introducing delays only in the finetuning stage provides only lukewarm gains, and even hurts in GSM8k.

- ablations: 在finetune的时候加多少pause token是认为给定的，这里对这个数量做了ablation (figure 4b)；figure 4c中表示出训M个pause token，在inference中略微改变pause个数，性能不会差太远

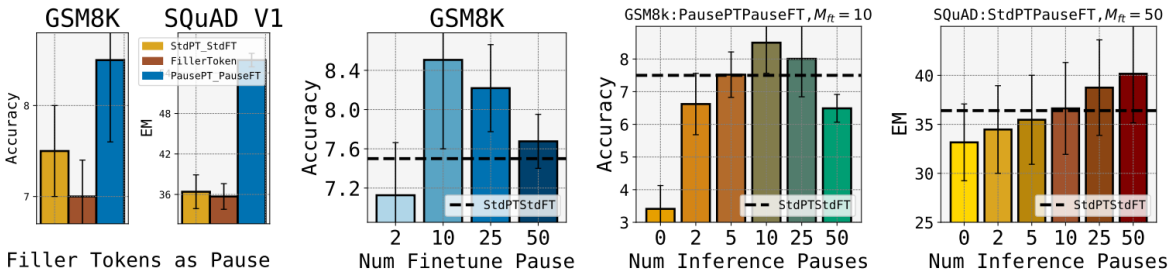


Figure 4: **Key Ablations (§5):** (a) We compare a pause-*trained* model vs. a standard model delayed using a filler periods ('...'). As in Lanham et al. (2023), the filler periods do not give any gains. (b) There exists an optimal number of finetuning <pause> tokens (M_{ft}) for a given downstream dataset beyond which gains diminish. (c) and (d) We test the robustness of pause-trained models to varying number of inference time <pause> tokens (setting M_{inf} not equal to M_{ft}), which exposes the model to a serious test-time distribution shift. Pause-training degrades gracefully to shifts as wide as $M_{\text{inf}} \in [5, 25]$ for $M_{\text{ft}} = 10$ both for (c) PausePT_PauseFT and (d) StdPT_PauseFT.

- Planning Tokens (Wang et al., 2024),
  - the token embedding that can be dynamically captured instead of manually assigned

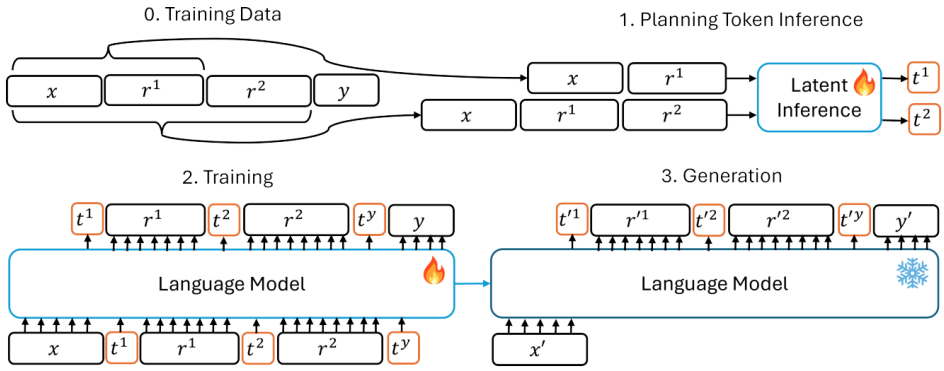


Figure 1: An overview of our method: Given an input question x and its CoT steps $r^1, ..., r^i$, we train a planning token inference model to infer the planning tokens t^i. Subsequently, we fine-tune an LM on data augmented with the inferred planning tokens. Given a new question x', the trained LM can infer planning tokens t'^i, CoT steps r'^i, and final answer y'. Fire and snowflake denote trainable and frozen models, respectively. Note that x, r^i, t^i, y can all include **multiple tokens**, we depict them as single blocks/variables for simplicity.

- Token Assorted (Su et al., 2025)
  - employed vector-quantized VAEs (VQ-VAEs) to condense reasoning steps into discrete latent tokens, reducing computational costs while maintaining performance.

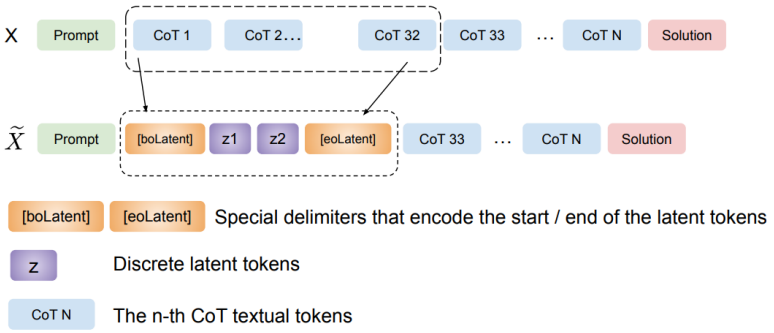


Figure 3.1: An example illustrating our replacement strategy. With chunk size $L = 16$ and compression rate $r = 16$, we encode 32 textual CoT tokens into 2 discrete latent tokens from left to right. The other CoT tokens will remain in their original forms.

Continuous tokens

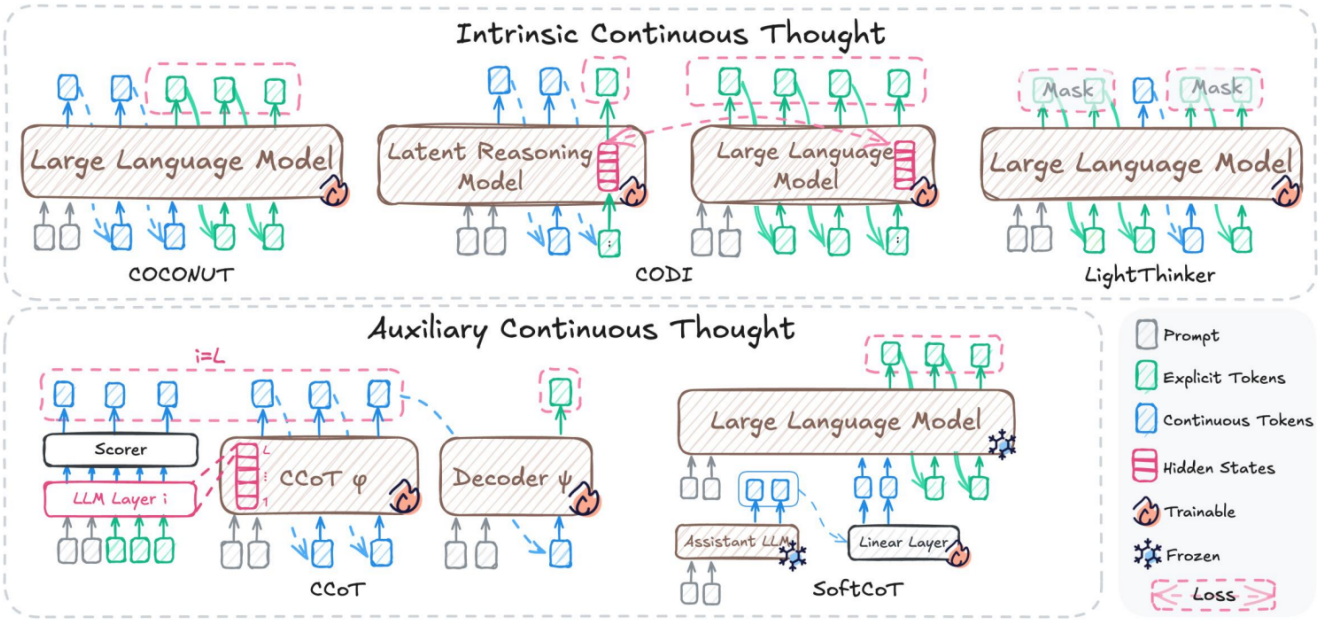


Figure 3: llustration of representative *Continuous Tokens*-based methods. Intrinsic methods generate and consume continuous tokens within a single LLM. Auxiliary methods use external modules to generate continuous tokens.

- COCONUT

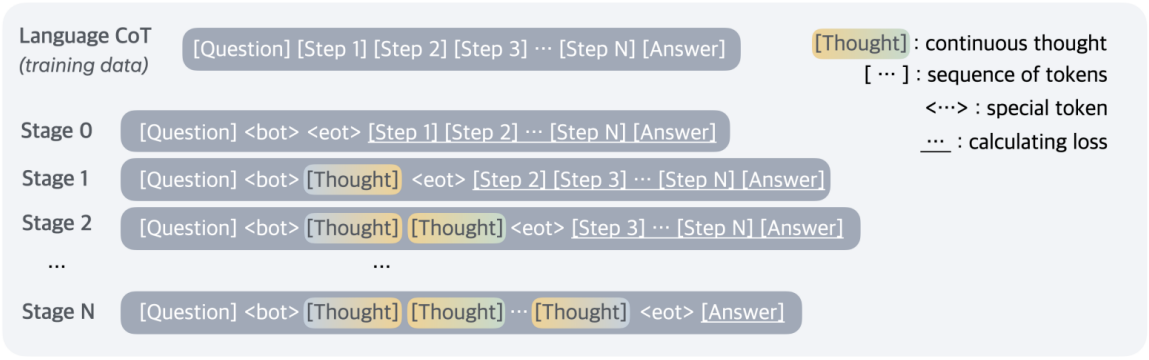


Figure 2 Training procedure of Chain of Continuous Thought (COCONUT). Given training data with language reasoning steps, at each training stage we integrate c additional continuous thoughts ($c = 1$ in this example), and remove one language reasoning step. The cross-entropy loss is then used on the remaining tokens after continuous thoughts.

- CODI

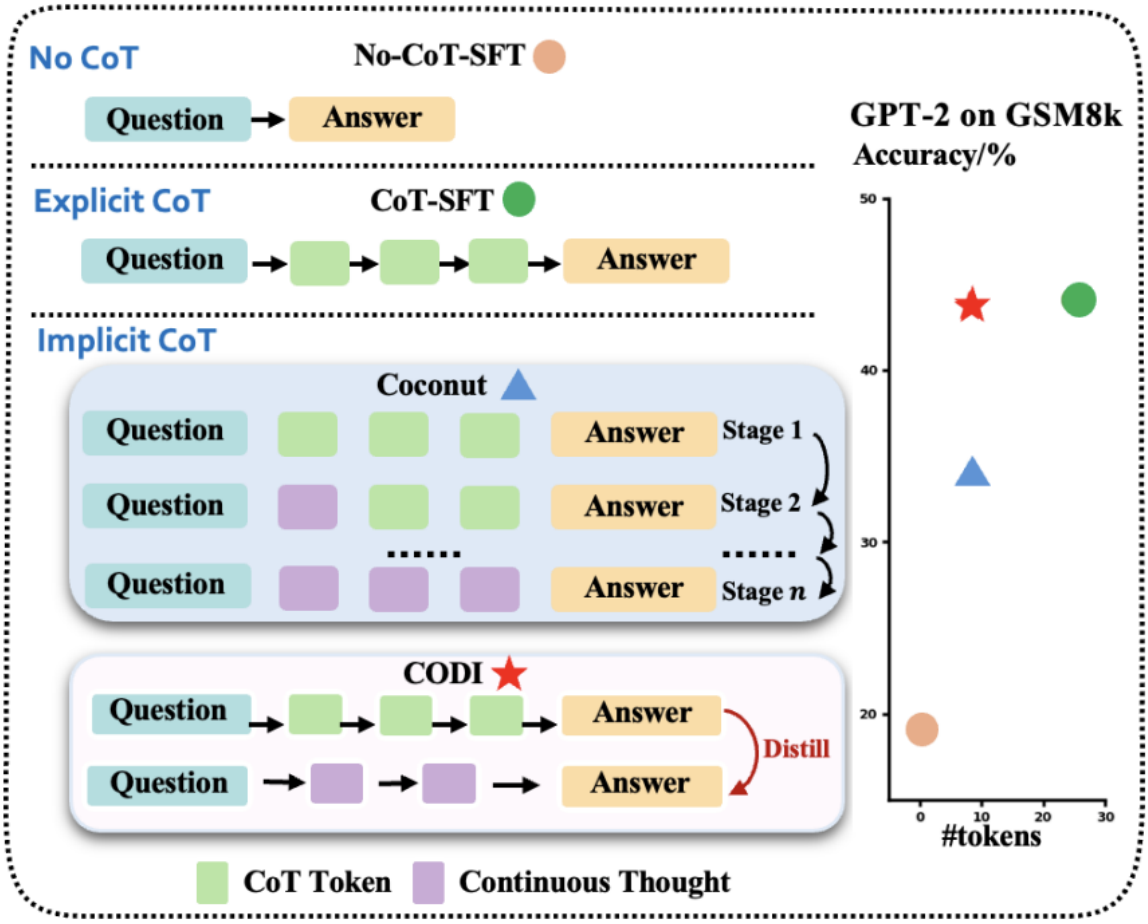
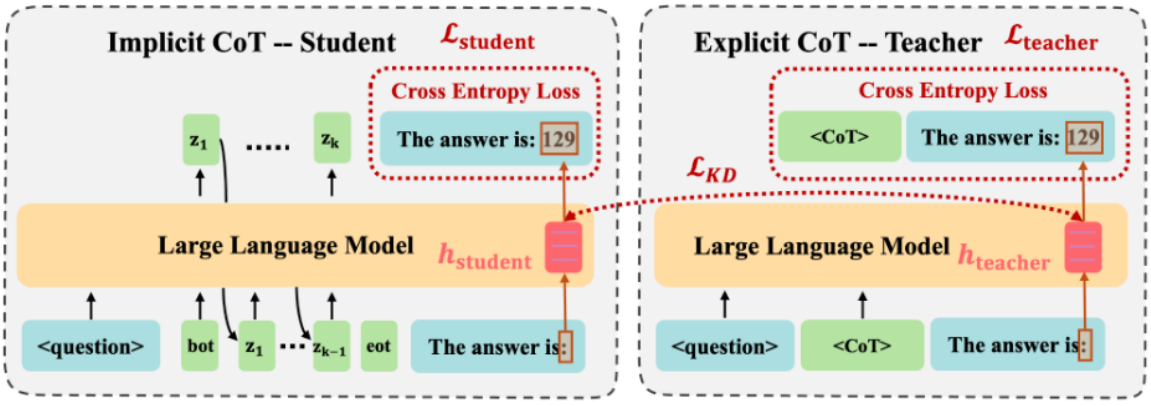


Figure 1: Comparison of reasoning strategies. **No-CoT-SFT**: Train model on (Q,A) pairs via SFT. **CoT-SFT**: Train model on (Q, CoT, A) triples via SFT, i.e., with explicitly annotated CoT reasoning steps. **Coconut**: requires multi-stage training to progressively replace CoT tokens with continuous representations. **CODI**: achieves this in a single stage by compressing CoT tokens into continuous space via self-distillation.



A Survey on Latent Reasoning

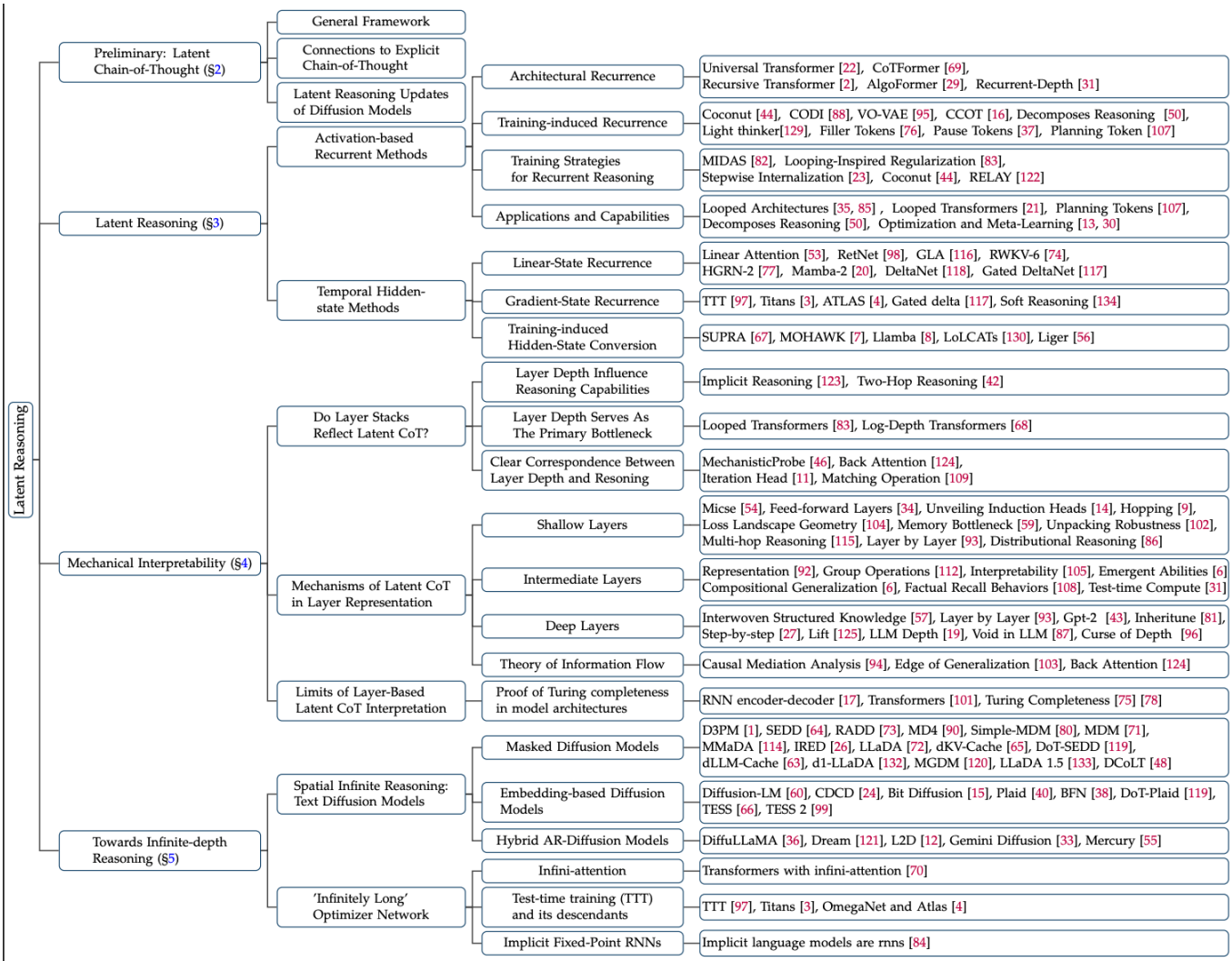


Figure 2. Taxonomy of Latent Reasoning.

Latent Reasoning

作者总体分为vertical recurrence: activation-based methods和horizontal recurrence: hidden state-based methods

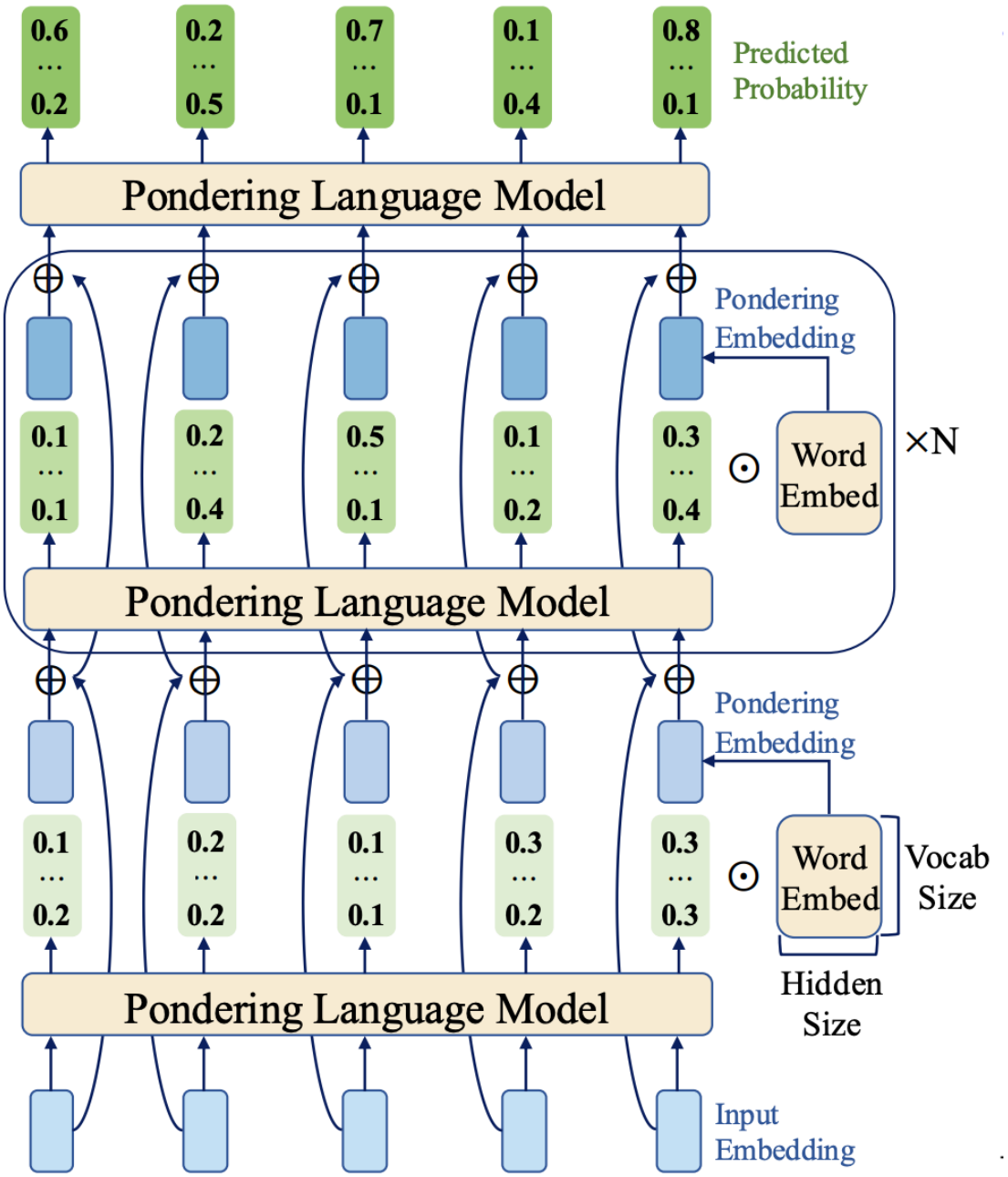
1. Activation-based Recurrent Method

1.1 Loop/Universal Transformer Recurrence

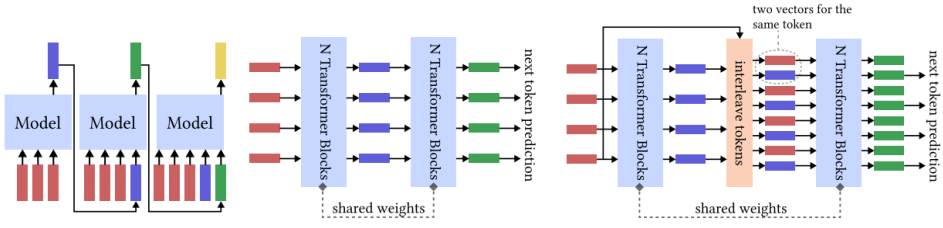
Loop/Universal transformer recurrence：通过architecture上的改动，使得一次前向可以包含多层的recurrence。

- **initial work**
  - universal transformer (2018)
    - 有一个机制控制循环几次
    - **Adaptive Computation Time** to control compute step for each token independently ⟹ 每个position的ACT是独立的，如果某个position a在t时刻被停止了，h_a^t 会被一直复制到最后一个position停止
    - The key innovation lies in treating network depth not as a static hyperparameter but as a dynamic computational resource that can be allocated based on task complexity
  - Pondering LM (2025.6)

- 有点像Jacobi decoding
- 一次前向，用输出的probability对token embedding加权，作为下一轮输入，再迭代几轮



- **some evolution trend**
  - **逐渐模块化**：The Rise of Pre/Loop/Coda Structure
    - Universal Transformer and CoTFormer 没有特别的stage的区分，都是统一的recurrent的架构
      - CoTFormer:把之前的token append直接append到输出token前（2 vectors for a same token）



(a) CoT (b) Block Universal Transformer (c) CoTFormer

    - 之后逐渐区分出Pre/Loop/Coda的structure

    - **每个模型中per-iteration的input会有区别：**
        - 给上一层的output
        - 给上一层的output+depth embeddng（Universal Transformer）
        - 给上一层的output和第一层的embedding
    - **depth embedding逐渐消退**
    - **dynamic stopping机制逐渐简化**
        - 从一开始universal transformer的ACT（算cumulative probability）到CoTFormer的MoR router，之后基本类似于不动点迭代

1.2 Activation with Explicit Hidden-State Feedback

While loop-based architectures refine token representations by rerunning the same set of layers, a distinct family of models *feeds hidden states back into the input stream* between iterations.

In these systems the *hidden activations themselves become new sequence elements*, so each recurrent step simultaneously extends the effective depth and exposes internal computation to subsequent attention.

- COCONUT

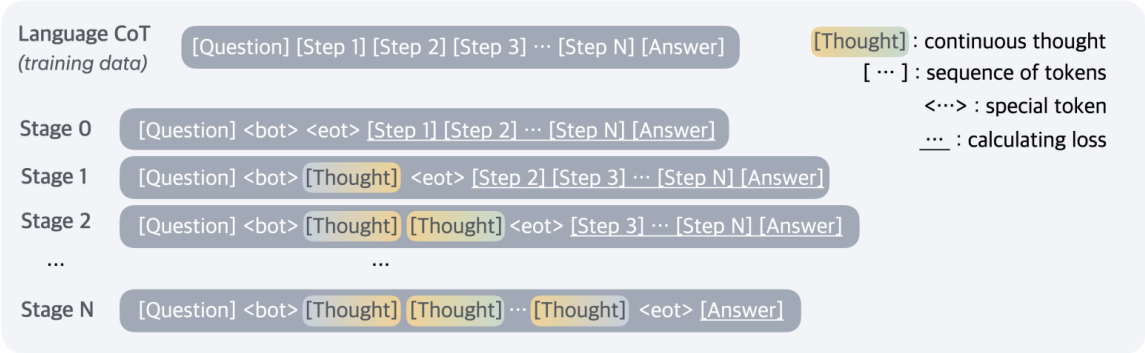


Figure 2 Training procedure of Chain of Continuous Thought (COCONUT). Given training data with language reasoning steps, at each training stage we integrate c additional continuous thoughts ($c = 1$ in this example), and remove one language reasoning step. The cross-entropy loss is then used on the remaining tokens after continuous thoughts.

    -
- CoTFormer

和上面的Universal Transformer的不同之处：

- 会重新把hidden vector当作token输进去（sequence随着每次过前向会变长），bridging vertical recurrence and horizontal memory
- no architectural expansion—the model reuses the same layers so parameter count stays constant while depth grows dynamically
- latent reasoning remains internal, thereby avoiding the latency of producing explicit CoT tokens

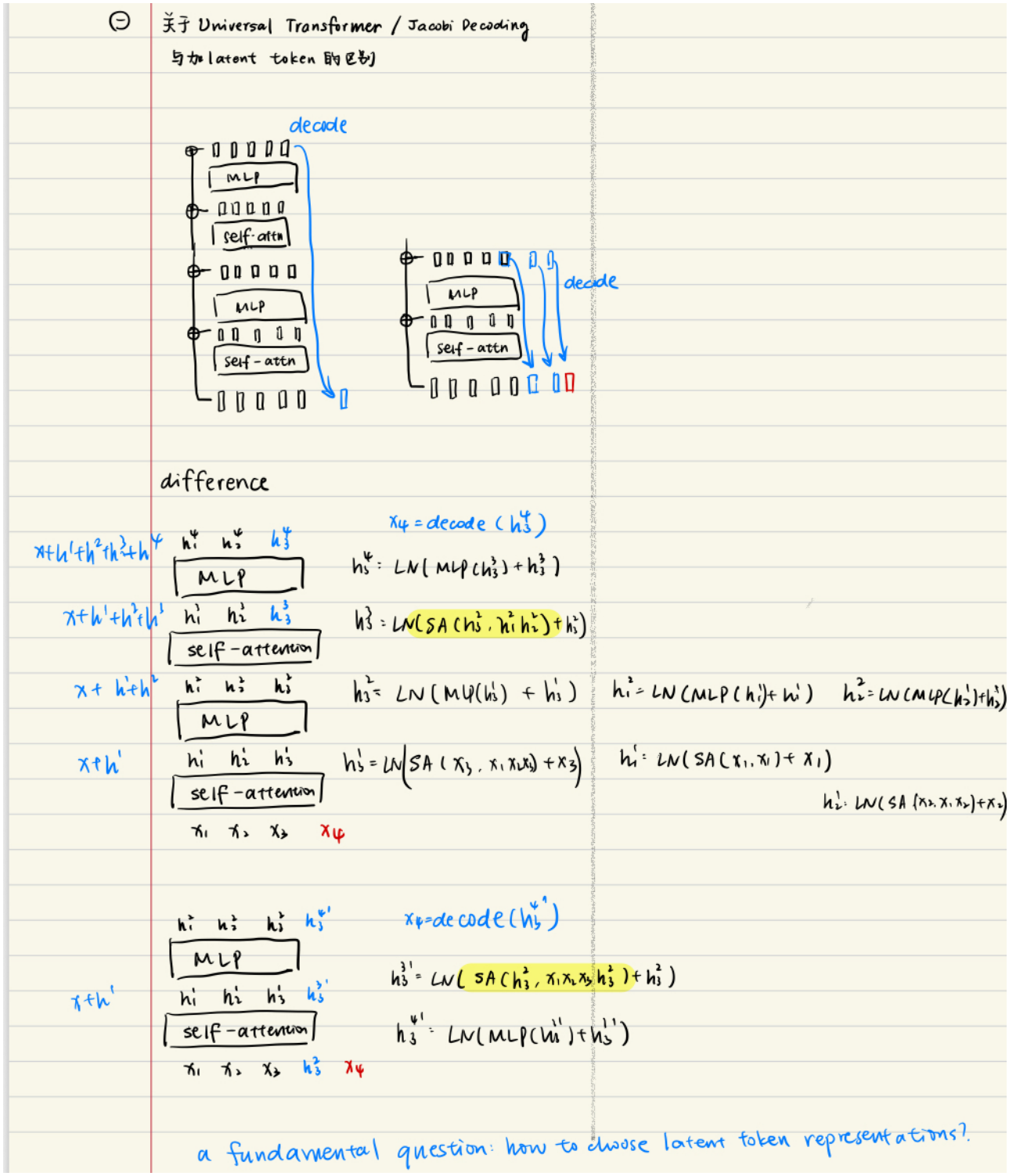
1.3 Training-induced Recurrence

不用做architectural改进

- core principal: create implicit loops in the computation graph
    1. continuous recurrence: 把activation重新喂给模型，by feeding activations back into the model
    2. compressed states: compressing multi-step reasoning into iteratively-processed representations

3. expanded iterations: extending the effective computation depth through strategic token insertion

⭐ the differences between architectural changes like universal transformer and adding latent tokens:



- 这么看来，从表达能力上来看，上面这种universal transformer的要比加latent token的要好，因为h_1^2, h_2^2包含了x_1, x_2的信息（through residual stream）。但效率上应该要低一些?

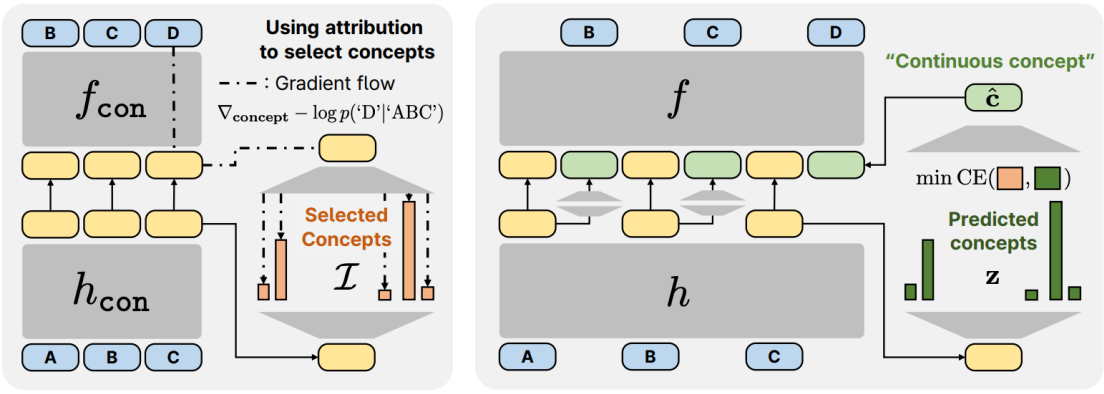
1. Continuous Activation Recurrence
    1. Coconut: implemented entirely through training, instead of modifying the model architecture.
        - trainig process

    - at the k-stage, the first k reasoning steps in the CoT are replaced with $k \times c$ continuous thoughts, where c is a hyperparameter controlling the number of latent thoughts replacing a single language reasoning step
    - does **not** encourage the continuous thought to *compress the removed language thought*, but rather to *facilitate the prediction of future reasoning*

- inference process

    1. train a binary classifier on latent thoughts to enable the model to autonomously decide when to terminate the latent reasoning
    2. always pad the latent thoughts to a constant length
    - ⭐ 这里直觉上来说，是否可以用不动点迭代思想，用latent thought是否收敛来判断

2. CoCoMix: Coconut 续作

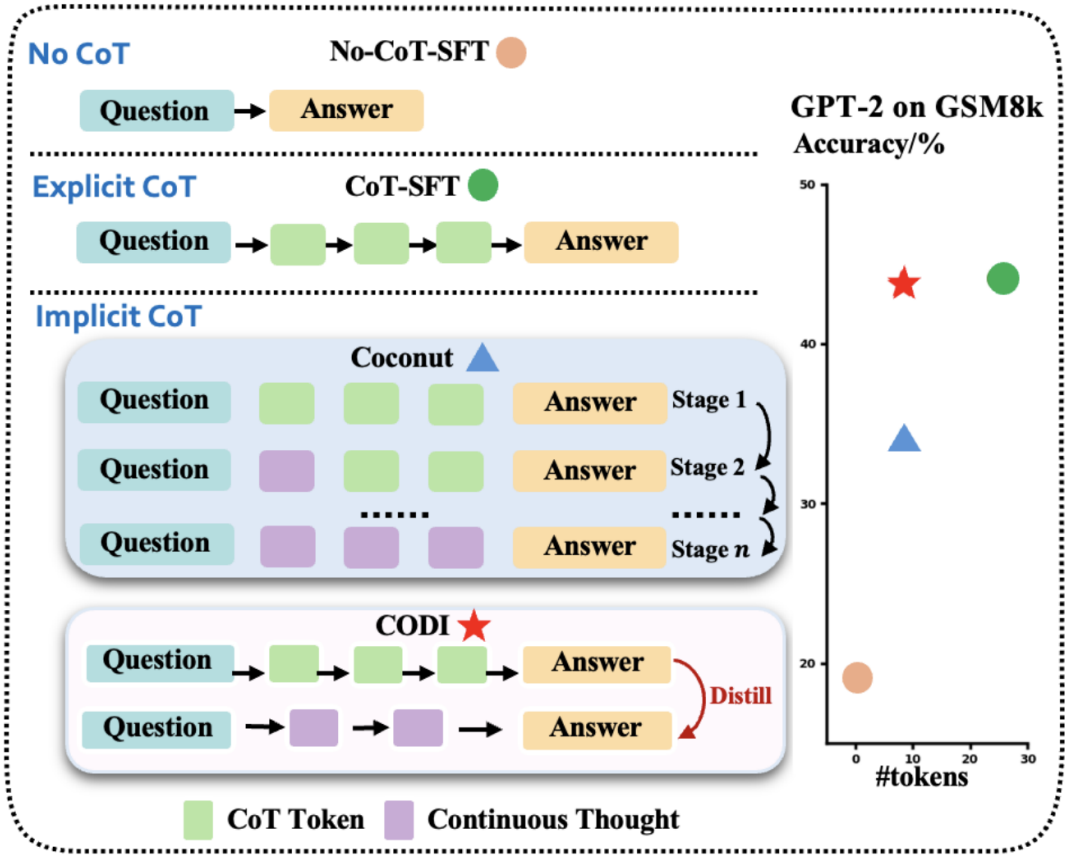


Extracting concepts from a pretrained SAE model's hidden state

Learning to predict concepts & Mixing/Interleaving continuous concepts into the hidden state

    - 用一个预训练的SAE来抽每一次前向中的influential concepts based on attribution scores
    - the model is then trained to predict these selected concepts from its hidden state using a cross entropy loss
    - model 能够 predict multiple concepts后，可以把这些压缩到一个single continuous concept然后mix or insert到hidden states中去
    - can do steering with the concepts

3. CODI: self distillation
    - By aligning the hidden activation before the final answer between teacher (with full CoT) and student (with compressed reasoning) paths, CODI effectively learns a fixed-point iteration in activation space.
    - more stable than coconut's curriculum learning



4. CCOT: variable lengths

Algorithm 1 Chain of Thought inference

Require: Query w, parameters θ

$\bar{w} \leftarrow \text{EMBED}_\theta(w)$ ▷ *embed query*
$\hat{w} \leftarrow \text{ATTN}_\theta(\bar{w})$ ▷ *compute hidden states*
$z \leftarrow [\langle COT \rangle]$
while $z_{-1} \neq \langle ANS \rangle$ **do**
  $[\hat{w}; \hat{z}] \leftarrow \text{ATTN}_\theta([\bar{w}; \text{EMBED}_\theta(z)])$
  $x \sim \text{HEAD}_\theta(\hat{z}^L_{-1})$
  $z \leftarrow [z; x]$
end while
$a \leftarrow [\langle ANS \rangle]$
while $a_{-1} \neq \langle EOS \rangle$ **do**
  $[\hat{w}; \hat{z}; \hat{a}] \leftarrow \text{ATTN}_\theta([\bar{w}_{1:n}; \text{EMBED}_\theta([z; a])])$
  $x \sim \text{HEAD}_\theta(\hat{a}^L_{-1})$ ▷ *sample answer token*
  $a \leftarrow [a; x]$
end while
return a

Algorithm 2 CCOT inference

Require: Query w, parameters θ, φ, ψ, autoregressive layer l

$\bar{w} \leftarrow \text{EMBED}_\theta(w)$ ▷ *embed query*
$\hat{w} \leftarrow \text{ATTN}_\theta(\bar{w})$ ▷ *compute hidden states*
$z \leftarrow [\hat{w}^l_{-1}]$
while $\text{END}_\psi(\hat{z}^L)$ is $False$ **do**
  $[\hat{w}; \hat{z}] \leftarrow \text{ATTN}_{\theta,\varphi}([\bar{w}; z])$ ▷ *gen. cont. token*
  $z \leftarrow [z; \hat{z}^l_{-1}]$ ▷ *append cont. token*
end while
$a \leftarrow [\langle ANS \rangle]$
while $a_{-1} \neq \langle EOS \rangle$ **do**
  $[\hat{w}; \hat{z}; \hat{a}] \leftarrow \text{ATTN}_{\theta,\varphi,\psi}([\bar{w}_{1:n}; z; \text{EMBED}_\theta(a)])$
  $x \sim \text{HEAD}_\psi(\hat{a}^L_{-1})$ ▷ *sample answer token*
  $a \leftarrow [a : x]$
end while
return a

Figure 2. Two algorithms for inference with contemplation tokens. Algorithm 1 describes the usual chain of thought decoding while Algorithm 2 describes our method, obtained by replacing the yellow text with the cyan text. While CoT decoding decodes contemplation tokens by passing the LM head across the final hidden state, we use the hidden state at the lth layer directly.

5. PCCOT: Jacobi-iteration allowing parallel continuous thoughts
6. System-1.5 Reasoning
   - 不止省token，也省层深的latent reasoning

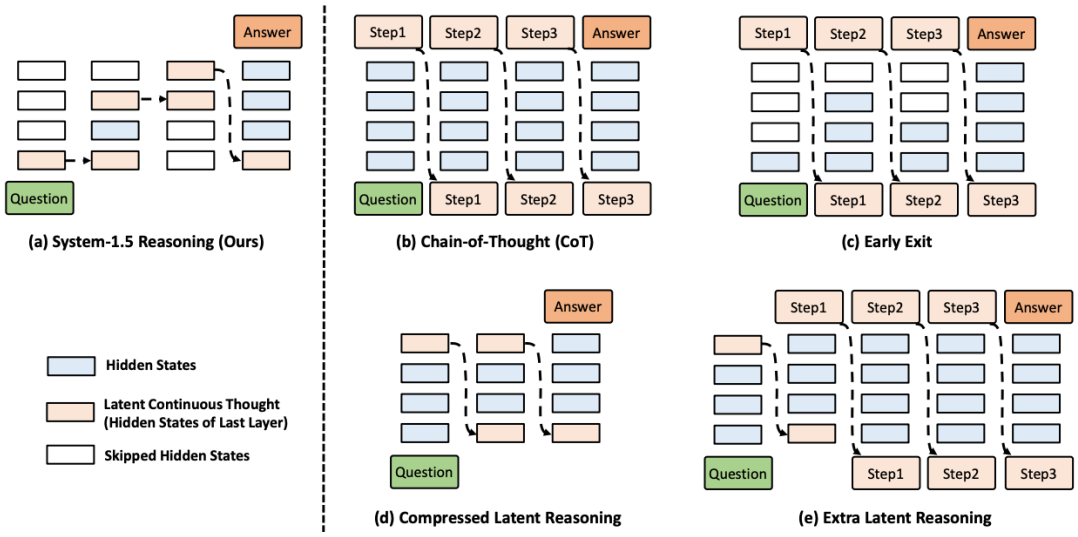


Figure 1: Comparison of (a) our proposed `System-1.5 Reasoning`, (b) chain-of-thought (CoT) reasoning, (c) early-exit (Elbayad et al., 2019; Elhoushi et al., 2024), (d) compressed latent-space reasoning (*e.g.*, Coconut (Hao et al., 2024) and CCoT (Cheng & Van Durme, 2024)), and (e) extra latent reasoning approaches that delay output (*e.g.*, *pause* token (Goyal et al., 2023) and *filler* token (Pfau et al., 2024)). `System-1.5 Reasoning` enables flexible latent reasoning along both the horizontal and vertical dimensions through dynamic shortcuts.

2. Compressed State Recurrence: compress reasoning steps into discrete / semi-discrete tokens
    1. Token Assorted

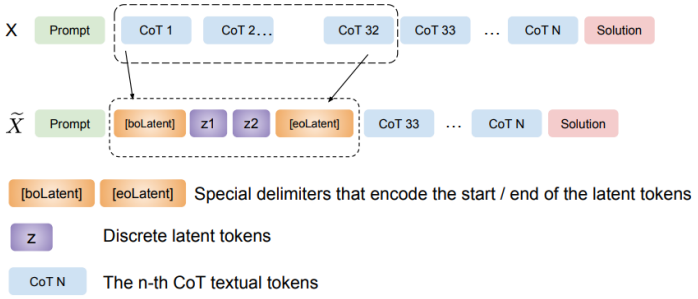


Figure 3.1: An example illustrating our replacement strategy. With chunk size $L = 16$ and compression rate $r = 16$, we encode 32 textual CoT tokens into 2 discrete latent tokens from left to right. The other CoT tokens will remain in their original forms.

    2. gist tokens
3. Iteration Expansion through Strategic Tokens
    1. Filler Token: Pfau et al. demonstrate that even meaningless filler tokens (e.g., ".....")
    2. Pause Token: Goyal et al. refine this with learnable `<pause>` tokens that explicitly signal computation steps
    3. More sophisticated approaches inject structured tokens that organize the recurrence pattern, 比如加入类似：`<memory>`, `<reason>`

1.4 Training Strategies for Recurrent Reasoning

- for architectural recurrence
    - MIDAS proposes a progressive stacking framework to address training stability in loop-based models. It defines a replication operator $\mathrm{M}(\mathrm{f}, \mathrm{b})$ that duplicates the middle layers of a base model f by a factor b, enabling gradual depth expansion.
- for training-induced recurrence
    - Stepwise Internalization pioneered curriculum-based compression of reasoning traces. This technique gradually removes CoT tokens during finetuning, allowing models to internalize

reasoning patterns into their parameters. ⟹ curriculum learning adopted by Coconut

---

2. Temporal Hidden-state Methods

discuss models like RNN, mamba... not my main focus, maybe fill here someday.

Mechanistic Interpretability

1. Do Layer Stacks Reflect Latent CoT?

conclusions:

- generally, layer depth serves as the primary bottleneck for latent reasoning capacity
- At a micro level, studies commonly reveal a clear correspondence between specific layers and tasks within CoT reasoning

2. Mechanisms of Latent CoT in Layer Representation

- Shallow Layers: Basic representational processor of Latent CoT
- Intermediate Layers: Core of Latent CoT
  - Intermediate layers contain specific, identifiable computational sub-circuits specialized for distinct reasoning sub-tasks
  - Intermediate layers exhibit unique characteristics in representation
  - Intermediate layers have a causal influence on final reasoning outcomes
- Deep Layers: Output Refinement and Decision-making of Latent CoT

3. Turing Completeness of Layer-Based Latent CoT

- Pérez et al. formally proved for the first time that the **Transformer architecture is Turing complete**, possessing the universal capability to execute any computable function. However, the validity of this proof relies on three crucial theoretical assumptions:
  - Arbitrary Precision
  - Positional Encodings
  - Hard-Max Attention.
- Further, Li and Wang proved for the first time that Turing completeness can be achieved under constant numerical precision
- A unifying view of latent CoT and standard CoT
  -

	Operation	Storage	Resource Constraint	Optimization Objective
Standard CoT	Full Model Forward Pass	Explicit Tokens In the Sequence	Context Window	End-to-end Task
Layer-based Latent CoT	Single Layer Forward Pass	Hidden States	Layer Nums	Next Token Prediction

---

Towards Infinite-depth Reasoning

to be continued