CS 957, System-2 AI
Program Synthesis Emerges

Mahdi Samiei | Mar 2025

Sharif University of Technology

Neural Networks and Complexity

One advantage of the program synthesis: Handling tasks with greater complexity.

Through program generation during inference, we could manage any complexity.

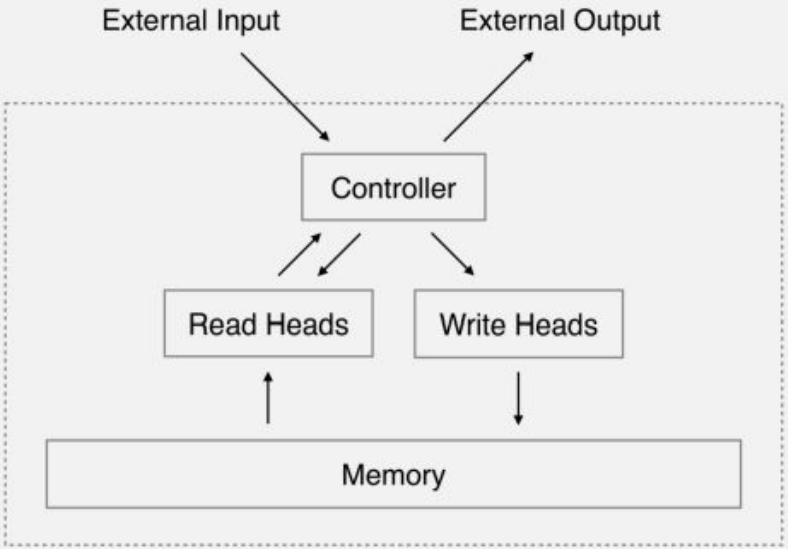
How did mainstream deep learning react?

Deep networks with fixed capacity need an additional missing element.

A simple forward neural network can't be enough

Neural Turing Machine, 2014, Memory Augmented Network

Extending neural networks with memory mechanism.



Neural Turing Machine, 2014

Copy Task: can store and recall a long sequence of arbitrary information?

Simpler Than Parity!

Training sequence lengths were randomised between 1 and 20, memory size 128*20

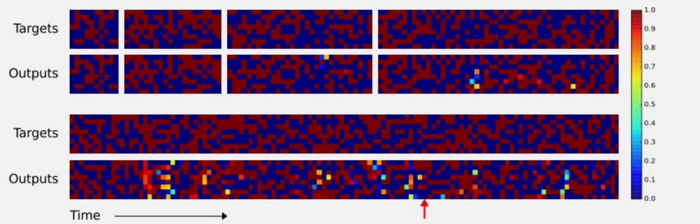


Figure 4: NTM Generalisation on the Copy Task. The four pairs of plots in the top row depict network outputs and corresponding copy targets for test sequences of length 10, 20, 30, and 50, respectively. The plots in the bottom row are for a length 120 sequence. The network was only trained on sequences of up to length 20. The first four sequences are reproduced with high confidence and very few mistakes. The longest one has a few more local errors and one global error: at the point indicated by the red arrow at the bottom, a single vector is duplicated, pushing all subsequent vectors one step back. Despite being subjectively close to a correct copy, this leads to a high loss.

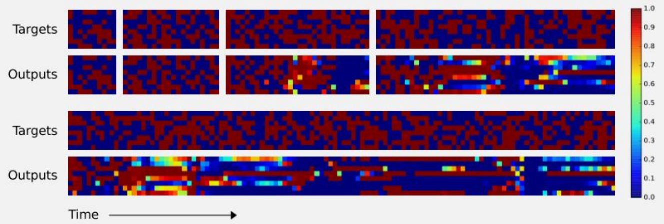


Figure 5: LSTM Generalisation on the Copy Task. The plots show inputs and outputs for the same sequence lengths as Figure 4. Like NTM, LSTM learns to reproduce sequences of up to length 20 almost perfectly. However it clearly fails to generalise to longer sequences. Also note that the length of the accurate prefix decreases as the sequence length increases, suggesting that the network has trouble retaining information for long periods.

Neural Turing Machine, 2014

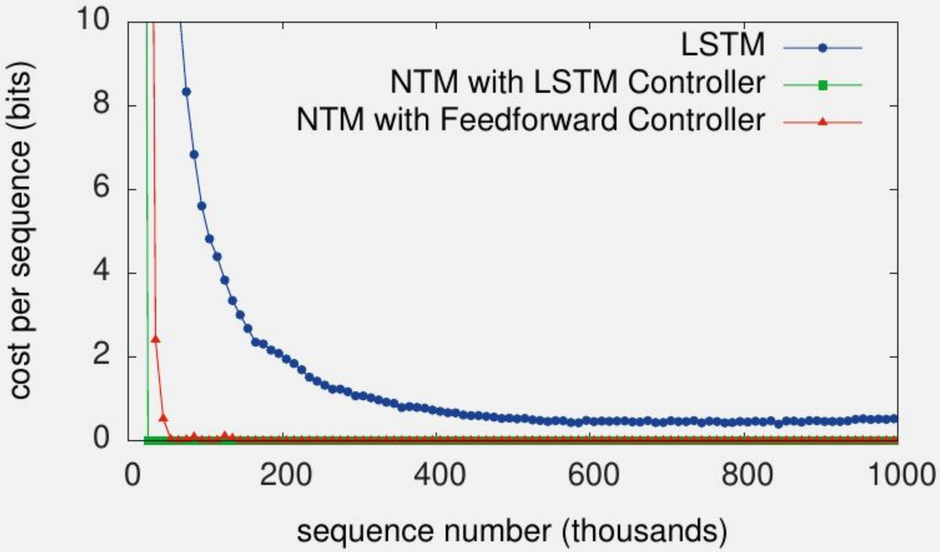


Figure 3: Copy Learning Curves.

Neural Turing Machine, 2014

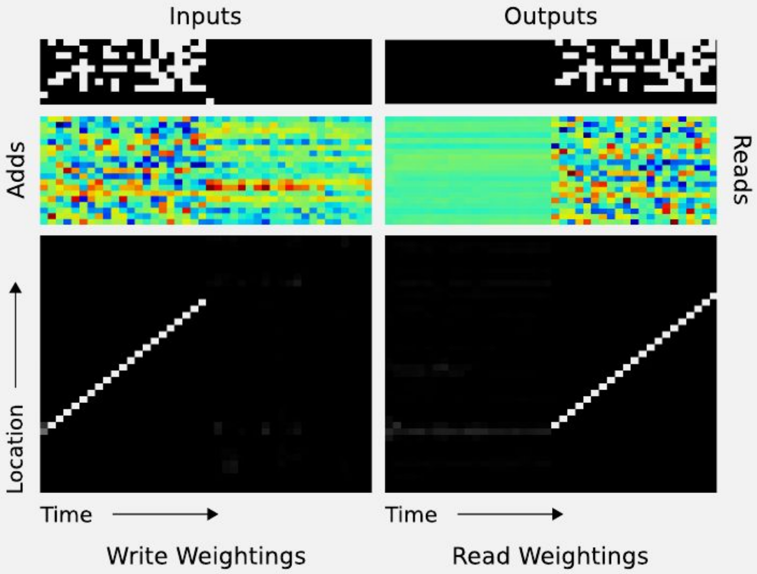


Figure 6: NTM Memory Use During the Copy Task. The plots in the left column depict the inputs to the network (top), the vectors added to memory (middle) and the corresponding write weightings (bottom) during a single test sequence for the copy task. The plots on the right show the outputs from the network (top), the vectors read from memory (middle) and the read weightings (bottom). Only a subset of memory locations are shown. Notice the sharp focus of

Transformer

- 👉 Role and connection of Transformer with program generation paradigm?
- 👉 Attention enables dynamic inference.
- 👉 Key, Value, Queries generated during inference.
- 👉 Activating different computational paths for varying inputs.
- 👉 Transformer paper just has two experiment tables for translation!!!!

Adaptive Computation Time, 2017

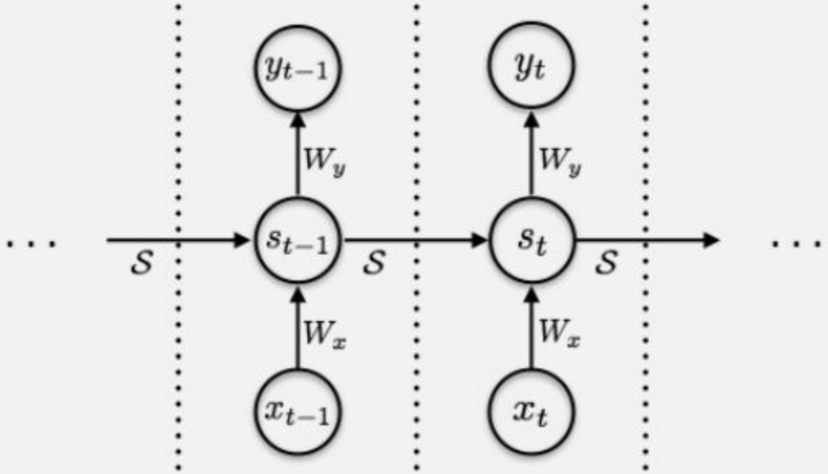


Figure 1: RNN Computation Graph. An RNN unrolled over two input steps (separated by vertical dotted lines). The input and output weights W_x, W_y, and the state transition operator $\mathcal{S}$ are shared over all steps.

Adaptive Computation Time

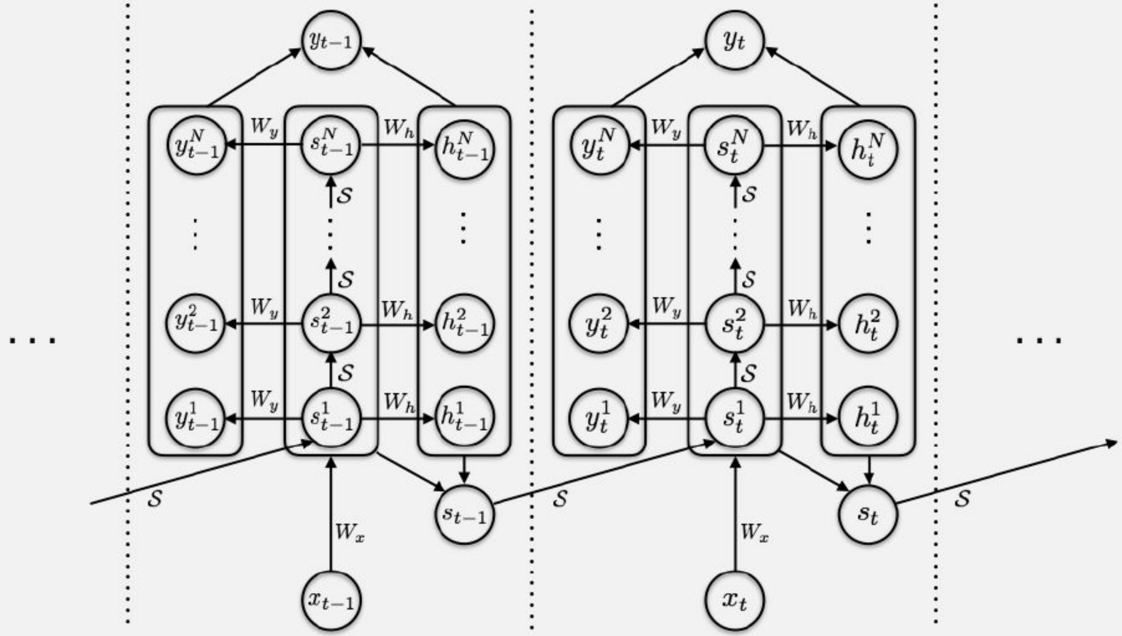
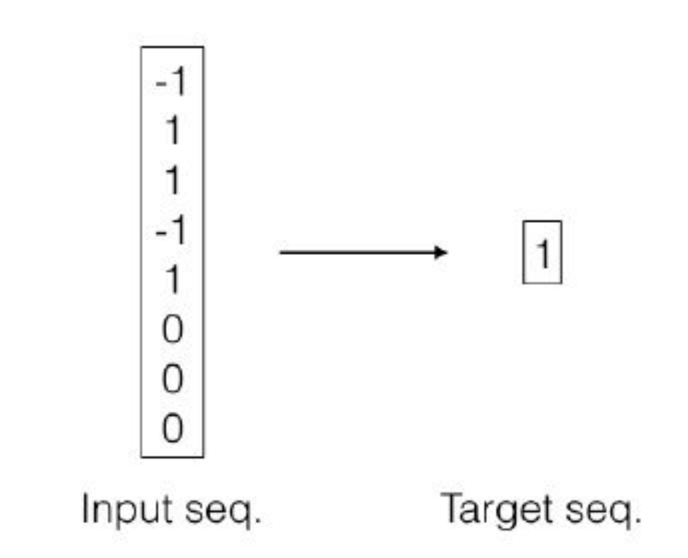


Figure 2: RNN Computation Graph with Adaptive Computation Time. The graph is equivalent to Figure 1 only with each state and output computation expanded to a variable number of intermediate updates. Arrows touching boxes denote operations applied to all units in the box, while arrows leaving boxes denote summations over all units in the box.

Adaptive Computation Time: Parity



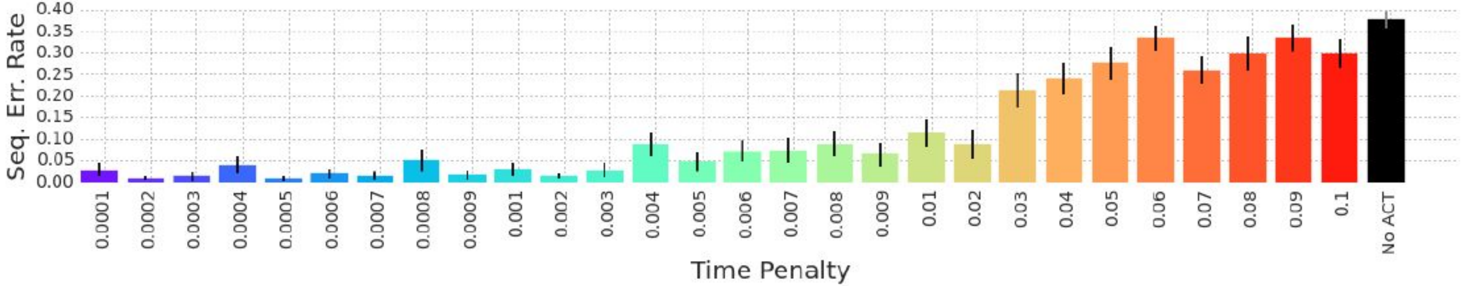


Figure 4: Parity Error Rates. Bar heights show the mean error rates for different time penalties at the end of training. The error bars show the standard error in the mean.

Universal Transformer, 2018

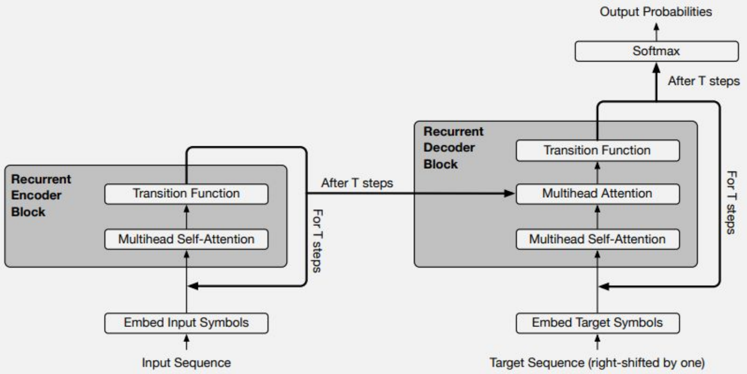
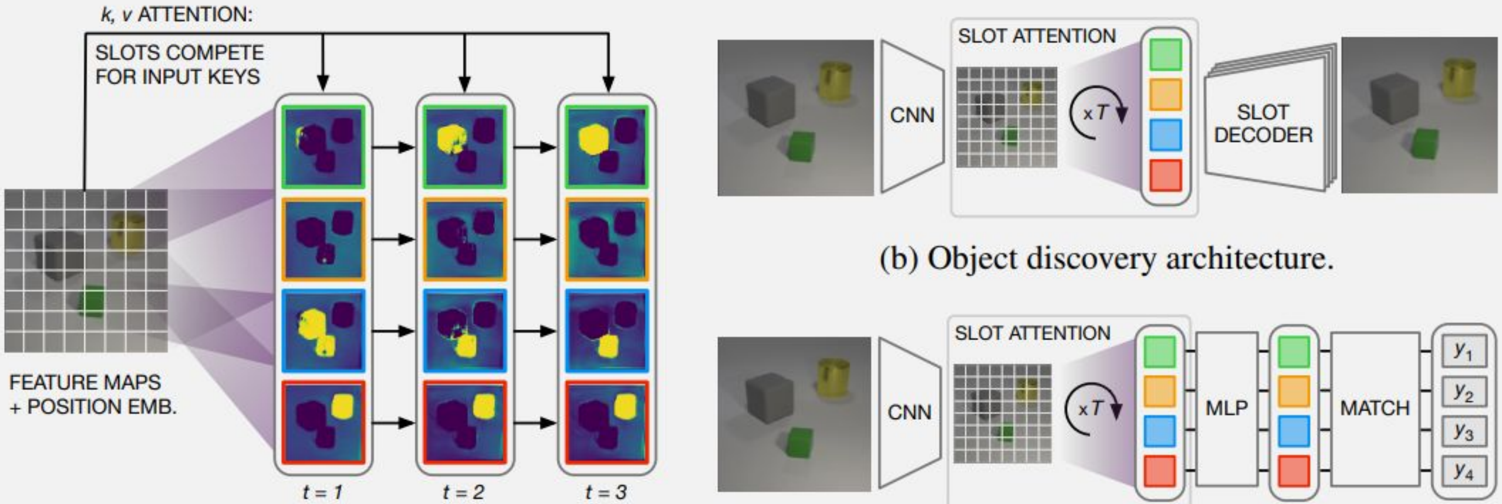


Figure 2: The recurrent blocks of the Universal Transformer encoder and decoder. This diagram omits position and time-step encodings as well as dropout, residual connections and layer normalization. A complete version can be found in Appendix A. The Universal Transformer with dynamic halting determines the number of steps T for each position individually using ACT (Graves, 2016).

Model	**10K examples**		**1K examples**	
	train single	train joint	train single	train joint
	Previous best results:			
QRNet (Seo et al., 2016)	0.3 (0/20)	-	-	-
Sparse DNC (Rae et al., 2016)	-	2.9 (1/20)	-	-
GA+MAGE Dhingra et al. (2017)	-	-	8.7 (5/20)	-
MemN2N Sukhbaatar et al. (2015)	-	-	-	12.4 (11/20)
	Our Results:			
Transformer (Vaswani et al., 2017)	15.2 (10/20)	22.1 (12/20)	21.8 (5/20)	26.8 (14/20)
Universal Transformer (this work)	0.23 (0/20)	0.47 (0/20)	5.31 (5/20)	8.50 (8/20)
UT w/ dynamic halting (this work)	**0.21 (0/20)**	**0.29 (0/20)**	**4.55 (3/20)**	**7.78 (5/20)**

Table 1: Average error and number of failed tasks ($> 5\%$ error) out of 20 (in parentheses; lower is better in both cases) on the bAbI dataset under the different training/evaluation setups. We indicate state-of-the-art where available for each, or '-' otherwise.

Object Centric Representation Learning: Slot Attention



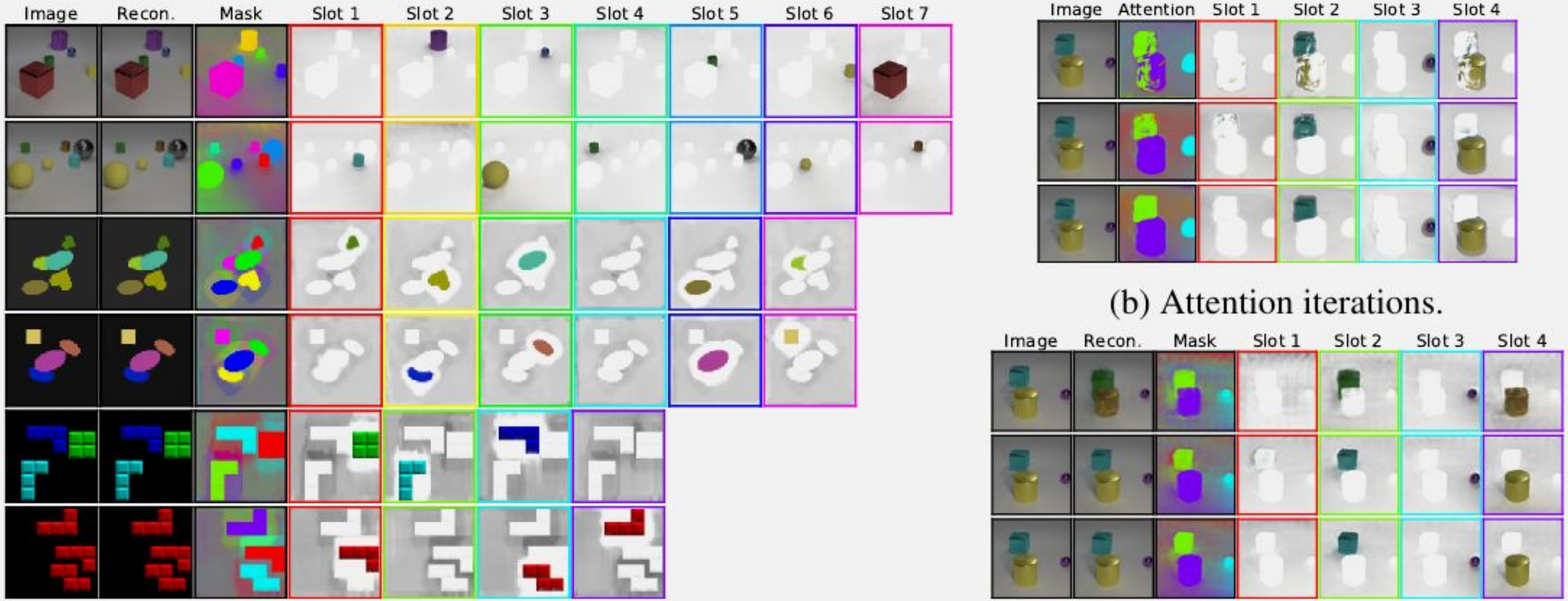
(a) Slot Attention module.

(b) Object discovery architecture.

(c) Set prediction architecture.

Figure 1: (**a**) Slot Attention module and example applications to (**b**) unsupervised object discovery and (**c**) supervised set prediction with labeled targets y_i. See main text for details.

Object Centric Representation Learning: Slot Attention

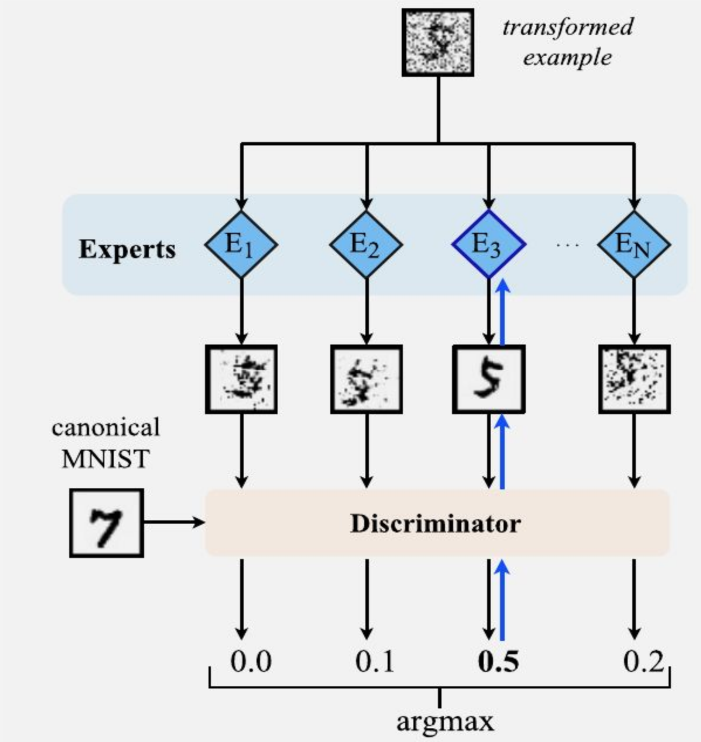


(a) Decomposition across datasets.

(b) Attention iterations.

(c) Reconstructions per iteration.

Learning Independent Mechanisms



Precondition: X: data sampled from P; X': data sampled from $\mathcal{D}_Q$; D discriminator; N': number of experts; T: maximum number of iterations;

$\triangleright$ Initialize experts as approximately identity (**p**):

$\{E_i \leftarrow \text{TrainAsIdentityOn}(X')\}_{j=1}^{N'}$

for $t \leftarrow 1$ to T **do**

$\triangleright$ Sample minibatches:

$x, x' \leftarrow \text{Sample}(X), \text{Sample}(X')$

$\triangleright$ Scores from D for all outputs from the experts (**p**):

$\{c_j \leftarrow D(E_j(x'))\}_{j=1}^{N'}$

$\triangleright$ Update D (**p**):

$$\theta_D^{t+1} \leftarrow \text{Adam}\Big(\theta_D^t, \nabla \log D(x) + \nabla\big(1/N' \textstyle\sum_{j=1}^{N'} \log(1 - c_j)\big)\Big)$$

$\triangleright$ Update experts (**p**):

$\{\theta_{E_j}^{t+1} \leftarrow \text{Adam}(\theta_{E_j}^t, \nabla \max_{j \in 1,\ldots,N'} \log(c_j))\}_{j=1}^{N'}$

Learning Independent Mechanisms for out of distribution

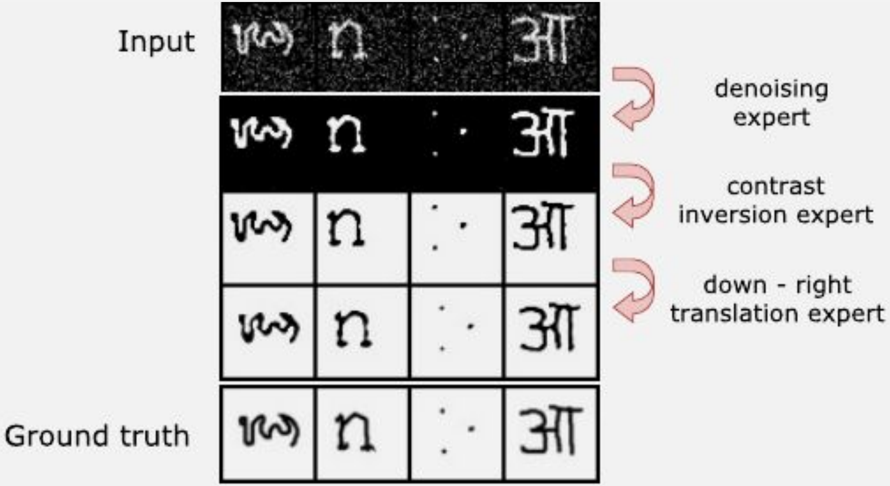


Figure 8. First row: input Omniglot letters that were transformed with noise, contrast inversion and translation up left. Second to fourth row: application of denoising, contrast inverting and right down translating experts. Last row: ground truth. Although the experts were not trained on a combination of mechanisms nor on Omniglot letters, they can be used to recover the original digits.

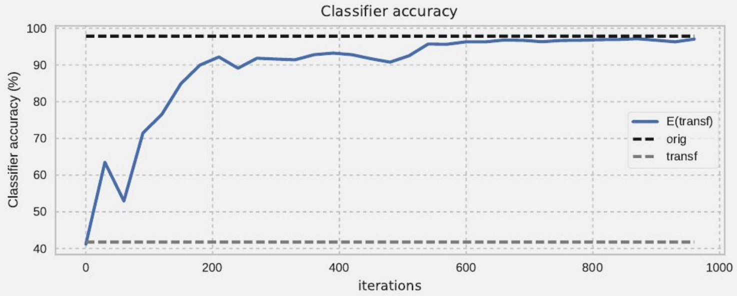
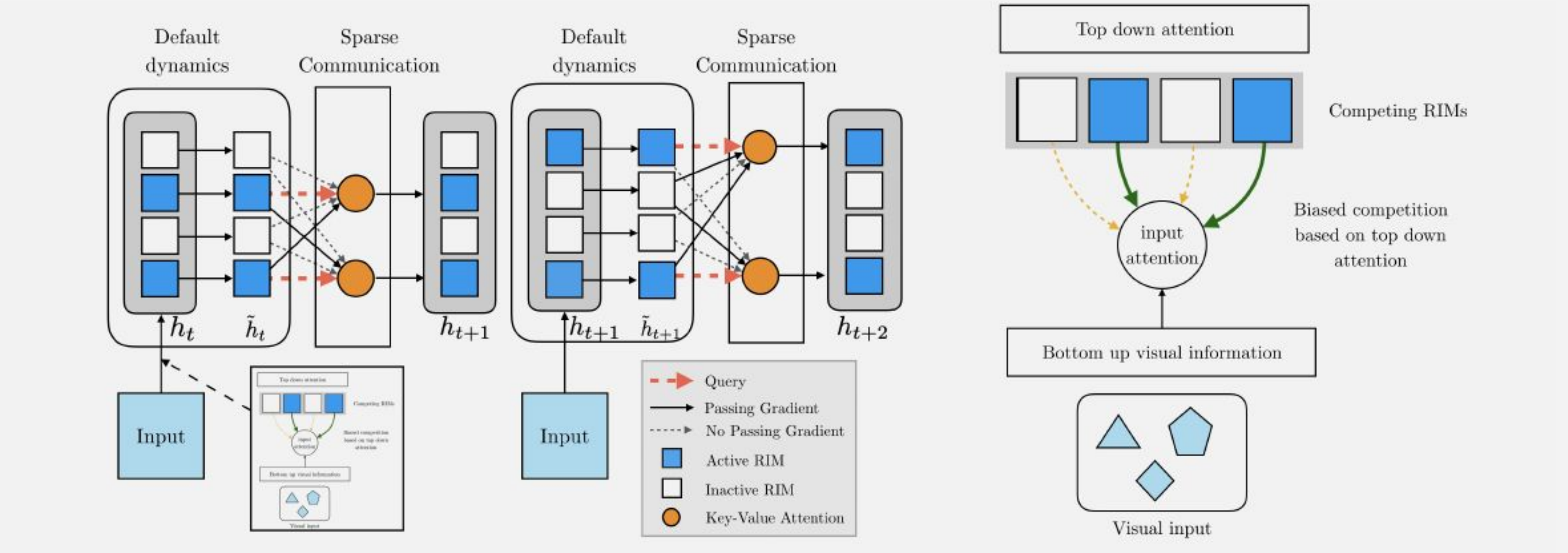


Figure 6. Accuracy of a pretrained CNN MNIST classifier on transformed test digits $\mathcal{D}_Q$, on the same digits after going through our model, and on the original digits. Our system manages to invert the transformations, with the classifier accuracy quickly approaching the optimum. Note that 600 iterations correspond to having seen about a third of the dataset.

RIM: Recurrent Independent Mechanisms



RIM: Recurrent Independent Mechanisms

Copying	k_T	k_A	h_{size}	Train(50) CE	Test(200) CE
RIMs	6	4	600	**0.00**	**0.00**
	6	3	600	**0.00**	**0.00**
	6	2	600	**0.00**	**0.00**
	5	2	500	**0.00**	**0.00**
LSTM	-	-	300	0.00	4.32
	-	-	600	0.00	3.56
NTM	-	-	-	0.00	2.54
RMC	-	-	-	0.00	0.13
Transformers	-		-	0.00	0.54

Sequential MNIST	k_T	k_A	h_{size}	16 x 16 Accuracy	19 x 19 Accuracy	24 x 24 Accuracy
RIMs	6	6	600	85.5	56.2	30.9
	6	5	600	88.3	43.1	22.1
	6	4	600	**90.0**	**73.4**	**38.1**
LSTM	-	-	300	86.8	42.3	25.2
	-	-	600	84.5	52.2	21.9
EntNet	-	-	-	89.2	52.4	23.5
RMC	-	-	-	89.58	54.23	27.75
DNC	-	-	-	87.2	44.1	19.8
Transformers	-		-	**91.2**	51.6	22.9

Table 1: Performance on the copying task (left) and Sequential MNIST resolution generalization (right). While all of the methods are able to learn to copy for the length seen during training, the RIMs model generalizes to sequences longer than those seen during training whereas the LSTM, RMC, and NTM degrade much more. On sequential MNIST, both the proposed and the Baseline models were trained on 14x14 resolution but evaluated at different resolutions (averaged over 3 trials).

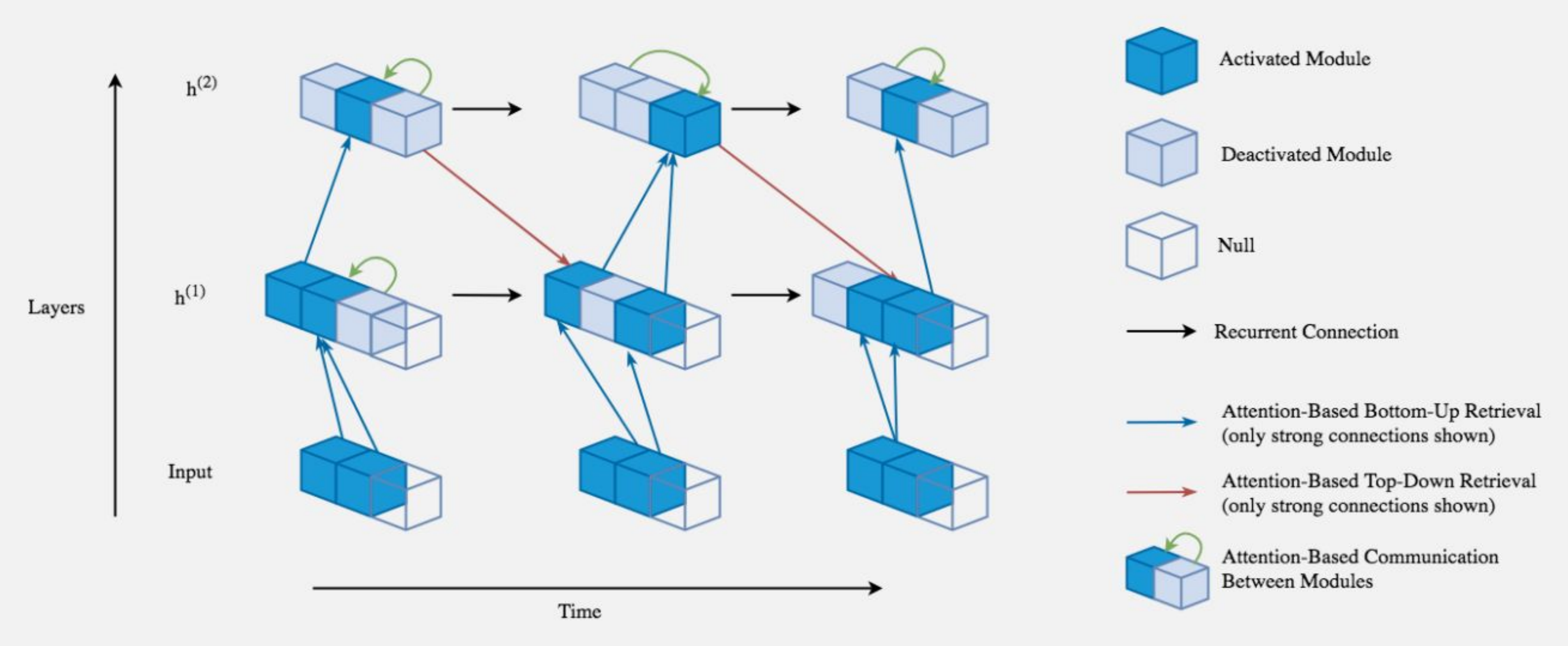
Bottom-Up VS. Top-Down

Hierarchical models are often taken to imply that computation proceeds in a feedforward or bottom up fashion, information processing in which low-level (sensory) representations construct or modulate high level (conceptual) representations.

feedback or top down fashion, information processing in which high-level representations modulate lower-level representations.

One goal is to conduct machine-learning experiments to <u>explore the value of top-down mechanisms</u> for learning, <u>achieving robustness to distributional shifts</u>.

Learning to Combine Top-Down and Bottom-Up Signals



S. Mittal, A. Lamb, A. Goyal, et, and Y. Bengio, "Learning to combine top-down and bottom-up signals in recurrent neural networks with attention over modules," 2020

BRIM: Some Experiments

Algorithm	*Properties*	*16×16*	*19×19*	*24×24*
LSTM	—	86.8	42.3	25.2
LSTM	H	87.2	43.5	22.9
LSTM	H+B	83.2	44.4	25.3
LSTM	H+A	84.3	47.5	31.0
LSTM	H+A+B	83.2	40.1	20.8
RMC	A	89.6	54.2	27.8
Transformers	H+A+B	91.2	51.6	22.9
RIMs	A+M	88.9	67.1	38.1
Hierarchical RIMs	H+A+M	85.4	72.0	50.3
MLD-RIMs	H+A+M	88.8	69.1	45.3
BRIMs (ours)	H+A+B+M	88.6	74.2	51.4

Table 1. Performance on the **Sequential MNIST resolution generalization:** Test Accuracy % after 100 epochs. All models were trained on 14x14 resolution but evaluated at different resolutions; results averaged over 3 different trials.

Algorithm	*Properties*	*19×19*	*24×24*	*32×32*
LSTM	—	54.4	44.0	32.2
LSTM	H	57.0	46.8	33.2
LSTM	H+B	56.5	52.2	42.1
LSTM	H+A	56.7	51.5	40.0
LSTM	H+A+B	59.9	54.6	43.0
RMC	A	49.9	44.3	31.3
RIMs	A+M	56.9	51.4	40.1
Hierarchical RIMs	H+A+M	57.2	54.6	46.8
MLD-RIMs	H+A+M	56.8	53.1	44.5
BRIMs (ours)	H+A+B+M	60.1	57.7	52.2

Table 2. Performance on **Sequential CIFAR generalization:** Test Accuracy % after 100 epochs. Both the proposed and the Baseline model (LSTM) were trained on 16x16 resolution but evaluated at different resolutions; results averaged over 3 different trials.

S. Mittal, A. Lamb, A. Goyal, et, and Y. Bengio, "Learning to combine top-down and bottom-up signals in recurrent neural networks with attention over modules," 2020

BRIM: Some Experiments

Learning to Combine Top-Down and Bottom-Up Signals

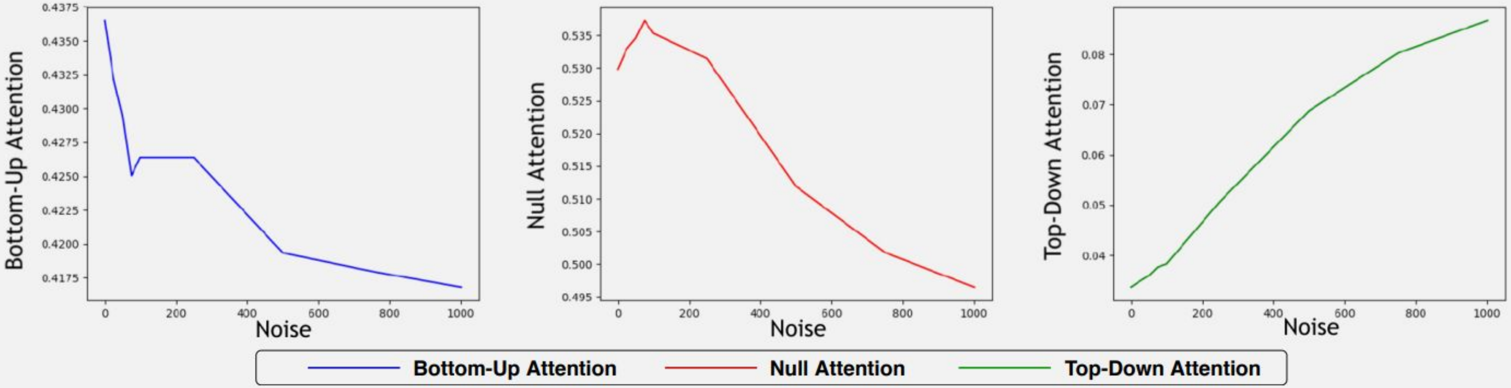
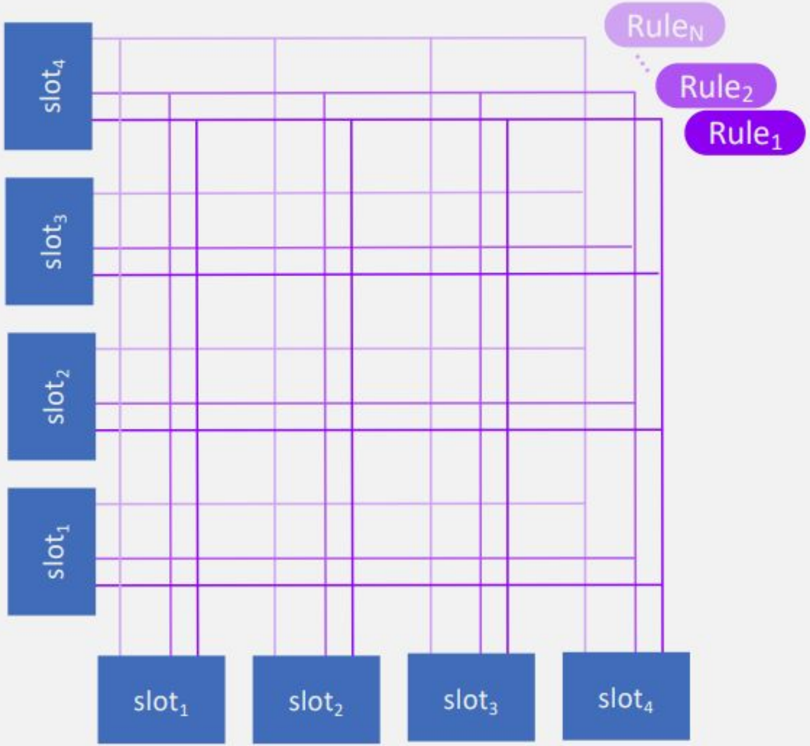


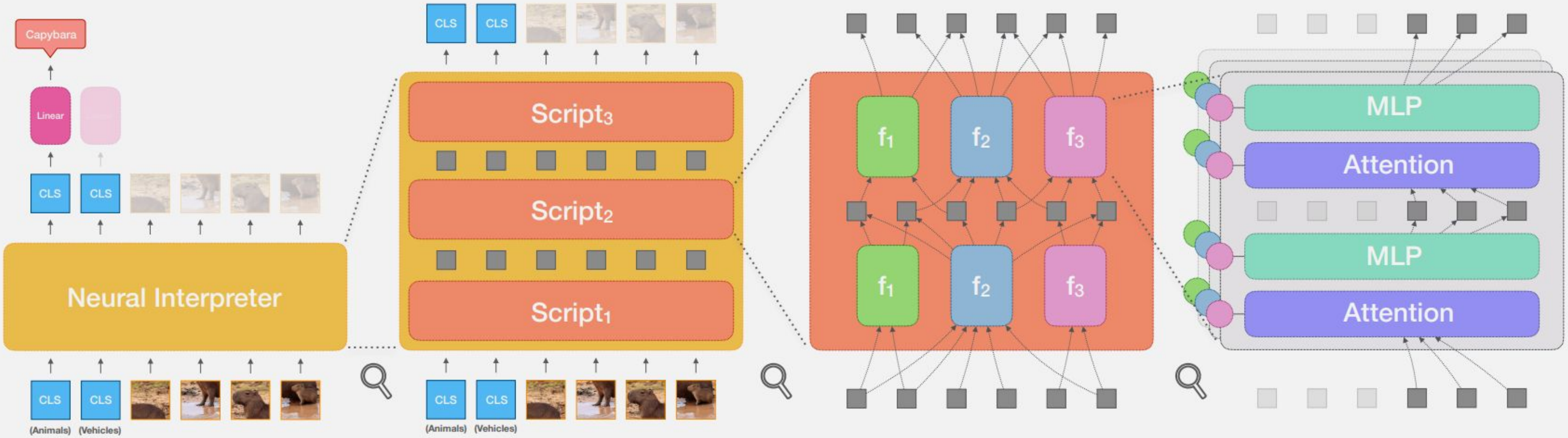
Figure 2. On sequential CIFAR-10, we evaluate on 32x32 test images and change a certain number of the pixels to random values. We show the **average activation weight (y-axis)** on Input (left), Null (middle) and Higher Layer (right) plotted against the **number of random pixels (x-axis)** (total number of pixels = 1024). We see that as more pixels are set to random values, the model becomes increasingly reliant on the higher-level information.

S. Mittal, A. Lamb, A. Goyal, et, and Y. Bengio, “Learning to combine top-down and bottom-up signals in recurrent neural networks with attention over modules,” 2020

NPS: Neural Production System



Dynamic Neural Interpreter



Higher Level Inductive Biases

💡 factoring procedural and declarative knowledge

💡 Variable Length Representation

💡 Dynamic computational path (dependent on input)

💡 Modular Computational Units

💡 Discreteness of semantics units

...

LLM was the key to program synthesis

🤔 What is analogous between llm and program synthesis?

👉 Program synthesis methods generate a program, that program is executed by an external executor.

👉 What about LLM?

👉 LLMs generate programs (natural language tokens) and are executed by the model itself as an executor.

🤔 How can we better utilize the program of natural language generated by language models from this programming perspective?

👉 Chain of Thought ...

Next Sessions:

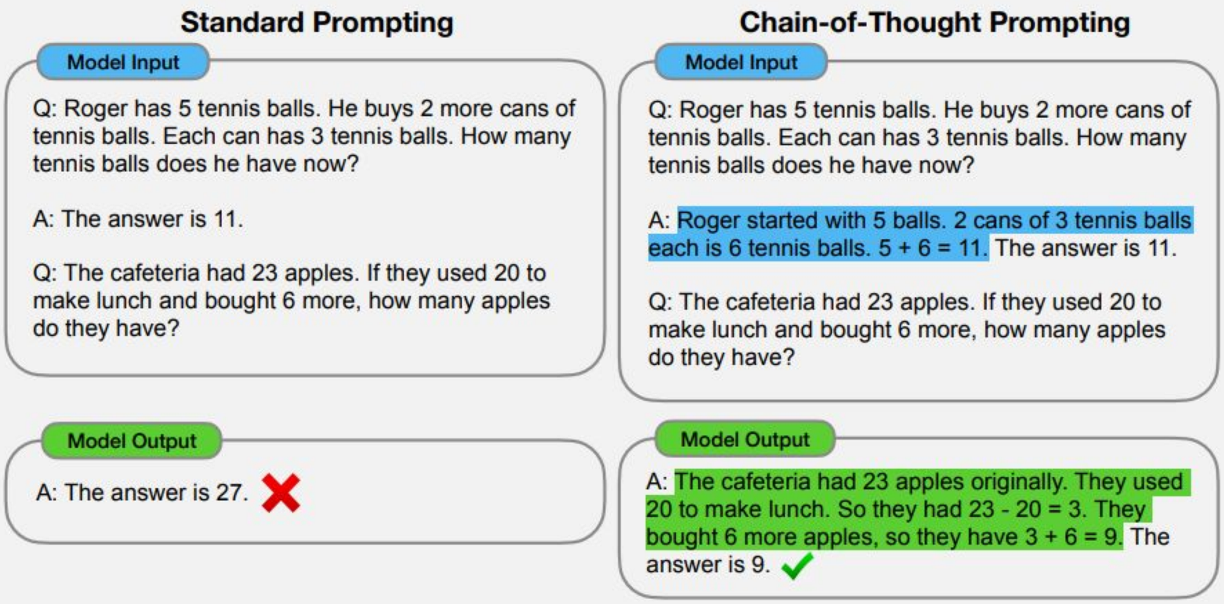


Figure 1: Chain-of-thought prompting enables large language models to tackle complex arithmetic, commonsense, and symbolic reasoning tasks. Chain-of-thought reasoning processes are highlighted.