Hierarchical Reasoning Model

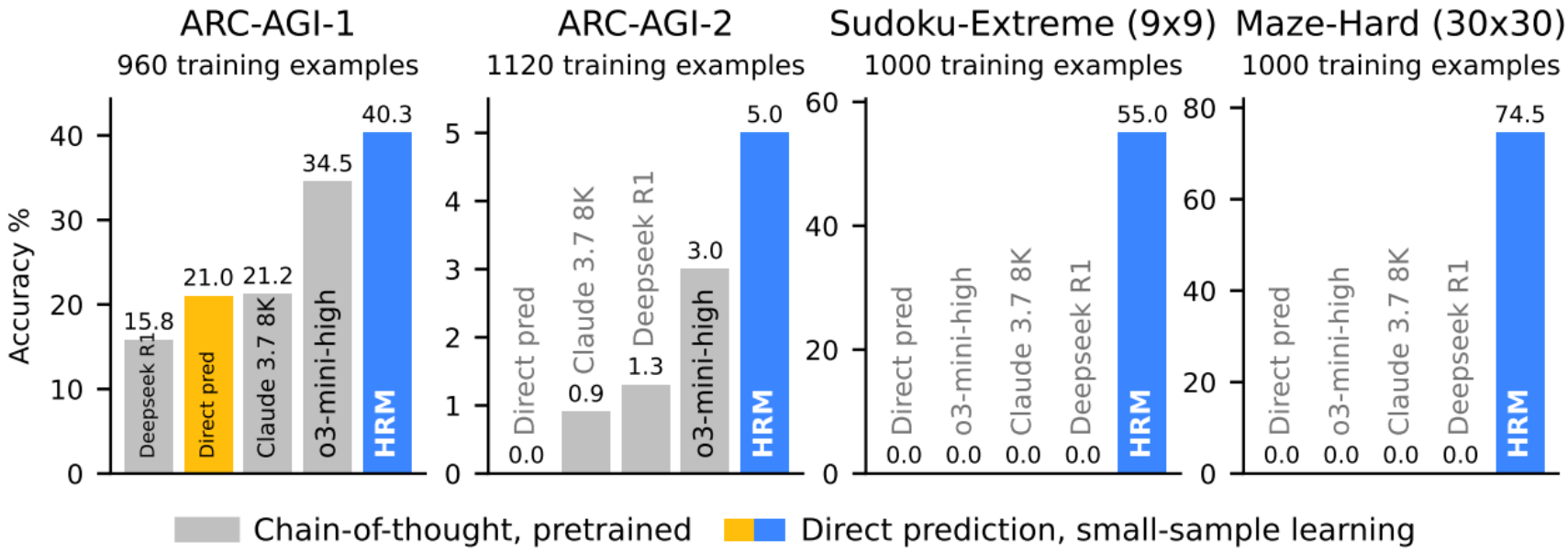
Guan Wang[1,†], Jin Li[1], Yuhao Sun[1], Xing Chen[1], Changling Liu[1],
Yue Wu[1], Meng Lu[1,†], Sen Song[2,†], Yasin Abbasi Yadkori[1,†]

[1]Sapient Intelligence, Singapore

Presented by:

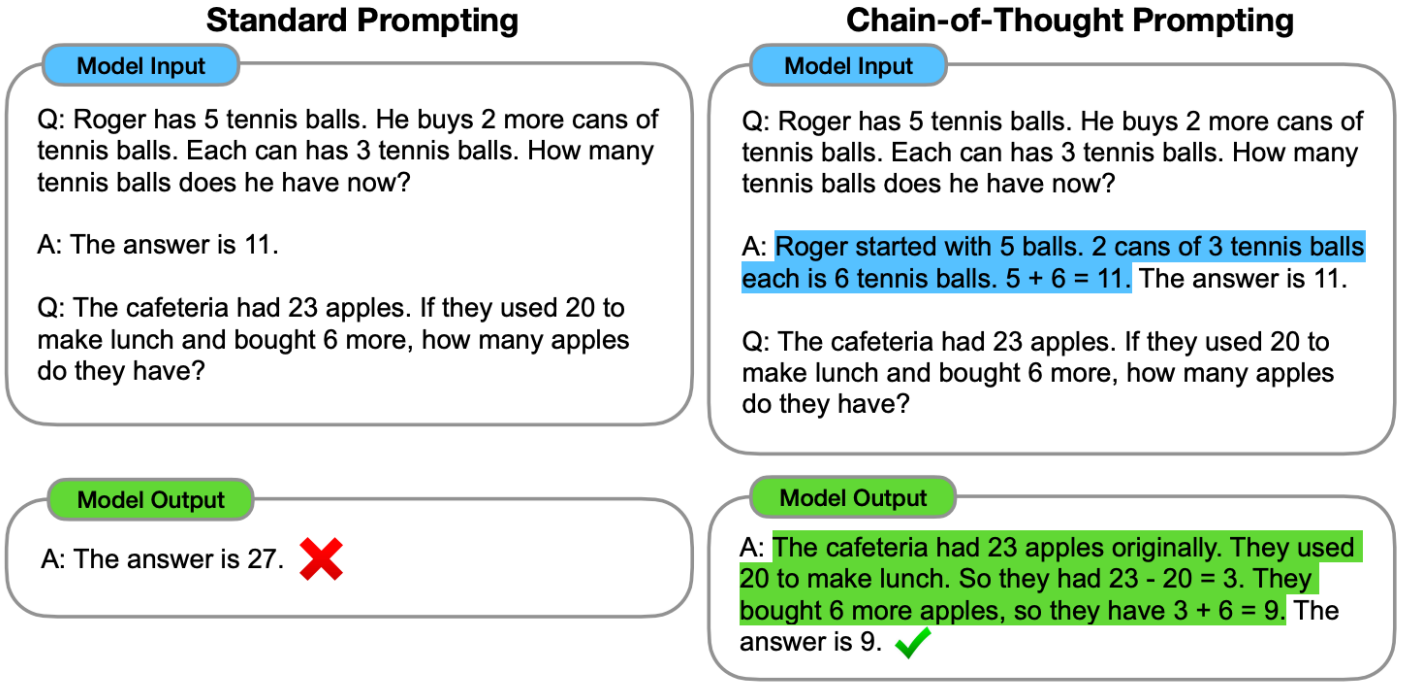
John Tan Chong Min

Impressive results (27mil param model surpassing models with billions of params)



Traditionally: Chain of Thought (CoT)

- Prompting with reasoning steps in context can help generate better output



Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. Jason et al. 2023.

Is language sufficient for reasoning?

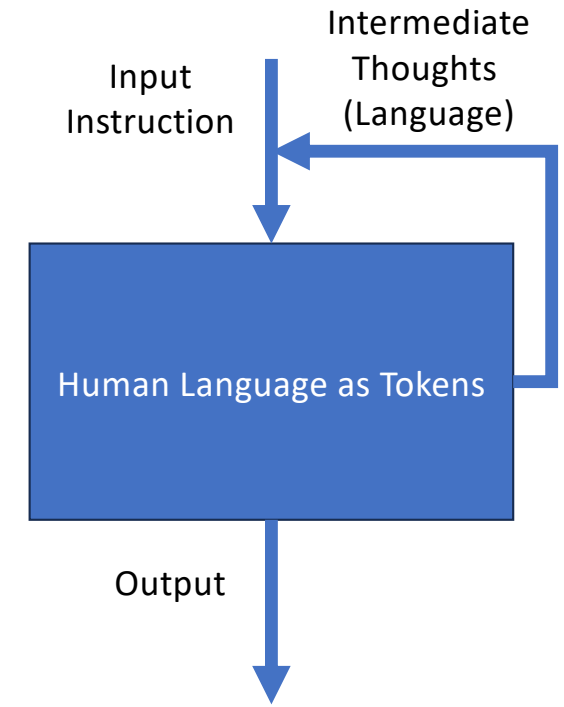
“A system trained on language alone will never approximate human intelligence, even if trained from now until the heat death of the universe.”

What else do we need?

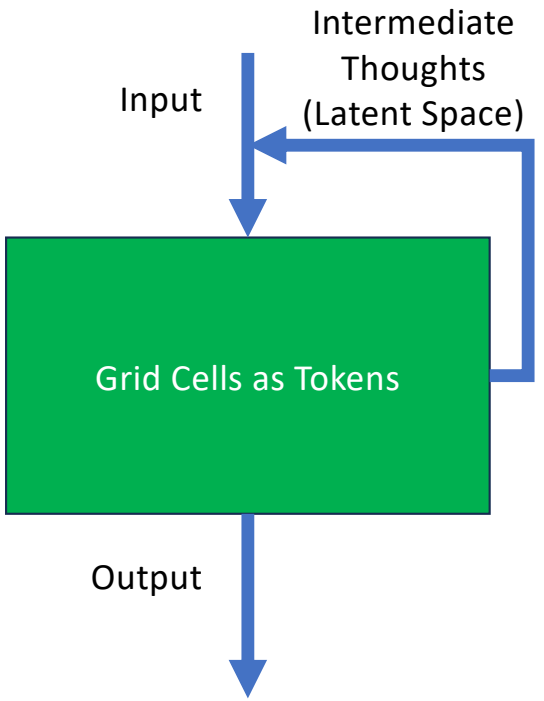
- World Models
- Visual Understanding
- Embodied Information (e.g. sensorimotor)
- Social/Cultural Context
- Other information that are not language-based

AI And The Limits Of Language, Jacob Browning and Yann LeCun. 2022.
https://www.noemamag.com/ai-and-the-limits-of-language/

New paradigm for thinking?



Traditional Text-based Transformers
Thinking in language



Hierarchical Reasoning Model
Thinking in latent space

What if the model can reason internally **without human language**?

Feature	Chain-of-Thought (CoT)	Latent Reasoning
Reasoning Mechanism	**Externalizes reasoning** via generated text	Computes within **internal hidden state**
Domain of Thought	Relies on brittle, human-defined decompositions in human language	Latent space. **No need to translate back to human language**
Data Requirements	Often needs **significant training data to** denote steps to take and sequence of thoughts to follow	**Minimal data** requirements as sequence of steps is done internally rather than externally
Response Times	**Slow due to extensive token generation**	**Faster response times**

Scaling depth is important;
Traditional Transformers do not scale depth well

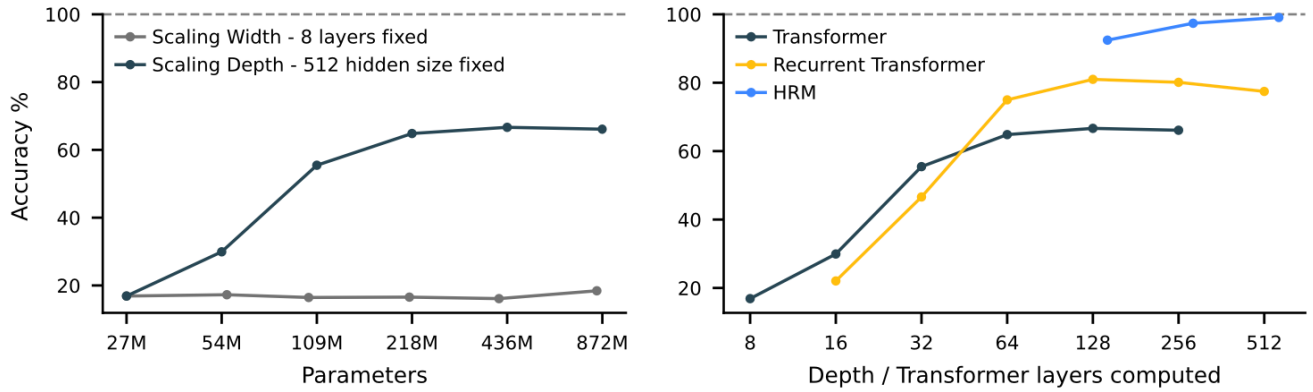


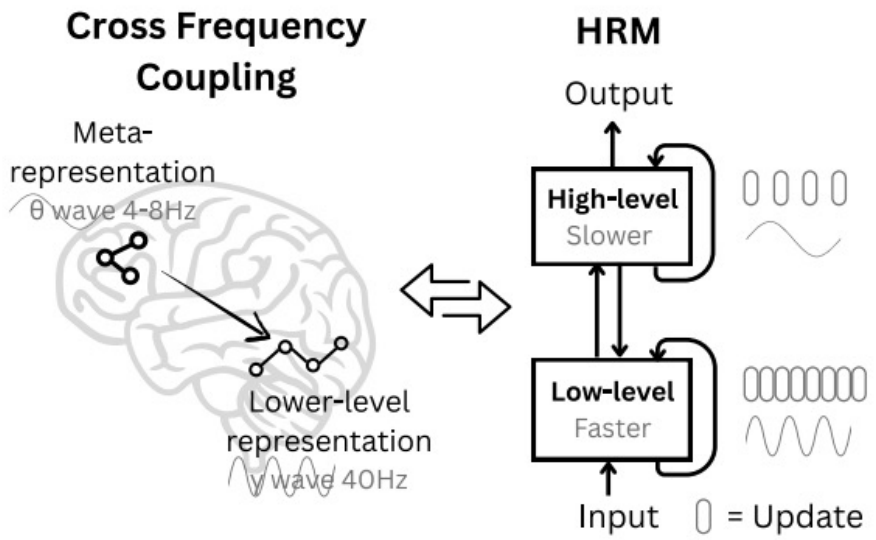
Figure 2: **The necessity of depth for complex reasoning. Left:** On *Sudoku-Extreme Full*, which require extensive tree-search and backtracking, increasing a Transformer's width yields no performance gain, while increasing depth is critical. **Right:** Standard architectures saturates, failing to benefit from increased depth. HRM overcomes this fundamental limitation, effectively using its computational depth to achieve near-perfect accuracy.

Neuroscience Inspiration

- **Hierarchical processing:** The brain processes information across a hierarchy of cortical areas. Higher-level areas integrate information over longer timescales and form abstract representations, while lower-level areas handle more immediate, detailed sensory and motor processing

- **Temporal Separation:** These hierarchical levels in the brain operate at distinct intrinsic timescales, reflected in neural rhythms (e.g., slow theta waves, 4–8 Hz and fast gamma waves, 30–100Hz). This separation allows for stable, high-level guidance of rapid, low-level computation

- **Recurrent Connectivity:** The brain features extensive recurrent connections. These feedback loops enable iterative refinement, yielding more accurate and context-sensitive representations at the cost of additional processing time

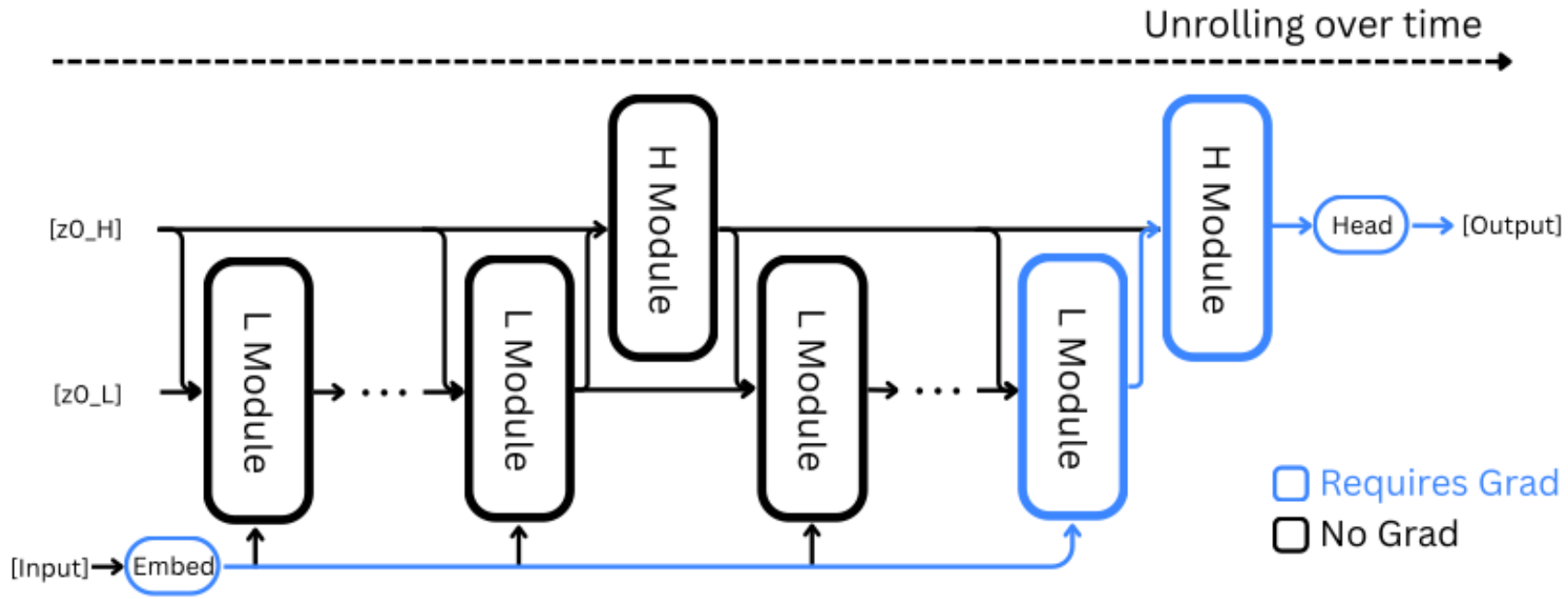


Hierarchical Reasoning Model

- HRM is designed to significantly increase the effective computational depth
- It features two coupled recurrent modules:
  - High-level (H) module for abstract deliberate reasoning
  - Low-level (L) module for fast, detailed computations
- The slow-updating H-module advances only after the fast-updating L-module has completed multiple computational steps and reached a local equilibrium, at which point the L-module is reset to begin a new computational phase

Truncated Backpropagation Through Time

- Instead of gradient going through all H and L modules of previous timesteps, just go through final state
- Helps avoid vanishing/exploding gradients with shorter backpropagation path



Implementation Code

```
def hrm(z, x, N=2, T=2):
    x = input_embedding(x)
    zH, zL = z

    with torch.no_grad():
        for _i in range(N * T - 1):
            zL = L_net(zL, zH, x)
            if (_i + 1) % T == 0:
                zH = H_net(zH, zL)

    # 1-step grad
    zL = L_net(zL, zH, x)
    zH = H_net(zH, zL)
    return (zH, zL), output_head(zH)
```

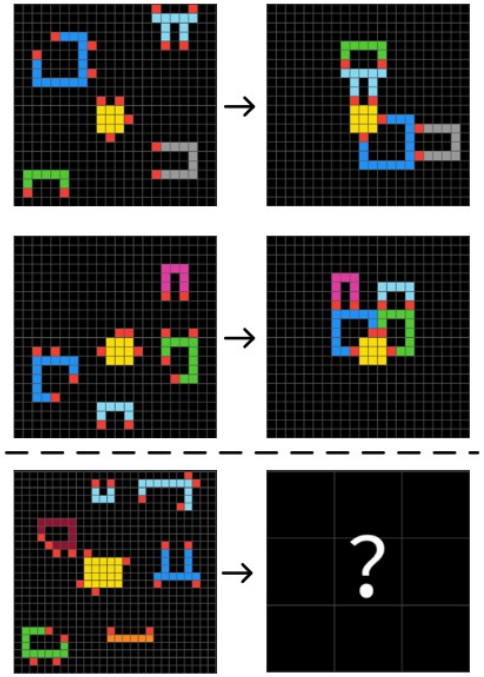
Only input_embedding and last two steps through L_net and H_net and output_head has gradient computations (yellow boxes)

```
# Deep Supervision
for x, y_true in train_dataloader:
    z = z_init
    for step in range(N_supervision):
        z, y_hat = hrm(z, x)

        loss = softmax_cross_entropy(y_hat, y_true)
        z = z.detach()

        loss.backward()
        opt.step()
        opt.zero_grad()
```

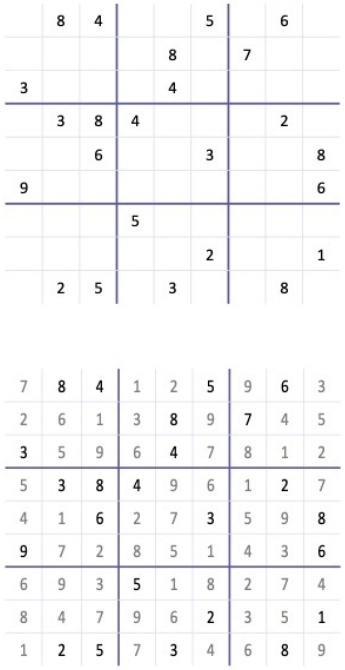
Trained using typical next-token-prediction cross entropy loss as Transformers
More loops in step represents more thinking

Experimental Tasks

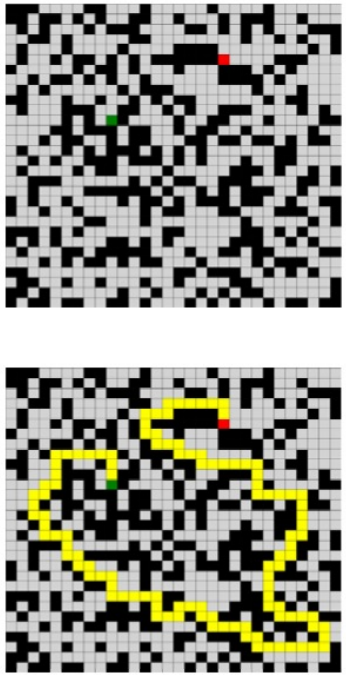
- Inputs and outputs are flattened from 2D grids to 1D tokens



(a) ARC-AGI



(b) Sudoku-Hard



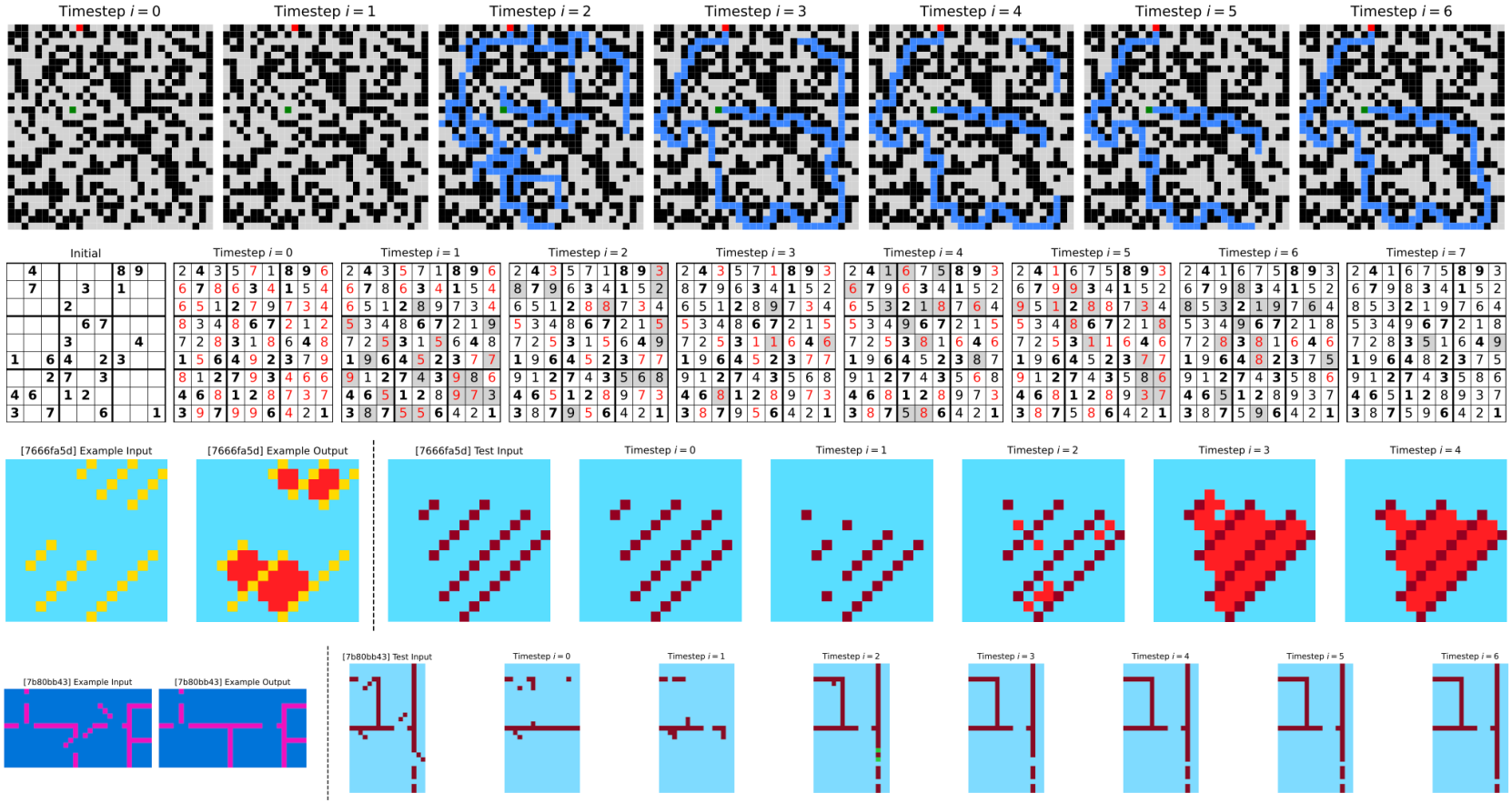
(c) Maze navigation

HRM is remarkable, but how much of its performance is due to data augmentation?

For ARC-AGI challenge, we start with all input-output example pairs in the training and the evaluation sets. The dataset is augmented by applying translations, rotations, flips, and color permutations to the puzzles. Each task example is prepended with a learnable special token that represents the puzzle it belongs to. At test time, we proceed as follows for each test input in the evaluation set: (1) Generate and solve 1000 augmented variants and, for each, apply the inverse-augmentation transform to obtain a prediction. (2) Choose the two most popular predictions as the final outputs.[3] All results are reported on the evaluation set.

We augment Sudoku puzzles by applying band and digit permutations, while data augmentation is disabled for Maze tasks. Both tasks undergo only a single inference pass.

Visualising Intermediate Thinking Steps



Math

HRM has 4 trainable networks

The HRM model consists of four learnable components: an input network $f_I(\cdot;\theta_I)$, a low-level recurrent module $f_L(\cdot;\theta_L)$, a high-level recurrent module $f_H(\cdot;\theta_H)$, and an output network $f_O(\cdot;\theta_O)$. The model's dynamics unfold over N high-level cycles of T low-level timesteps each[2]. We index the total timesteps of one forward pass by $i = 1, \ldots, N \times T$. The modules f_L and f_H each keep a hidden state—z_L^i for f_L and z_H^i for f_H—which are initialized with the vectors z_L^0 and z_H^0, respectively.

The HRM maps an input vector x to an output prediction vector $\hat{y}$ as follows. First, the input x is projected into a working representation $\tilde{x}$ by the input network:

$$\tilde{x} = f_I(x;\theta_I) \ .$$

TLDR:

We only have 4 major modules
- Input Mapping
- Low-level Recurrent
- High-level Recurrent
- Output Mapping

Fast and slow updates

At each timestep i, the L-module updates its state conditioned on its own previous state, the H-module's current state (which remains fixed throughout the cycle), and the input representation. The H-module only updates once per cycle (i.e., every T timesteps) using the L-module's final state at the end of that cycle:

$$z_L^i = f_L\left(z_L^{i-1}, z_H^{i-1}, \tilde{x}; \theta_L\right) ,$$

$$z_H^i = \begin{cases} f_H\left(z_H^{i-1}, z_L^{i-1}; \theta_H\right) & \text{if } i \equiv 0 \pmod{T} , \\ z_H^{i-1} & \text{otherwise} . \end{cases}$$

Finally, after N full cycles, a prediction $\hat{y}$ is extracted from the hidden state of the H-module:

$$\hat{y} = f_O(z_H^{NT}; \theta_O) .$$

TLDR:

Update the high-level recurrent module every T timesteps
Update low-level recurrent module every timestep

Deep Supervision (Learn from intermediate thinking states)

Deep supervision Inspired by the principle that periodic neural oscillations regulate when learning occurs in the brain[38], we incorporate a deep supervision mechanism into HRM, as detailed next.

Given a data sample (x, y), we run multiple forward passes of the HRM model, each of which we refer to as a *segment*. Let M denote the total number of segments executed before termination. For each segment $m \in \{1, \ldots, M\}$, let $z^m = (z_H^{mNT}, z_L^{mNT})$ represent the hidden state at the conclusion of segment m, encompassing both high-level and low-level state components.

At each segment m, we apply a deep supervision step as follows:

1. Given the state z^{m-1} from the previous segment, compute the next state z^m and its associated output $\hat{y}^m$ through a forward pass in the HRM model:

$$(z^m, \hat{y}^m) \leftarrow \text{HRM}(z^{m-1}, x; \theta)$$

2. Compute the loss for the current segment:

$$L^m \leftarrow \text{LOSS}(\hat{y}^m, y)$$

3. Update parameters:

$$\theta \leftarrow \text{OPTIMIZERSTEP}(\theta, \nabla_\theta L^m)$$

TLDR:

Learn from intermediate thinking
But don't relearn past learnings

Knowing when to stop thinking by predicting rewards?

algorithm[47] to adaptively determine the number of segments. A Q-head uses the final state of the H-module to predict the Q-values $\hat{Q}^m = (\hat{Q}^m_{\text{halt}}, \hat{Q}^m_{\text{continue}})$ of the "halt" and "continue" actions:

$$\hat{Q}^m = \sigma(\theta_Q^\top z_H^{mNT}) \,,$$

The Q-head is updated through a Q-learning algorithm, which is defined on the following episodic Markov Decision Process (MDP). The state of the MDP at segment m is z^m, and the action space is $\{\text{halt}, \text{continue}\}$. Choosing the action "halt" terminates the episode and returns a binary reward indicating prediction correctness, i.e., $\mathbf{1}\{\hat{y}^m = y\}$. Choosing "continue" yields a reward of 0 and the state transitions to z^{m+1}. Thus, the Q-learning targets for the two actions $\hat{G}^m = (\hat{G}^m_{\text{halt}}, \hat{G}^m_{\text{continue}})$ are given by

$$\hat{G}^m_{\text{halt}} = \mathbf{1}\{\hat{y}^m = y\} \,,$$

$$\hat{G}^m_{\text{continue}} = \begin{cases} \hat{Q}^{m+1}_{\text{halt}}, & \text{if } m \geq N_{\max} \,, \\ \max(\hat{Q}^{m+1}_{\text{halt}}, \hat{Q}^{m+1}_{\text{continue}}) \,, & \text{otherwise} \,. \end{cases}$$

We can now define the loss function of our learning procedure. The overall loss for each supervision segment combines both the Q-head loss and the sequence-to-sequence loss:

$$L^m_{\text{ACT}} = \text{LOSS}(\hat{y}^m, y) + \text{BINARYCROSSENTROPY}(\hat{Q}^m, \hat{G}^m) \,.$$

TLDR:

Use two scalar reward values to determine whether to continue next thinking step, or end thinking

BUT: Adaptive Computational Time (ACT) isn't so good!

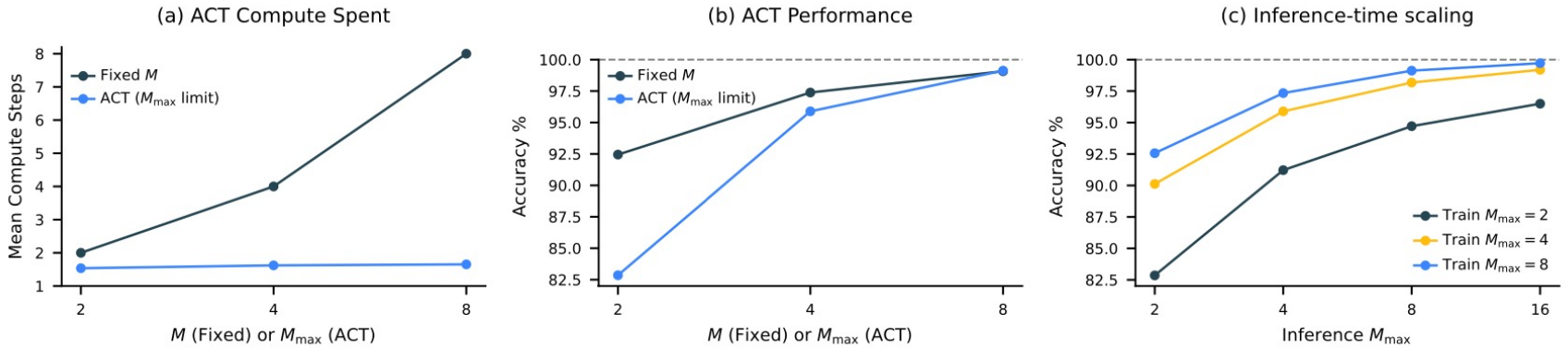


Figure 5: **Effectiveness of Adaptive Computation Time (ACT)** on the *Sudoku-Extreme-Full*. **(a)** Mean compute steps used by models with ACT versus models with a fixed number of compute steps (M). ACT maintains a low and stable number of average compute steps even as the maximum limit (M_{max}) increases. **(b)** Accuracy comparison. The ACT model achieves performance comparable to the fixed-compute model while utilizing substantially fewer computational steps on average. **(c)** Inference-time scalability. Models trained with a specific M_{max} can generalize to higher computational limits during inference, leading to improved accuracy. For example, a model trained with $M_{\text{max}} = 8$ continues to see accuracy gains when run with $M_{\text{max}} = 16$ during inference.

My thoughts

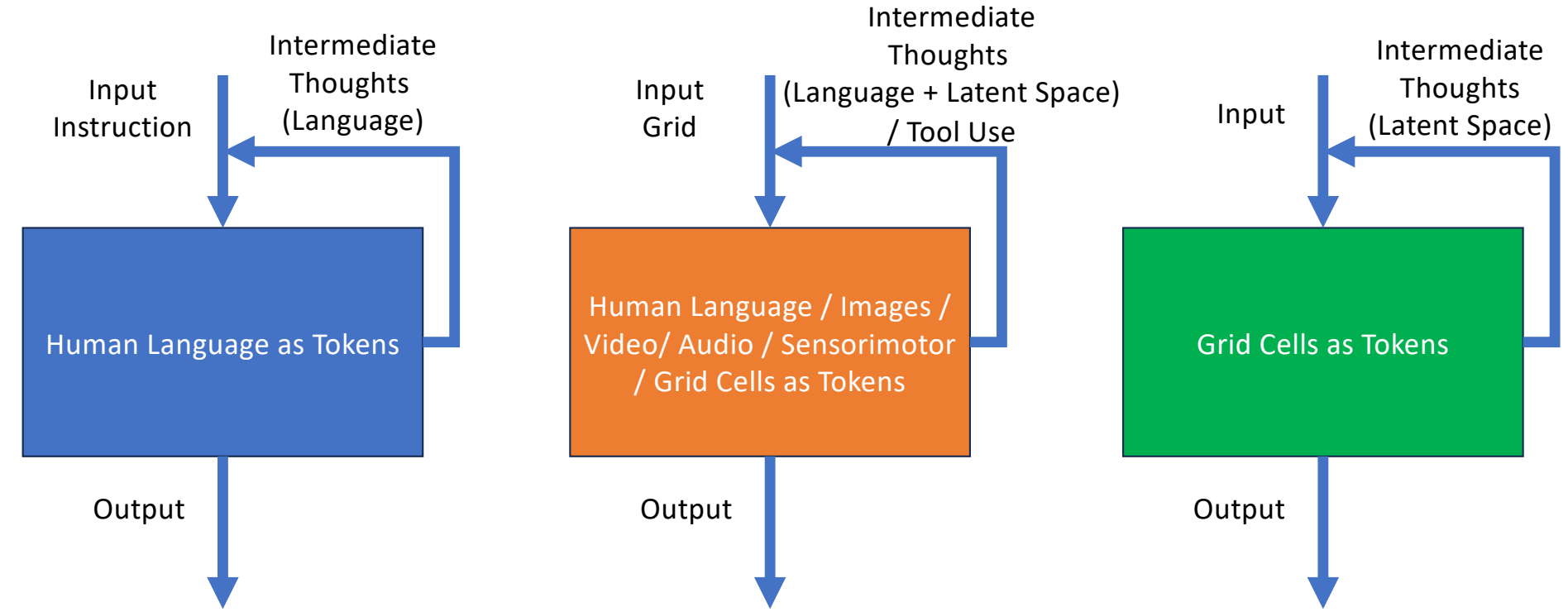
My thoughts (1/2)

- Iterative thinking has its benefits and helps with solving the problem in small bits and with error correction
- Impressive that no logic-based algorithm is used for pathfinding / sudoku solving and it can still navigate the mazes / solve the sudoku puzzles
- Benchmark tasks are only grid tasks, would be interesting to find out if language tasks can be used to provide instruction to HRM as well
- Perhaps we can even visualize language output by just doing forward pass with a certain latent state
- Remains to be seen if HRM can interface with other modules like planner, simulation engines, visual interpreter etc.

My thoughts (2/2)

- Might be possible that traditional LLMs can already perform the recurrent thinking step by stacking multi-headed attention blocks
- The substrate of thinking is currently language / images / audio in terms of tokens, but it could also be made to be grid tokens
- LLMs are highly reliant on training data - remains to be seen if training traditional LLMs using the data augmentation methods described in the paper could already outperform HRM

Can we have a hybrid of language / non-language?



Traditional Text-based Transformers
Thinking in language

Universal Thought Machine
Thinking in anything
+ allows external tool use

Hierarchical Reasoning Model
Thinking in latent space

Question to Ponder

- Can this high-level, low-level thinking cycle proposed in HRM be done back in human language with the current Chain-of-Thought paradigm?
- How explainable is HRM if the thoughts are not in words?
- Will HRM scale to larger input context and datasets?
- Can HRM scale to multimodal, instructional-based, and possibly embodied systems?