ETH zürich

Energy-Based Transformers are Scalable Learners and Thinkers

Alexi Gladstone, Ganesh Nanduru, Md Mofijul Islam, Peixuan Han, Hyeonjeong Ha, Aman Chadha, Yilun Du, Heng Ji, Jundong Li, Tariq Iqbal

Presented by Benedict Armstrong & Supervised by Joël Mathys

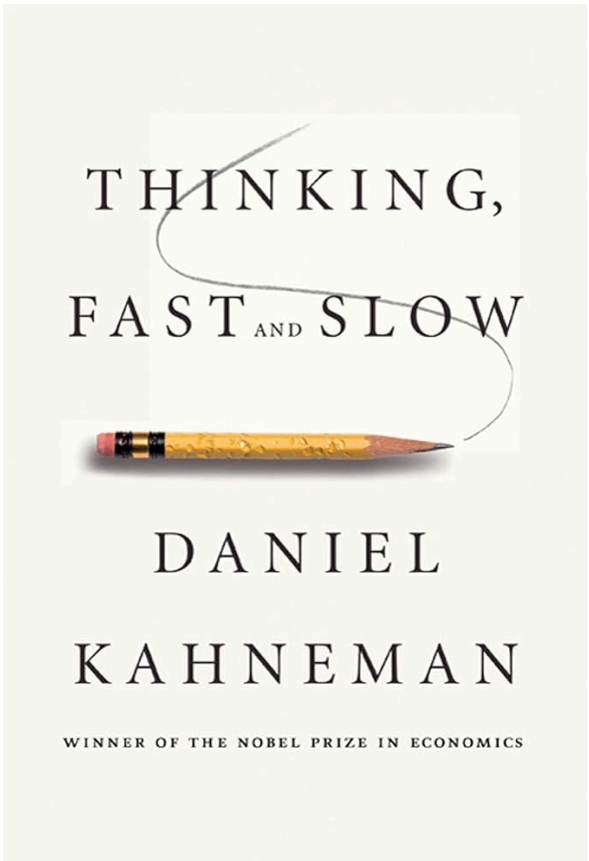


Figure 1: Thinking fast and slow by Daniel Kahneman (Kahneman, 2011)

System 1 vs System 2 Thinking

System 1 — Fast

- Quick, intuitive, automatic
- Pattern recognition
- "What is $2 + 2$?"

System 1 vs System 2 Thinking

System 1 — Fast

- Quick, intuitive, automatic
- Pattern recognition
- "What is $2 + 2$?"

System 2 — Slow

- Deliberate, analytical, effortful
- Logical reasoning, planning
- "What is 173×242?"

System 1 vs System 2 Thinking

System 1 – Fast

- Quick, intuitive, automatic
- Pattern recognition
- "What is $2 + 2$?"

System 2 – Slow

- Deliberate, analytical, effortful
- Logical reasoning, planning
- "What is 173×242?"

To further improve models we want models with better System 2 Thinking capabilities.

Three Facets of System 2 Thinking

- **Facet 1:** Allocate more compute to harder problems

Three Facets of System 2 Thinking

- **Facet 1:** Allocate more compute to harder problems
- **Facet 2:** Express how uncertain a prediction is

Three Facets of System 2 Thinking

- **Facet 1:** Allocate more compute to harder problems
- **Facet 2:** Express how uncertain a prediction is
- **Facet 3:** Verify quality of predictions

“Is it possible to **generalize** these System 2 Thinking approaches, and **develop models that learn to think solely from unsupervised learning?**”

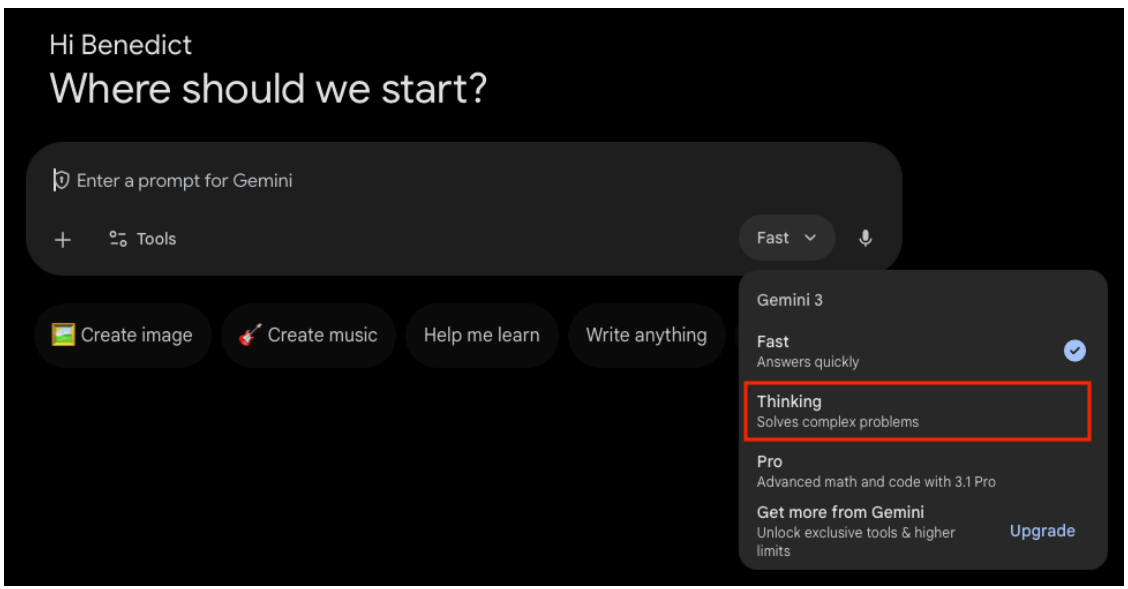
Related Work

Text based “thinking” Models

- Initially **Chain of Thought (CoT)** (Wei et al., 2023)

Text based “thinking” Models

- Initially **Chain of Thought (CoT)** (Wei et al., 2023)
- RL trained “thinking” (Guo et al., 2025) (DeepSeek R1)



Text based “thinking” Models

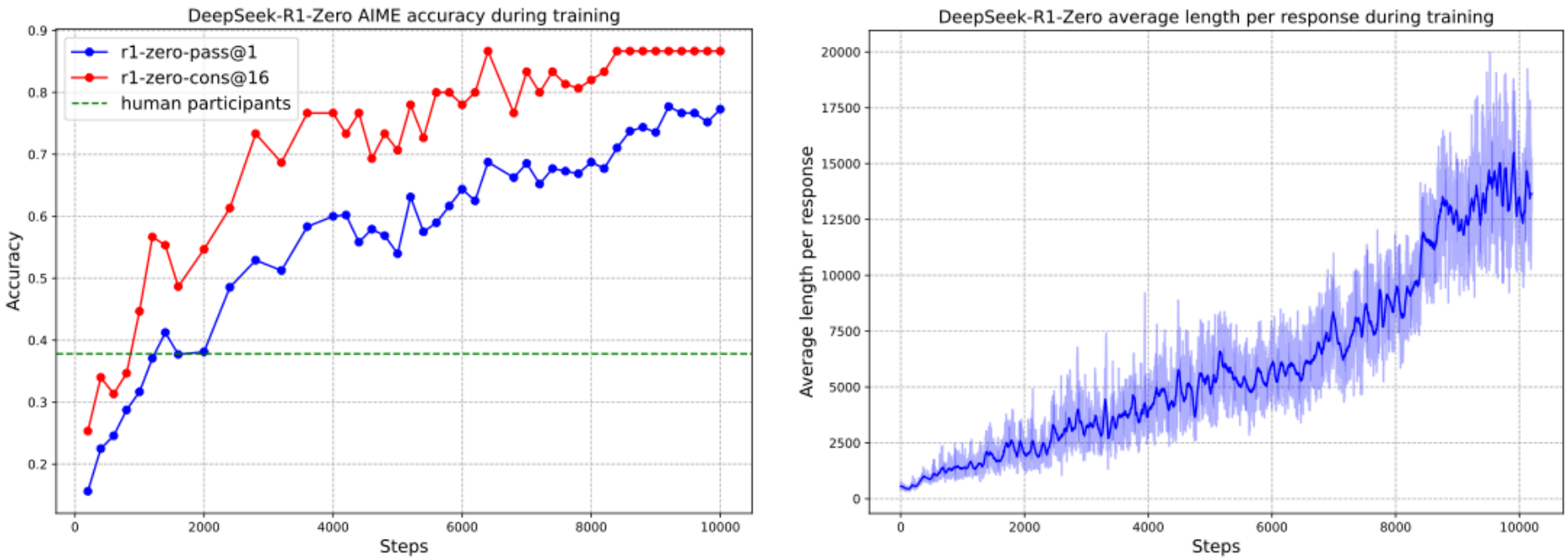


Figure 3: DeepSeek R1 average response length during post training RL (Guo et al., 2025)

Text based “thinking” Models

- Discrete thinking in tokens
- Fixed compute per token
- Only applicable for text
- requires RL supervision (verifiable rewards)

Recurrent Models[1]

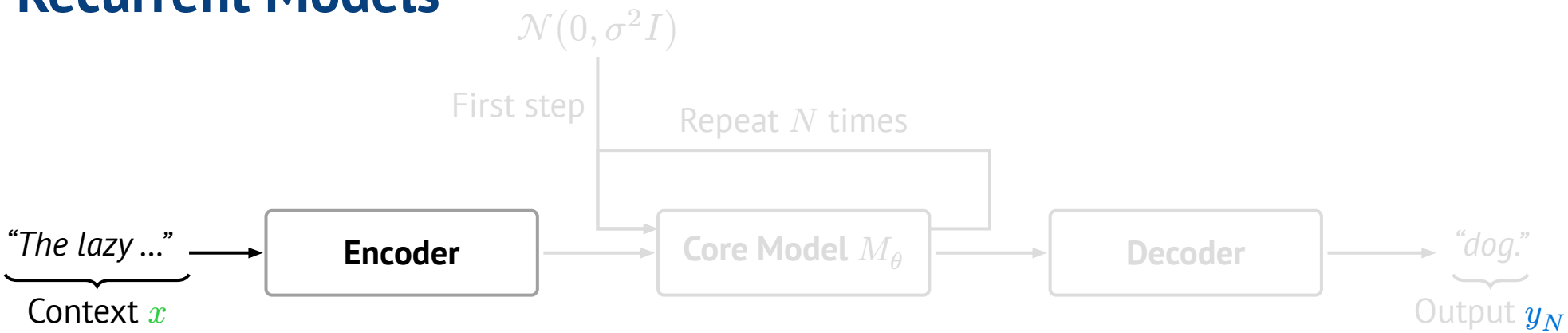


Figure 4: Forward pass in Recurrent Models

[1] (Dehghani et al., 2019; Geiping et al., 2025; Jolicoeur-Martineau, 2025)

Recurrent Models[2]

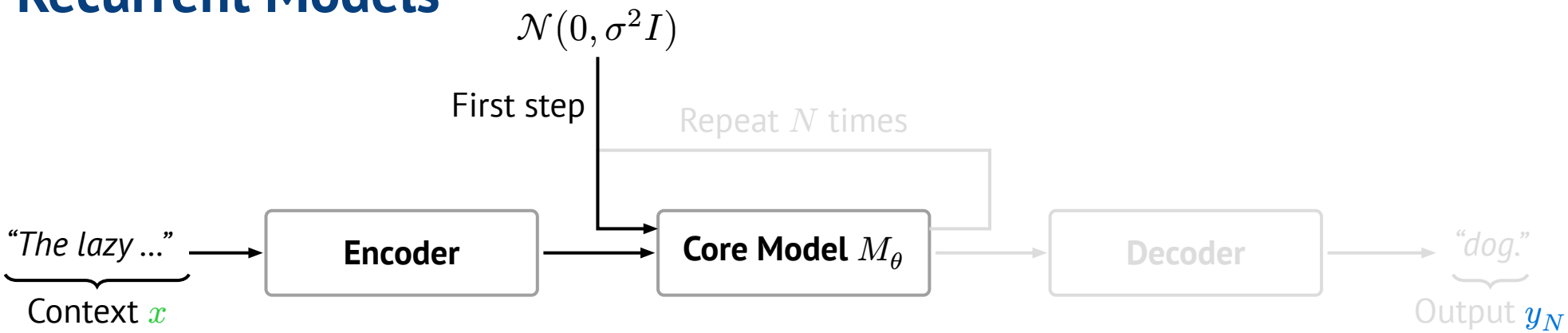


Figure 5: Forward pass in Recurrent Models

[2](Dehghani et al., 2019; Geiping et al., 2025; Jolicoeur-Martineau, 2025)

Recurrent Models[3]

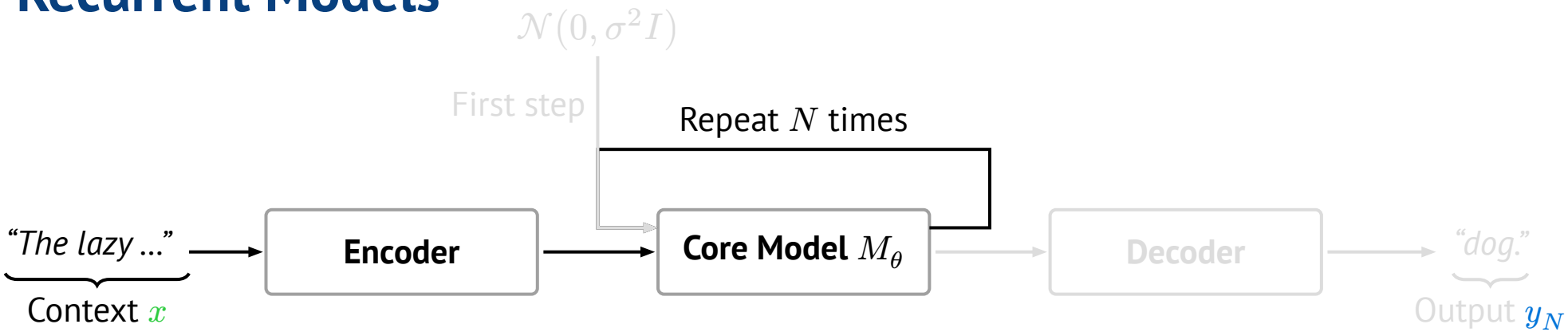


Figure 6: Forward pass in Recurrent Models

[3](Dehghani et al., 2019; Geiping et al., 2025; Jolicoeur-Martineau, 2025)

Recurrent Models[4]

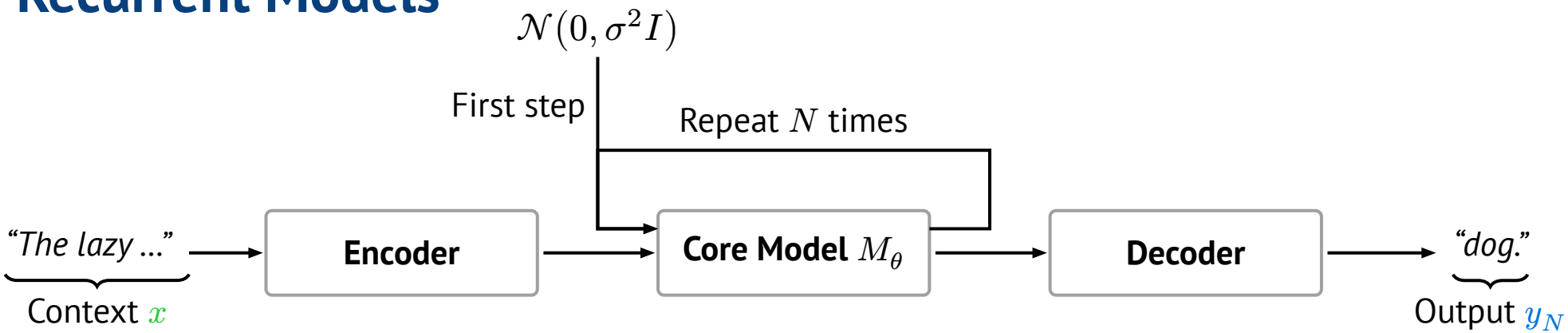


Figure 7: Forward pass in Recurrent Models

[4](Dehghani et al., 2019; Geiping et al., 2025; Jolicoeur-Martineau, 2025)

Recurrent Models[5]

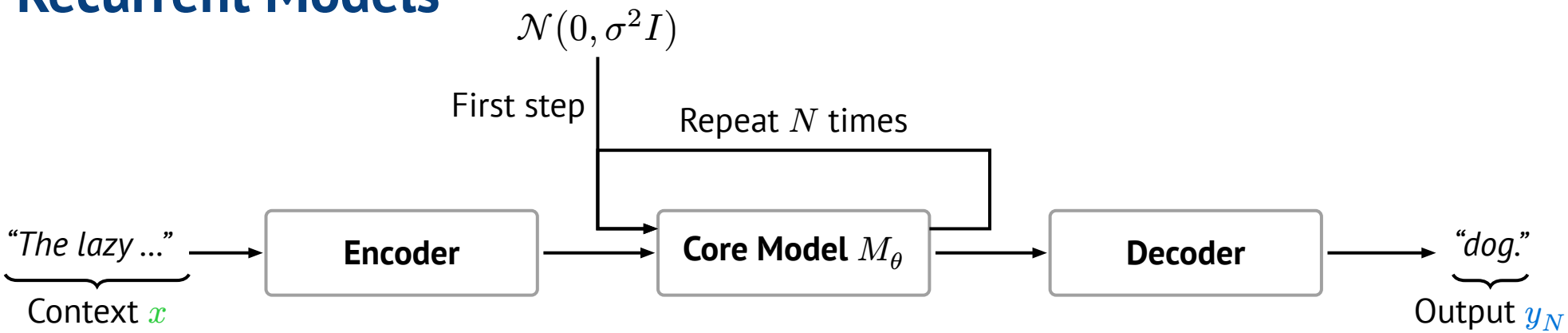


Figure 8: Forward pass in Recurrent Models

Inside our core model:

$$\boldsymbol{y_0} \sim \mathcal{N}(0, \sigma^2 I)$$

[5](Dehghani et al., 2019; Geiping et al., 2025; Jolicoeur-Martineau, 2025)

Recurrent Models[6]

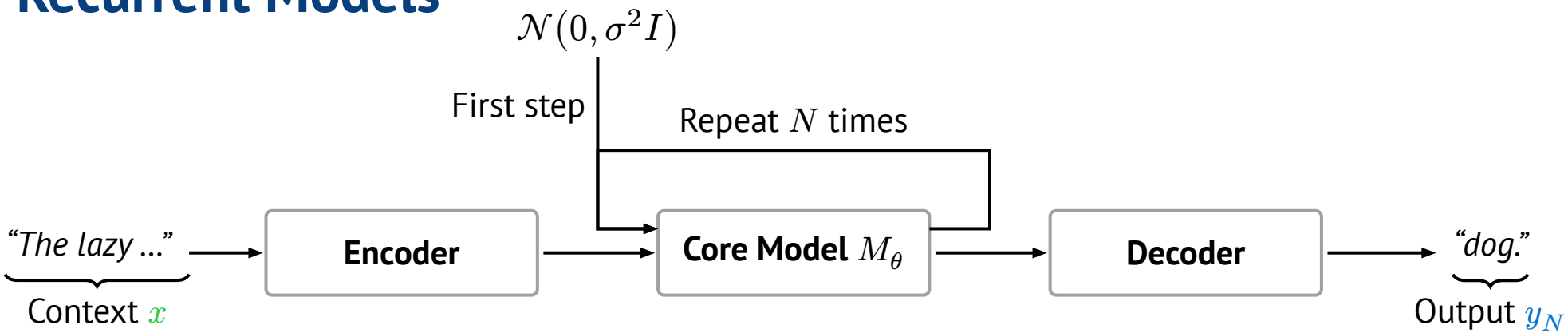


Figure 9: Forward pass in Recurrent Models

Inside our core model:

$$\boldsymbol{y_0} \sim \mathcal{N}(0, \sigma^2 I)$$

$$\boldsymbol{y_n} = \boldsymbol{y_{n-1}} - \alpha M_\theta(\boldsymbol{x}, \boldsymbol{y_{n-1}})$$

[6](Dehghani et al., 2019; Geiping et al., 2025; Jolicoeur-Martineau, 2025)

Recurrent Models[7]

- Thinking in (continuous) latent space
- Fixed compute per token
- No obvious verifiability

[7] (Dehghani et al., 2019; Geiping et al., 2025; Jolicoeur-Martineau, 2025)

Diffusion models[8]

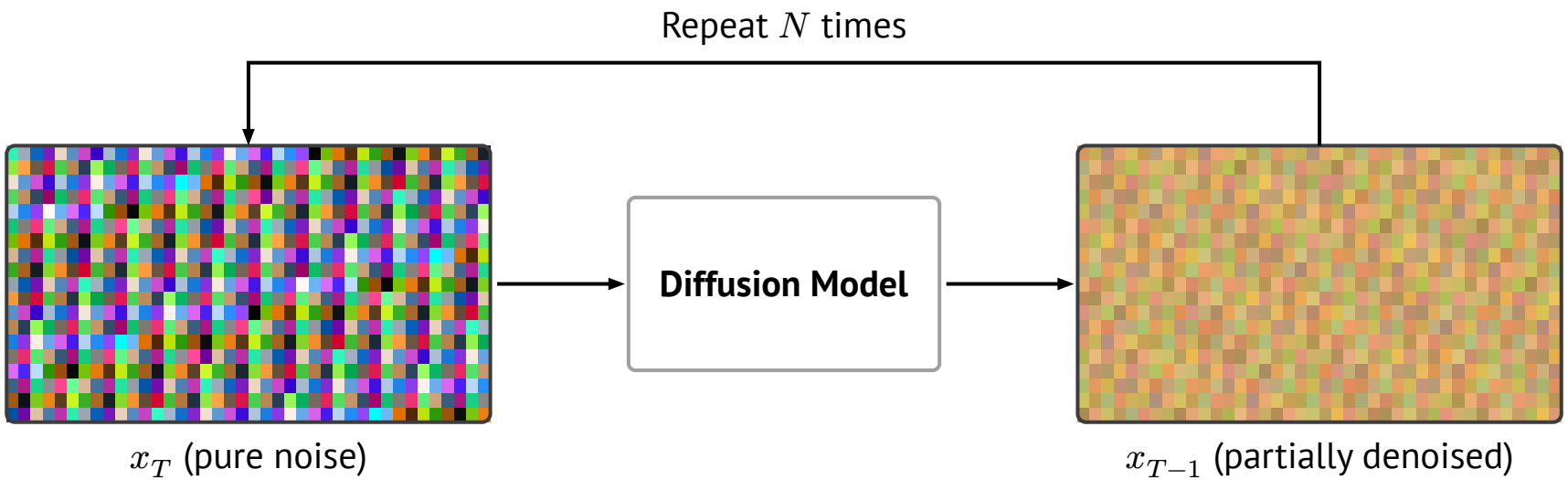


Figure 10: One step in a Diffusion model

[8] (Ho et al., 2020)

Diffusion models[9]

- Thinking in latent space
- Variable compute per token
- No obvious verifiability
- Supervision at every step during training

[9] (Ho et al., 2020)

Energy Based Models (EBMs)

Energy-Based Models (EBMs)[10]

[10](Teh et al., 2003)

Energy-Based Models (EBMs)

For a state s and energy function E:

$$p(s) = \frac{1}{Z} e^{-\mathrm{E}\,(s)}$$

where Z is a normalization integral over all states such that $\int p = 1$.

Energy-Based Models (EBMs)

$$s = \begin{cases} \boldsymbol{x} : \text{“The dog catches the“} \\ \boldsymbol{y} : \text{“frisbee“} \end{cases}$$

$$p(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{Z} e^{-\mathrm{E}\ (\boldsymbol{x}, \boldsymbol{y})}$$

where Z is a normalization integral over all states such that $\int p = 1$.

Energy-Based Models (EBMs)

$$s = \begin{cases} x : \text{“The dog catches the“} \\ y : \text{“frisbee“} \end{cases}$$

$$p(x, y) \propto e^{-\mathrm{E}\,(x,y)}$$

Energy-Based Models (EBMs)

$$s = \begin{cases} x : \text{“The dog catches the“} \\ y : \text{“frisbee“} \end{cases}$$

$$p(x, y) \propto e^{-E_\theta(x,y)}$$

Energy-Based Models (EBMs)

$$E_\theta(\text{“The dog catches the“}, \text{“frisbee“}) = \textbf{Low} \text{ energy value}$$

Energy-Based Models (EBMs)

$$E_\theta(\text{“The dog catches the“}, \text{“frisbee“}) = \textbf{Low} \text{ energy value}$$

$$E_\theta(\text{“The dog catches the“}, \text{“seminar“}) = \textbf{High} \text{ energy value}$$

Energy-Based Models (EBMs)

$$E_\theta(\text{“The dog catches the“}, \text{“frisbee“}) = \textbf{Low} \text{ energy value}$$

$$E_\theta(\text{“The dog catches the“}, \text{“seminar“}) = \textbf{High} \text{ energy value}$$

The model acts as a **verifier**: given any (x, y) pair, it outputs a scalar: “how well does this prediction y match the input x?”

Energy-Based Models (EBMs)

$$E_\theta(\text{“The dog catches the“}, \text{“frisbee“}) = \textbf{Low} \text{ energy value}$$

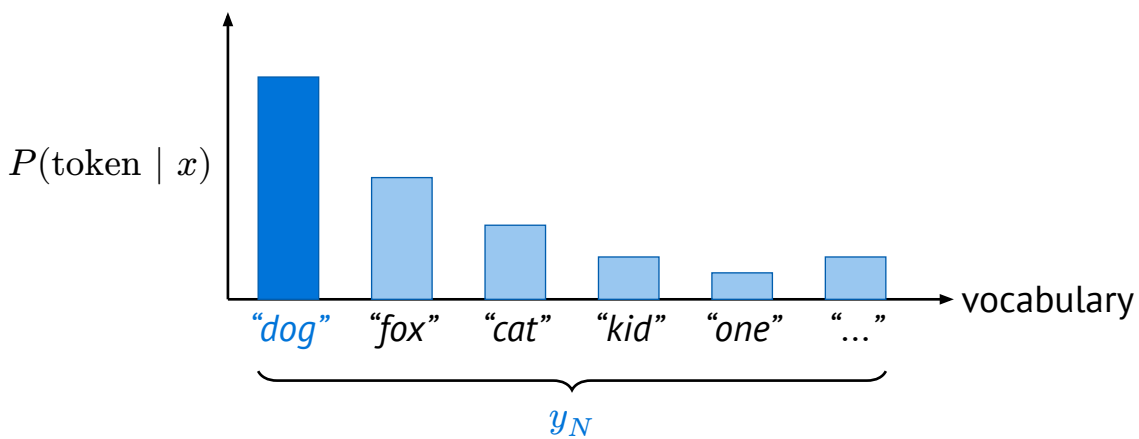
$$E_\theta(\text{“The dog catches the“}, \text{“seminar“}) = \textbf{High} \text{ energy value}$$

The model acts as a **verifier**: given any (x, y) pair, it outputs a scalar: “how well does this prediction y match the input x?”

We can compare outputs y_1 and y_2 by their energy $E_\theta(x, y_1) \leq E_\theta(x, y_2)$.

Quick aside: Output Distribution

y is not a single token but a **probability distribution** over the vocabulary:



Energy-Based Transformers are Scalable Learners and Thinkers

Energy-Based Transformers are Scalable Learners and Thinkers

EBT: Verification as Generation

EBT: Verification as Generation

Sample initial starting state $\hat{y}_0$ from:

$$\hat{y}_0 \sim \mathcal{N}(0, I)$$

EBT: Verification as Generation

Sample initial starting state $\hat{y}_0$ from:

$$\hat{\boldsymbol{y}}_0 \sim \mathcal{N}(0, I)$$

To generate a final prediction we do gradient descent on the energy function:

$$\hat{\boldsymbol{y}}_{i+1} = \hat{\boldsymbol{y}}_i - \alpha \nabla_{\hat{\boldsymbol{y}}_i} E_\theta(\boldsymbol{x}, \hat{\boldsymbol{y}}_i)$$

EBT: Verification as Generation

Sample initial starting state $\hat{y}_0$ from:

$$\hat{\boldsymbol{y}}_0 \sim \mathcal{N}(0, I)$$

To generate a final prediction we do gradient descent on the energy function:

$$\hat{\boldsymbol{y}}_{i+1} = \hat{\boldsymbol{y}}_i - \alpha \nabla_{\hat{\boldsymbol{y}}_i} E_\theta(\boldsymbol{x}, \hat{\boldsymbol{y}}_i)$$

- Each step = one “thought step”

EBT: Verification as Generation

Sample initial starting state $\hat{y}_0$ from:

$$\hat{y}_0 \sim \mathcal{N}(0, I)$$

To generate a final prediction we do gradient descent on the energy function:

$$\hat{y}_{i+1} = \hat{y}_i - \alpha \nabla_{\hat{y}_i} E_\theta(x, \hat{y}_i)$$

- Each step = one “thought step”
- Energy at each step = verification score

EBT: Verification as Generation

Sample initial starting state $\hat{y}_0$ from:

$$\hat{y}_0 \sim \mathcal{N}(0, I)$$

To generate a final prediction we do gradient descent on the energy function:

$$\hat{y}_{i+1} = \hat{y}_i - \alpha \nabla_{\hat{y}_i} E_\theta(x, \hat{y}_i)$$

- Each step = one “thought step”
- Energy at each step = verification score
- Convergence = “I’m confident enough to stop”

EBT Training

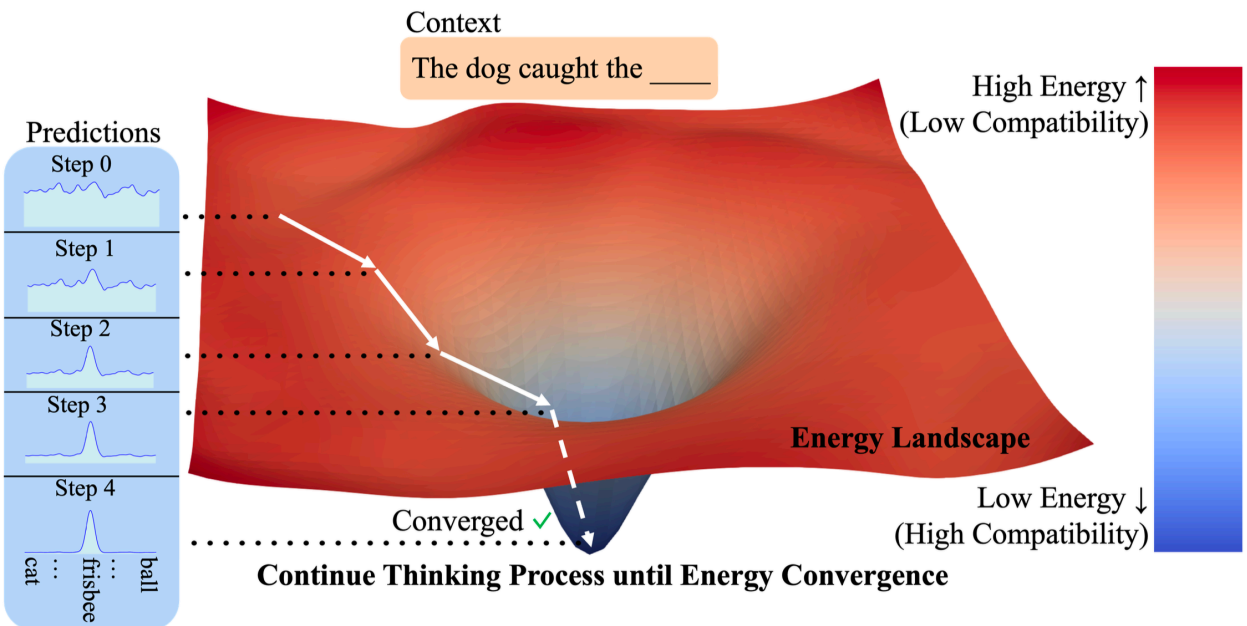


Figure 11: Ideal energy landscape for an EBT at inference time

EBT Training

Training Step:

Inputs: Context x, Target y, EBM $E_\theta(x, y)$

1 Sample $\hat{y}_0 \sim \mathcal{N}(0, I)$

2 **for** $i = 0, ..., M - 1$:

3 $\quad \hat{y}_{i+1} = \hat{y}_i - \alpha \nabla_{\hat{y}_i} E_\theta(x, \hat{y}_i) + \eta_i$

4 Compute loss: $\mathcal{L} = J(\hat{y}_N, y)$

5 Backprop through entire optimization

EBT Training

Training Step:

Inputs: Context x, Target y, EBM $E_\theta(x, y)$

1 Sample $\hat{y}_0 \sim \mathcal{N}(0, I)$

2 **for** $i = 0, ..., M-1$:

3 $\quad \hat{y}_{i+1} = \hat{y}_i - \alpha \nabla_{\hat{y}_i} E_\theta(x, \hat{y}_i) + \eta_i$

4 Compute loss: $\mathcal{L} = J(\hat{y}_N, y)$

5 Backprop through entire optimization

This requires second-order derivatives!

EBT Training

Training Step:

Inputs: Context x, Target y, EBM $E_\theta(x, y)$

Sample $\hat{y}_0 \sim \mathcal{N}(0, I)$

for $i = 0, ..., M-1$:

  $\hat{y}_{i+1} = \hat{y}_i - \alpha \nabla_{\hat{y}_i} E_\theta(x, \hat{y}_i) + \eta_i$

Compute loss: $\mathcal{L} = J(\hat{y}_N, y)$

Backprop through entire optimization

This requires second-order derivatives!

Making EBT training work:

1. **Langevin dynamics**: $\eta_i \sim \mathcal{N}(0, \sigma)$

EBT Training

Training Step:

Inputs: Context x, Target y, EBM $E_\theta(x, y)$

1 Sample $\hat{y}_0 \sim \mathcal{N}(0, I)$

2 **for** $i = 0, ..., M - 1$:

3 $\quad \hat{y}_{i+1} = \hat{y}_i - \alpha \nabla_{\hat{y}_i} E_\theta(x, \hat{y}_i) + \eta_i$

4 Compute loss: $\mathcal{L} = J(\hat{y}_N, y)$

5 Backprop through entire optimization

This requires second-order derivatives!

Making EBT training work:

1. **Langevin dynamics**: $\eta_i \sim \mathcal{N}(0, \sigma)$
2. **Random step size & steps**: α, M

EBT Training

Training Step:

Inputs: Context x, Target y, EBM $E_\theta(x, y)$

Sample $\hat{y}_0 \sim \mathcal{N}(0, I)$

for $i = 0, \ldots, M-1$:

  $\hat{y}_{i+1} = \hat{y}_i - \alpha \nabla_{\hat{y}_i} E_\theta(x, \hat{y}_i) + \eta_i$

Compute loss: $\mathcal{L} = J(\hat{y}_N, y)$

Backprop through entire optimization

This requires second-order derivatives!

Making EBT training work:

1. **Langevin dynamics**: $\eta_i \sim \mathcal{N}(0, \sigma)$
2. **Random step size & steps**: α, M
3. **Replay buffer:** reuse previous optimization trajectories $\rightarrow$ landscape well-defined near minima

EBT Inference

Inference:

Inputs: Context x, EBM $E_\theta(x, y)$,

for $j = 1, ..., N$:

  Sample $\hat{y}_{0,j} \sim \mathcal{N}(0, I)$

  for $i = 0, ..., N-1$:

    $\hat{y}_{i+1,j} = \hat{y}_{i,j} - \alpha \nabla_{\hat{y}_{i,j}} E_\theta\left(x, \hat{y}_{i,j}\right)$

return: $\hat{y}^* = \arg\min_j E_{\theta(x, \hat{y}_{N,j})}$

Same as training except we sample N initial points (BoN)

EBT Inference

Inference:

Inputs: Context x, EBM $E_\theta(x, y)$,

1 **for** $j = 1, ..., N$:

2 Sample $\hat{y}_{0,j} \sim \mathcal{N}(0, I)$

3 **for** $i = 0, ..., N-1$:

4 $\hat{y}_{i+1,j} = \hat{y}_{i,j} - \alpha \nabla_{\hat{y}_{i,j}} E_\theta(x, \hat{y}_{i,j})$

5 **return**: $\hat{y}^* = \arg\min_j E_{\theta(x, \hat{y}_{N,j})}$

Verification: Natural way to pick the best prediction based on the Energy $E_{\theta(x,\hat{y})}$

Same as training except we sample N initial points (BoN)

Energy-Based Transformers are Scalable Learners and Thinkers

Energy-Based Transformers are **Scalable Learners and Thinkers?**

Learning (Pretraining) Scalability: Language

EBTs **out-scale** Transformer++ (Touvron et al., 2023) across **six axes**[11] during pretraining:

[11] Parameters, training tokens, FLOPs, embedding dimension, batch size and number of transformer blocks

Learning (Pretraining) Scalability: Language

EBTs **out-scale** Transformer++ (Touvron et al., 2023) across **six axes**[12] during pretraining:

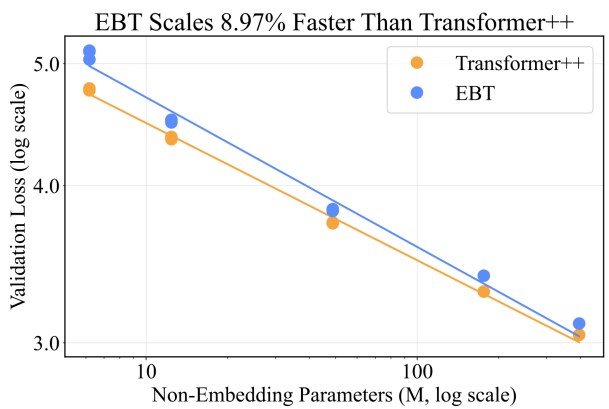


Figure 12: Parameter, token, and FLOPs scaling compared to Transformer++

[12] Parameters, training tokens, FLOPs, embedding dimension, batch size and number of transformer blocks

Learning (Pretraining) Scalability: Language

EBTs **out-scale** Transformer++ (Touvron et al., 2023) across **six axes**[13] during pretraining:

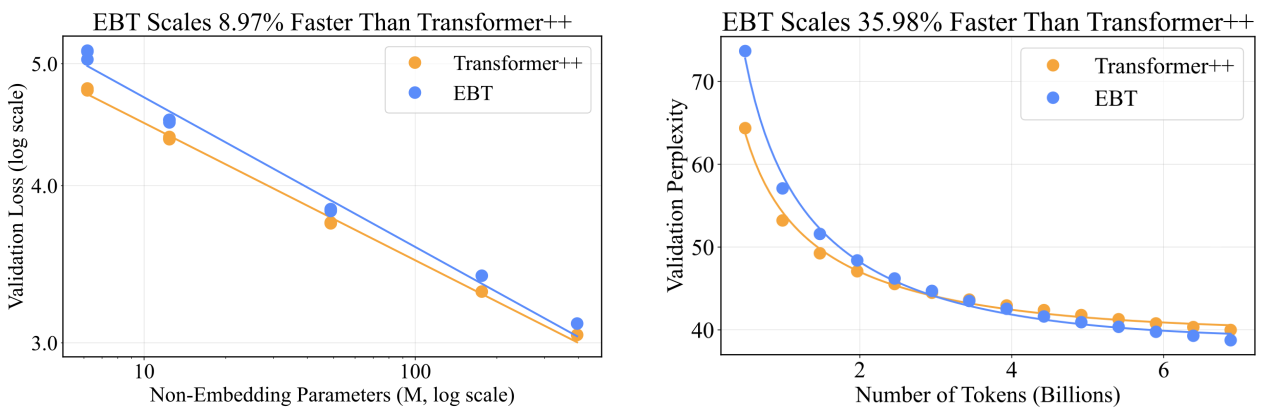


Figure 13: Parameter, token, and FLOPs scaling compared to Transformer++

[13] Parameters, training tokens, FLOPs, embedding dimension, batch size and number of transformer blocks

Learning (Pretraining) Scalability: Language

EBTs **out-scale** Transformer++ (Touvron et al., 2023) across **six axes**[14] during pretraining:

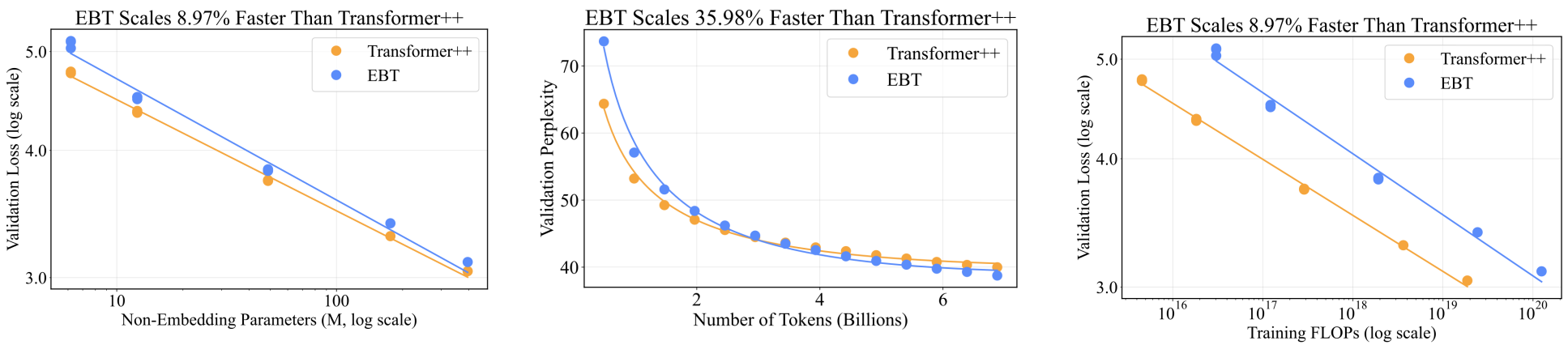


Figure 14: Parameter, token, and FLOPs scaling compared to Transformer++

[14] Parameters, training tokens, FLOPs, embedding dimension, batch size and number of transformer blocks

Learning (Pretraining) Scalability: Language

EBTs **out-scale** Transformer++ (Touvron et al., 2023) across **six axes**[15] during pretraining:

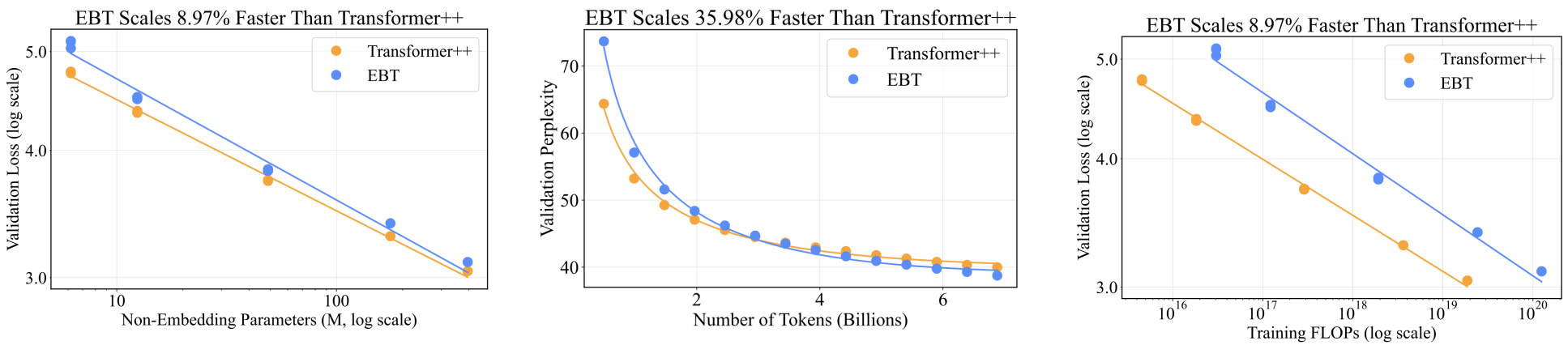


Figure 15: Parameter, token, and FLOPs scaling compared to Transformer++

[15] Parameters, training tokens, FLOPs, embedding dimension, batch size and number of transformer blocks

Calculating FLOPs

Transformer++ (Casson, 2023)[16]

$$\text{FLOPs} \approx \underbrace{2N}_{\text{Forward}} + \underbrace{4N}_{\text{Backward}} = 6N$$

[16] N = Number of non-embedding parameters

Calculating FLOPs

Transformer++ (Casson, 2023)[17]

$$\text{FLOPs} \approx \underbrace{2N}_{\text{Forward}} + \underbrace{4N}_{\text{Backward}} = 6N$$

EBT[18]

$$\text{FLOPs} \approx F + B + \underbrace{B}_{\text{HVP}} = 2N + 4N + \mathbf{4N} = 10N$$

[17] N = Number of non-embedding parameters

[18] Backprop through ∇E_θ needs a Hessian-vector product (HVP): Same theoretical complexity as a gradient (Pearlmutter, 1994).

Calculating FLOPs

Transformer++ (Casson, 2023)[19]

$$\text{FLOPs} \approx \underbrace{2N}_{\text{Forward}} + \underbrace{4N}_{\text{Backward}} = 6N$$

EBT[20]

$$\text{FLOPs} \approx F + B + \underbrace{B}_{\text{HVP}} = 2N + 4N + 4N = 10N \times \mathbf{2} = \mathbf{20}\boldsymbol{N}$$

[19] N = Number of non-embedding parameters

[20] Backprop through ∇E_θ needs a Hessian-vector product (HVP): Same theoretical complexity as a gradient (Pearlmutter, 1994).

Calculating FLOPs

Transformer++ (Casson, 2023)[21]

$$\text{FLOPs} \approx \underbrace{2N}_{\text{Forward}} + \underbrace{4N}_{\text{Backward}} = 6N$$

EBT[22]

$$\text{FLOPs} \approx F + B + \underbrace{B}_{\text{HVP}} = 2N + 4N + 4N = 10N \times 2 = \mathbf{20N}$$

For $M = 2$ optimization steps $\rightarrow 2 * 20N \approx 6.66 \times$ Transformer++.

[21] N = Number of non-embedding parameters

[22] Backprop through ∇E_θ needs a Hessian-vector product (HVP): Same theoretical complexity as a gradient (Pearlmutter, 1994).

Thinking Scalability: Language

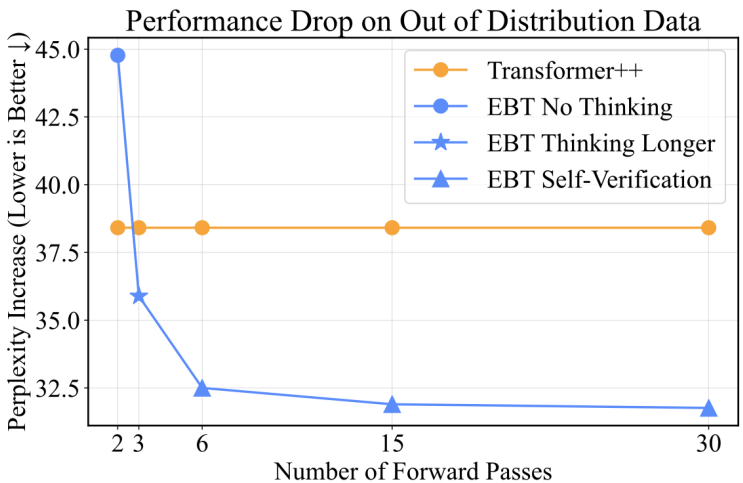


Figure 16: Generalization to Out-of-Distribution Data vs. Transformer++

- **Thinking Longer:** $\rightarrow$ More optimization steps M
- **Self-Verification:** $\rightarrow$ Optimizing many predictions and choosing the best. Best of N (BoN)

Thinking Scalability: Language

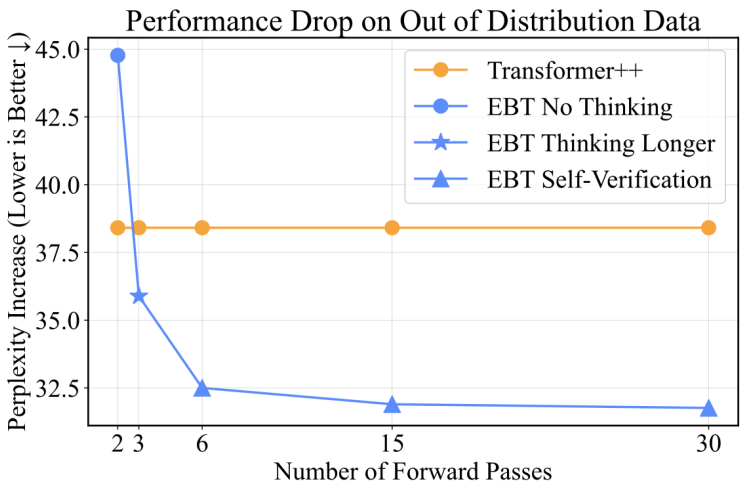


Figure 17: Generalization to Out-of-Distribution Data vs. Transformer++

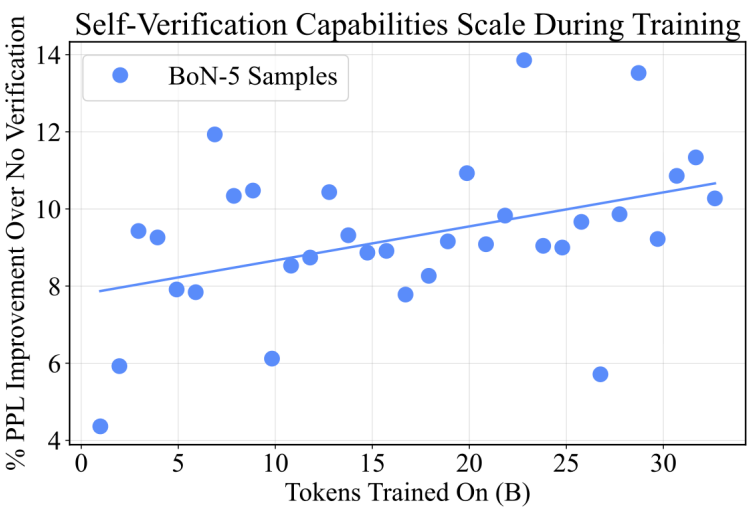
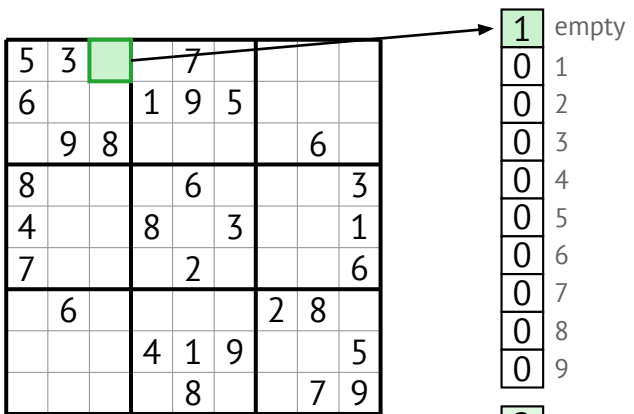


Figure 18: Verification vs. number of Tokens trained on

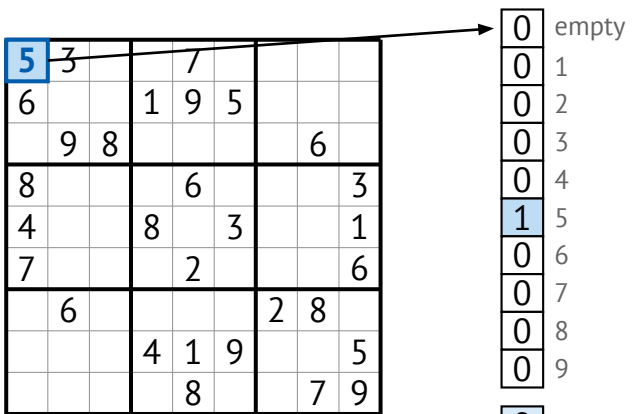
- **Thinking Longer:** $\rightarrow$ More optimization steps M
- **Self-Verification:** $\rightarrow$ Optimizing many predictions and choosing the best. Best of N (BoN)

Solving Sudoku



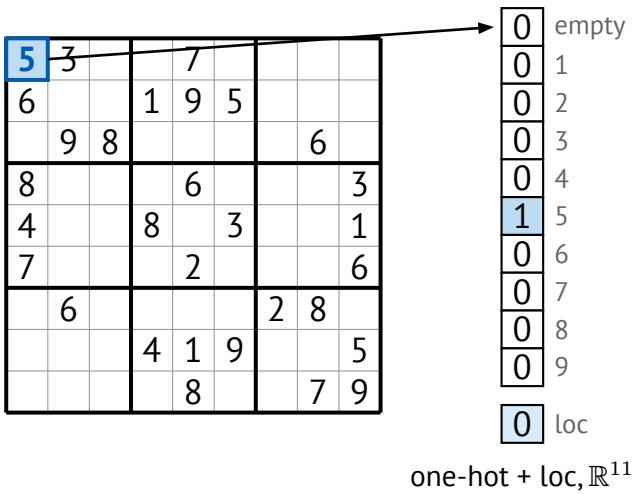
Each of the 81 cells is encoded as a vector in $\mathbb{R}^{11}$: a one-hot for the digit (index 0 = empty, 1–9 = digits) plus its position index (0–80).

Solving Sudoku



Each of the 81 cells is encoded as a vector in $\mathbb{R}^{11}$: a one-hot for the digit (index 0 = empty, 1–9 = digits) plus its position index (0–80).

Solving Sudoku



Each of the 81 cells is encoded as a vector in $\mathbb{R}^{11}$: a one-hot for the digit (index 0 = empty, 1–9 = digits) plus its position index (0–80).

Architecture	Accuracy
FF. Transformer	0.03%
TRM (RNN)[23]	17.7%
EBTs	**29.7%**

Table 1: Solving Sudoku with different architectures

[23] Tiny Recursive Model (TRM) (Jolicoeur-Martineau, 2025)

Generalization to Out-of-Distribution Data

Key finding: Thinking benefits scale with OOD shift: the further from training distribution, the more thinking helps.

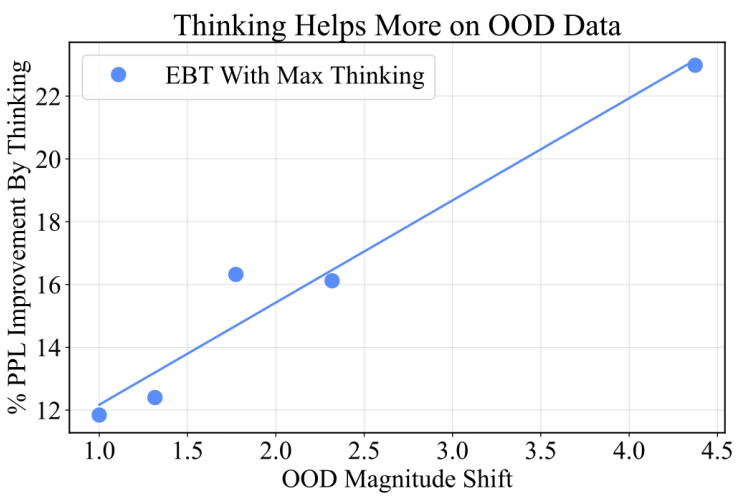


Figure 19: OOD thinking performance

EBT for Image Denoising

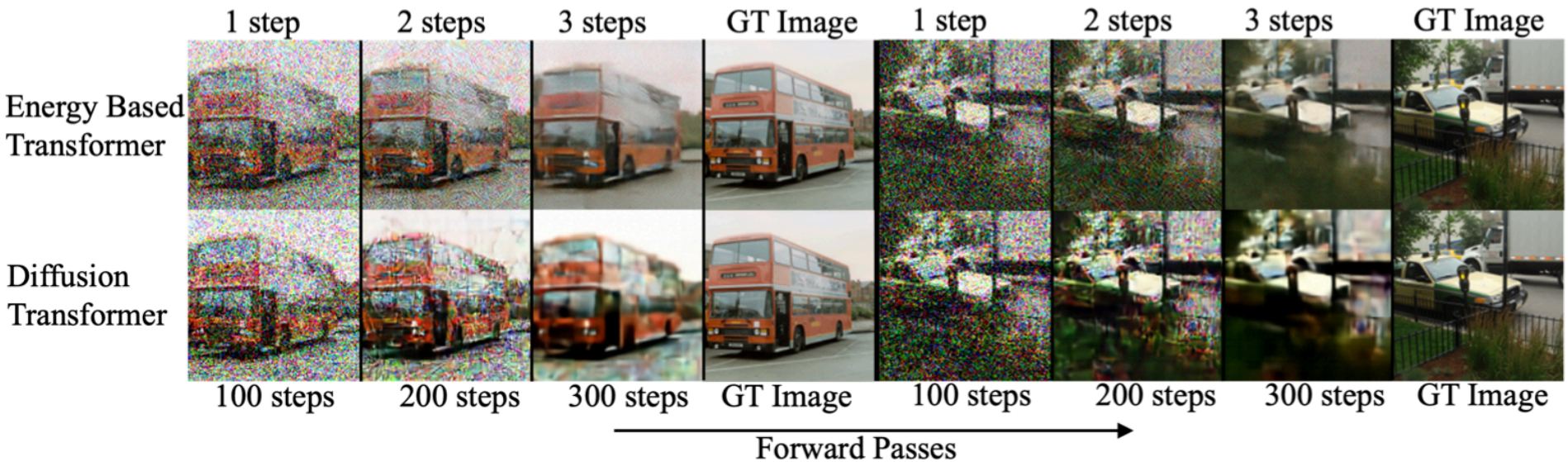


Figure 20: EBT image denoising example

Limitations & Conclusion

Limitations:

- Extra hyperparameters (α, M), training stability is sensitive
- Higher training/inference cost (**second-order gradients**)
- Only tested up to **800M** parameters (resource constraints)

Limitations & Conclusion

Limitations:

- Extra hyperparameters (α, M), training stability is sensitive
- Higher training/inference cost (**second-order gradients**)
- Only tested up to **800M** parameters (resource constraints)

Conclusion:

- EBTs **out-scale Transformer++** during pretraining
- System 2 Thinking emerges **from unsupervised learning alone** modality & problem agnostic
- Thinking benefits **scale with OOD shift**: harder problems gain more
- A interesting paradigm shift for future foundation models

My personal take on the paper

ICLR 2026 **Oral** Paper

—

- Inference delay/speed?
- Memory usage?
- Transformers are so heavily optimized nowadays. EBTs have to catch up a lot to be competitive.
- Will it actually scale?
- How meaningful is a couple of % when they fit to so few datapoints?
- EBTs can potentially be combined with other techniques like CoT

Questions

Bibliography

Casson, A. (2023,). *Transformer FLOPs*. https://adamcasson.com/posts/transformer-flops

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., & Kaiser, Ł. (2019,). *Universal Transformers*. https://arxiv.org/abs/1807.03819

Geiping, J., McLeish, S., Jain, N., Kirchenbauer, J., Singh, S., Bartoldson, B. R., Kailkhura, B., Bhatele, A., & Goldstein, T. (2025,). *Scaling up Test-Time Compute with Latent Reasoning: A Recurrent Depth Approach*. https://arxiv.org/abs/2502.05171

Guo, D., Yang, D., Zhang, H., Song, J., & others. (2025). DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *Arxiv Preprint Arxiv:2501.12948*.

Ho, J., Jain, A., & Abbeel, P. (2020,). *Denoising Diffusion Probabilistic Models*. https://arxiv.org/abs/2006.11239

Jolicoeur-Martineau, A. (2025,). *Less is More: Recursive Reasoning with Tiny Networks*. https://arxiv.org/abs/2510.04871

Kahneman, D. (2011). *Thinking, fast and slow*. Macmillan.

Pearlmutter, B. A. (1994). Fast Exact Multiplication by the Hessian. *Neural Computation*, *6*(1), 147–160.

Teh, Y. W., Welling, M., Osindero, S., & Hinton, G. E. (2003). Energy-based models for sparse overcomplete representations. *J. Mach. Learn. Res.*, *4*(null), 1235–1260.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., … Scialom, T. (2023,). *Llama 2: Open Foundation and Fine-Tuned Chat Models*. https://arxiv.org/abs/2307.09288

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., & Zhou, D. (2023,). *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. https://arxiv.org/abs/2201.11903

Appendix

System 2 Thinking Ablations vs. Regularization

Thinking Longer: $\rightarrow$ More optimization steps

Self-Verification: $\rightarrow$ Optimizing many predictions and choosing the best

Model	Thinking Longer ↑	Thinking Longer and Self-Verification ↑
No Random Step Size	-1.47	0.19
No Random Num. Steps	0.00	9.65
No Langevin Dynamics	**17.2**	17.0
No Replay Buffer	14.8	17.8
Full System 2 Configuration	7.19	**18.7**

Figure 21: System 2 Thinking performance measured by percent perplexity improvement

Language Model Task Generalization

Model	Pretrain	GSM8K ↓	SQuAD ↓	BB Math QA ↓	BB Dyck ↓
Transformer++	**31.36**	49.6	**52.3**	79.8	131.5
EBT	33.43	**43.3**	53.1	**72.6**	**125.3**

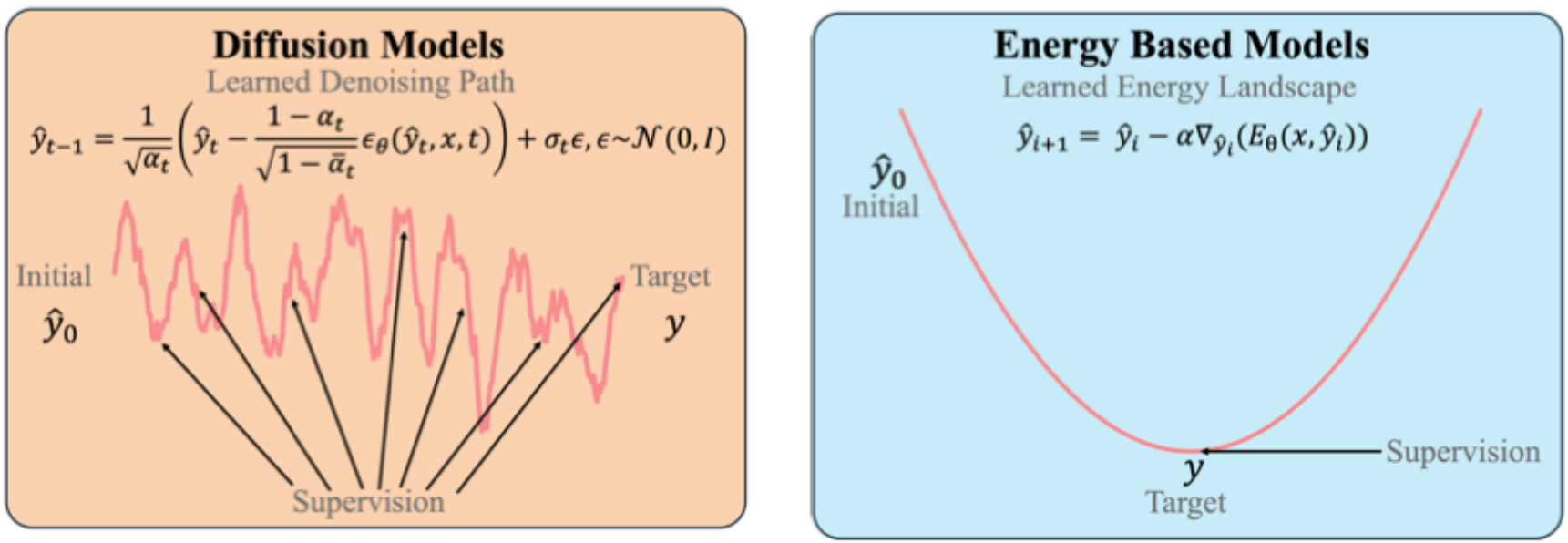
Figure 22: Generalization to Out-of-Distribution Data compared to Transformer++ (Touvron et al., 2023)

Connection to Diffusion Models & RNNs

Diffusion and RNNs can be considered as "implicit" EBMs that learn $M_{\theta(x,\hat{y})}$.

$$\hat{y}_{i+1} = \hat{y}_i - \alpha \underbrace{\nabla_{\hat{y}_i} E_\theta(x, \hat{y}_i)}_{M_{\theta(x,\hat{y}_i)}}$$

Connection to Diffusion Models & RNNs



(a) Diffusion Model

(b) Energy-Based Model (EBM)

Figure 23: Connection to RNNs