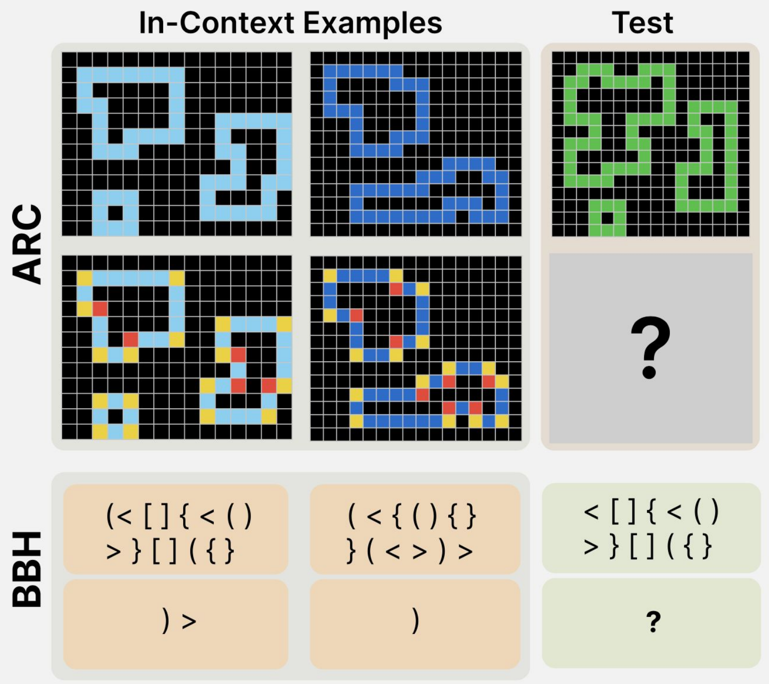
CS 957, System-2 AI
Test-Time Training

Mohammad Hossein Rohban | Apr. 2025

Sharif University of Technology

Motivation

- Some tasks that appear at the test time are overly novel.
- e.g. ARC and BBH
- ICL does not work.



Test Time Computation

Previously seen test-time scaling (TTS) helps with system-2 tasks in LLMs.

- CoT
- Self-Consistency
- BoN
- ...

Issues:

-Sometimes it is not multi-step reasoning;

-Solution diversity is not easy to come by;

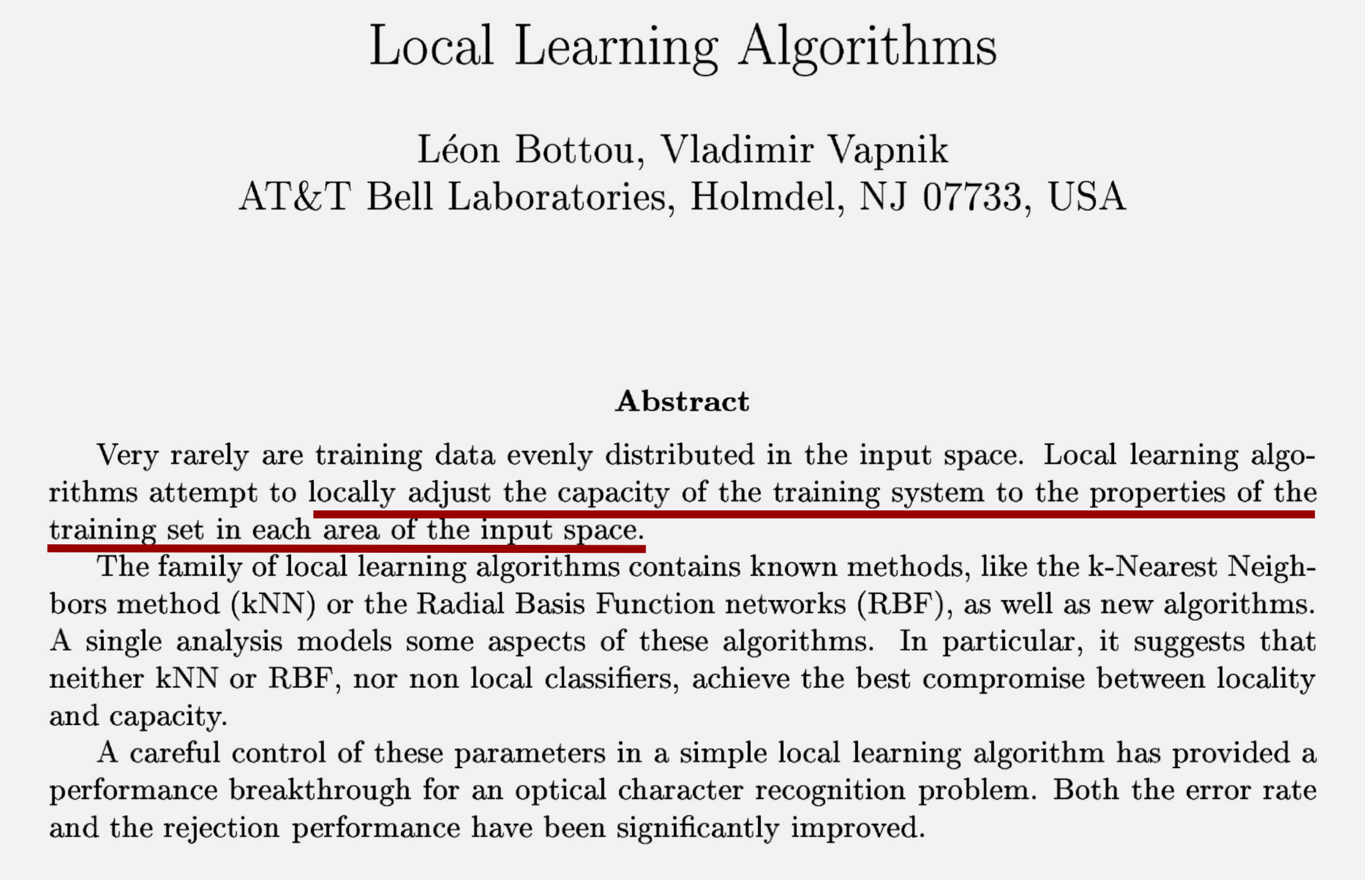
-Verifier is not accessible to trainable due to small sample size.

Test Time Training

How about training the model based on the test data, without its label of course?

Consider it as a form of adapting the model towards solving a specific task.

Local Learning; Bottou, Vapnik 1992



Local Learning Algorithms

Léon Bottou, Vladimir Vapnik
AT&T Bell Laboratories, Holmdel, NJ 07733, USA

Abstract

Very rarely are training data evenly distributed in the input space. Local learning algorithms attempt to locally adjust the capacity of the training system to the properties of the training set in each area of the input space.

The family of local learning algorithms contains known methods, like the k-Nearest Neighbors method (kNN) or the Radial Basis Function networks (RBF), as well as new algorithms. A single analysis models some aspects of these algorithms. In particular, it suggests that neither kNN or RBF, nor non local classifiers, achieve the best compromise between locality and capacity.

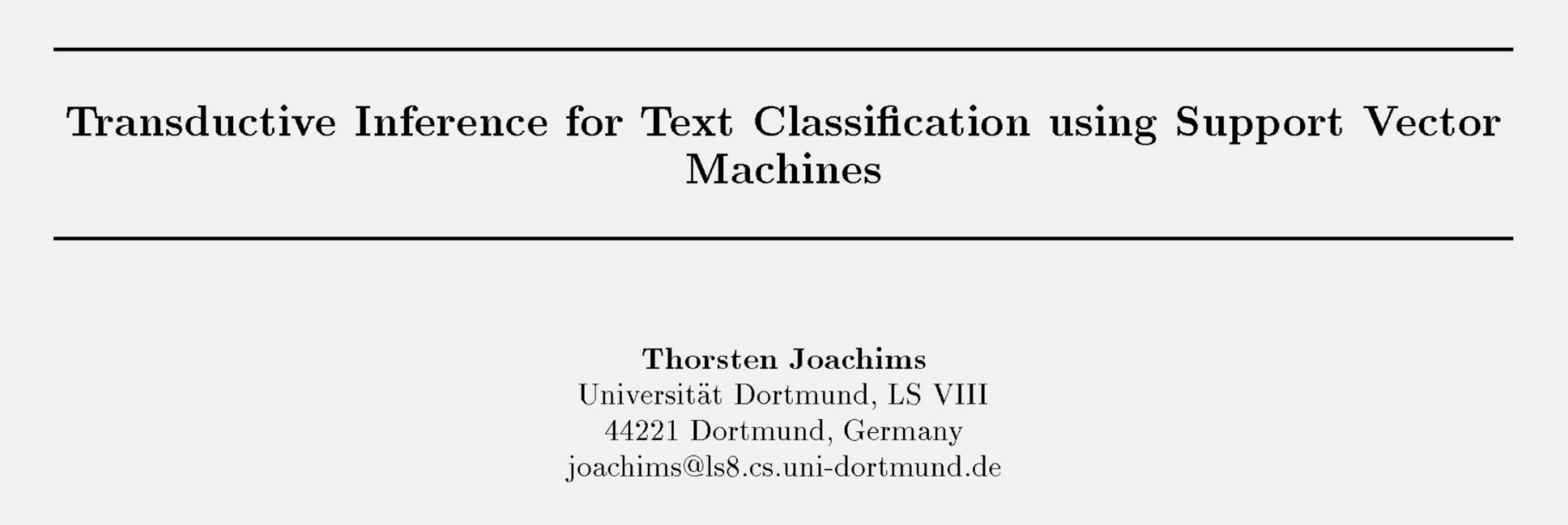
A careful control of these parameters in a simple local learning algorithm has provided a performance breakthrough for an optical character recognition problem. Both the error rate and the rejection performance have been significantly improved.

Local Learning; Bottou, Vapnik 1992

$$w^*(\gamma) \;=\; \operatorname*{Arg\,Min}_{w \in W_\gamma} \; \frac{1}{l} \sum_{i=1}^{l} \; J(y_i, f_w(x_i))$$

$$w^*(x_0, b, \gamma) \;=\; \operatorname*{Arg\,Min}_{w \in W_\gamma} \; \frac{1}{l} \sum_{i=1}^{l} \; K(x_i - x_0, b) \; J(y_i, f_w(x_i))$$

Transductive Learning; Joachims 1999



Transductive Inference for Text Classification using Support Vector Machines

Thorsten Joachims
Universität Dortmund, LS VIII
44221 Dortmund, Germany
joachims@ls8.cs.uni-dortmund.de

Transductive Learning; Joachims 1999

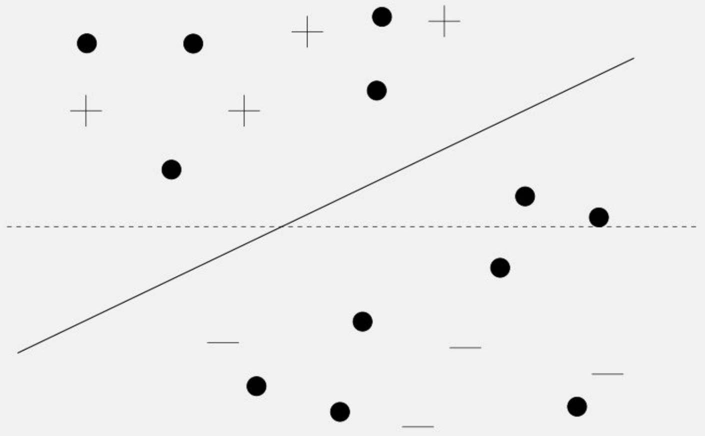
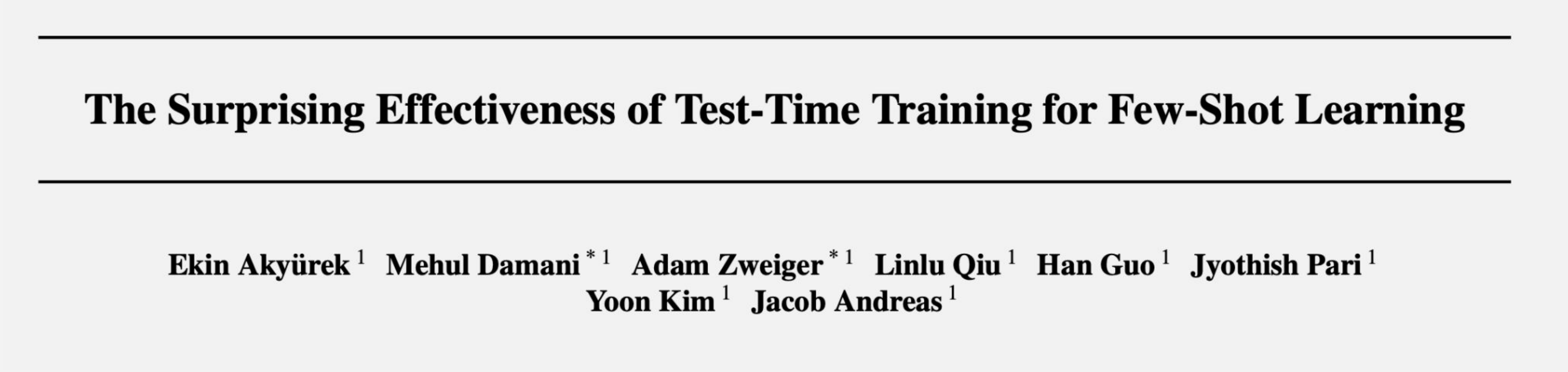


Figure 2: The maximum margin hyperplanes. Positive/negative examples are marked as +/−, test examples as dots. The dashed line is the solution of the inductive SVM. The solid line shows the transductive classification.

Test-time Training for LLMs

Starting with initial model parameters θ_0, for each test input (or batch of inputs) d, we generate a temporary training dataset D_{TTT}.

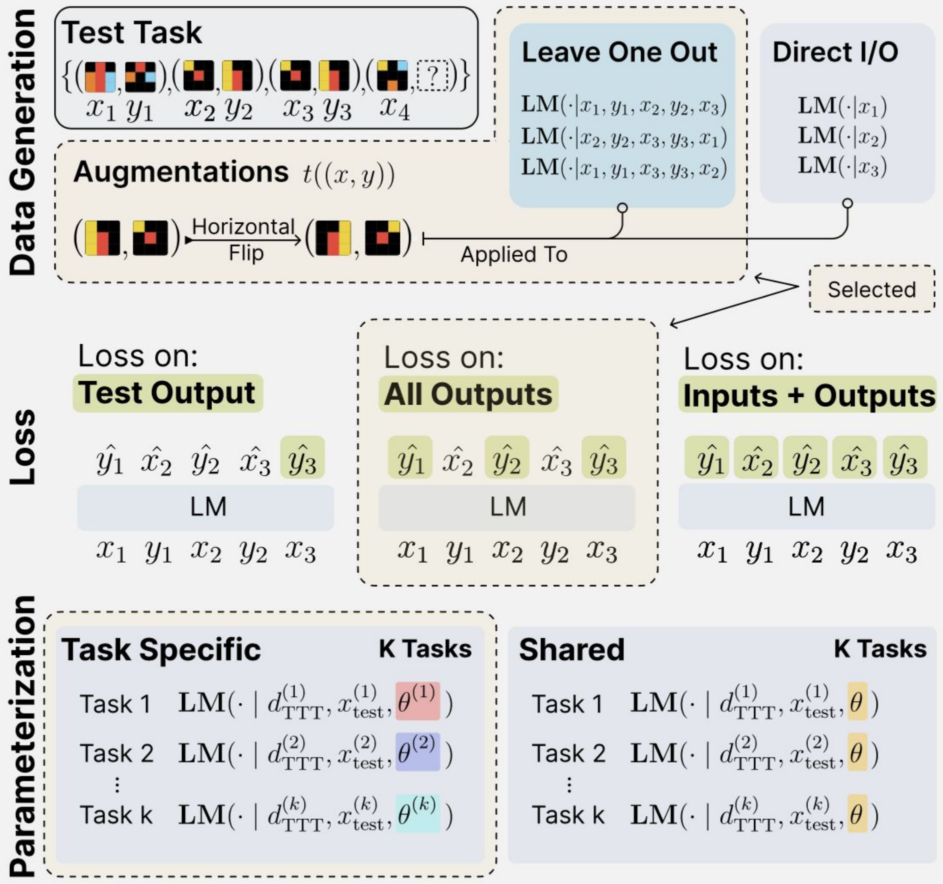
$$\arg\min_{\boldsymbol{\theta}} \sum_{d_{\text{TTT}} \in \mathcal{D}_{\text{TTT}}} \mathcal{L}(\text{LM}(d_{\text{TTT}}; \boldsymbol{\theta})),$$



The Surprising Effectiveness of Test-Time Training for Few-Shot Learning

Ekin Akyürek [1] Mehul Damani [*1] Adam Zweiger [*1] Linlu Qiu [1] Han Guo [1] Jyothish Pari [1]
Yoon Kim [1] Jacob Andreas [1]

General Scheme


Data Generation
Test Task
$x_1\ y_1\ x_2\ y_2\ x_3\ y_3\ x_4$
Leave One Out
$\mathrm{LM}(\cdot|x_1,y_1,x_2,y_2,x_3)$
$\mathrm{LM}(\cdot|x_2,y_2,x_3,y_3,x_1)$
$\mathrm{LM}(\cdot|x_1,y_1,x_3,y_3,x_2)$
Direct I/O
$\mathrm{LM}(\cdot|x_1)$
$\mathrm{LM}(\cdot|x_2)$
$\mathrm{LM}(\cdot|x_3)$
Augmentations $t((x,y))$
Horizontal Flip
Applied To
Selected
Loss
Loss on: Test Output
$\hat{y}_1\ \hat{x}_2\ \hat{y}_2\ \hat{x}_3\ \hat{y}_3$
LM
$x_1\ y_1\ x_2\ y_2\ x_3$
Loss on: All Outputs
$\hat{y}_1\ \hat{x}_2\ \hat{y}_2\ \hat{x}_3\ \hat{y}_3$
LM
$x_1\ y_1\ x_2\ y_2\ x_3$
Loss on: Inputs + Outputs
$\hat{y}_1\ \hat{x}_2\ \hat{y}_2\ \hat{x}_3\ \hat{y}_3$
LM
$x_1\ y_1\ x_2\ y_2\ x_3$
Parameterization
Task Specific
K Tasks
Task 1 $\mathrm{LM}(\cdot \mid d^{(1)}_{\mathrm{TTT}}, x^{(1)}_{\mathrm{test}}, \theta^{(1)})$
Task 2 $\mathrm{LM}(\cdot \mid d^{(2)}_{\mathrm{TTT}}, x^{(2)}_{\mathrm{test}}, \theta^{(2)})$
⋮
Task k $\mathrm{LM}(\cdot \mid d^{(k)}_{\mathrm{TTT}}, x^{(k)}_{\mathrm{test}}, \theta^{(k)})$
Shared
K Tasks
Task 1 $\mathrm{LM}(\cdot \mid d^{(1)}_{\mathrm{TTT}}, x^{(1)}_{\mathrm{test}}, \theta)$
Task 2 $\mathrm{LM}(\cdot \mid d^{(2)}_{\mathrm{TTT}}, x^{(2)}_{\mathrm{test}}, \theta)$
⋮
Task k $\mathrm{LM}(\cdot \mid d^{(k)}_{\mathrm{TTT}}, x^{(k)}_{\mathrm{test}}, \theta)$

Dataset Creation

Leave-one-out tasks

$$d_j^{\text{ICL}} = \Big(\{(x_k, y_k)\}_{k \neq j},\, x_j,\, y_j\Big).$$

Direct input-output (I/O) tasks

$$d_j^{\text{I/O}} = (x_j, y_j)\,.$$

Data Augmentation

Let T be a set of invertible transformations. For each t∈T , we have $t^{-1}(t(x)) = x$, so we can apply t to each training and test instance in d_j to yield a transformed task $t(d_j)$.

$$\mathcal{D}_{\mathrm{TTT}} = \bigcup_{t\in\mathcal{T}} \bigcup_j t(d_j).$$

Loss Functions

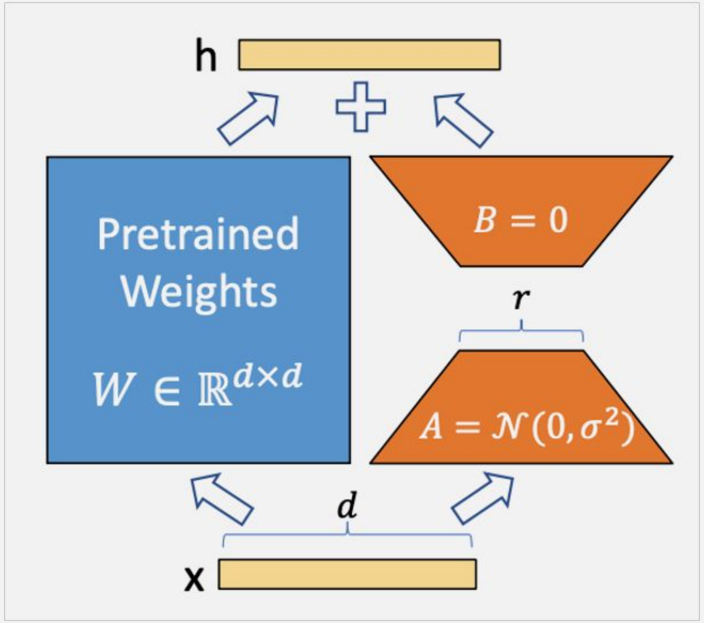
$$\mathcal{L}_{\mathrm{LM}}^{\mathrm{label}} = \mathcal{L}_{\mathrm{LM}}(y_{\mathrm{test}} \mid x_1, y_1, \ldots, x_K, y_K, x_{\mathrm{test}}; \boldsymbol{\theta})$$

$$\mathcal{L}_{\mathrm{LM}}^{\mathrm{outputs}} = \mathcal{L}_{\mathrm{LM}}^{\mathrm{label}} + \sum_{k=1}^{K} \mathcal{L}_{\mathrm{LM}}(y_k | x_1, y_1, ..., x_k; \boldsymbol{\theta})$$

$$\mathcal{L}_{\mathrm{LM}}^{\mathrm{all}} = \mathcal{L}_{\mathrm{LM}}^{\mathrm{outputs}} + \sum_{k=1}^{K} \mathcal{L}_{\mathrm{LM}}(x_k | x_1, y_1, \ldots, y_{k-1}; \boldsymbol{\theta})$$

Training

- Learn task-specific LoRA adapters for each ARC or BBH task at test-time.
- K different LoRA adapters, where K is the number of test tasks.


h
Pretrained Weights
$W \in \mathbb{R}^{d \times d}$
$B = 0$
r
$A = \mathcal{N}(0, \sigma^2)$
d
x

Fine-tuning before TTT

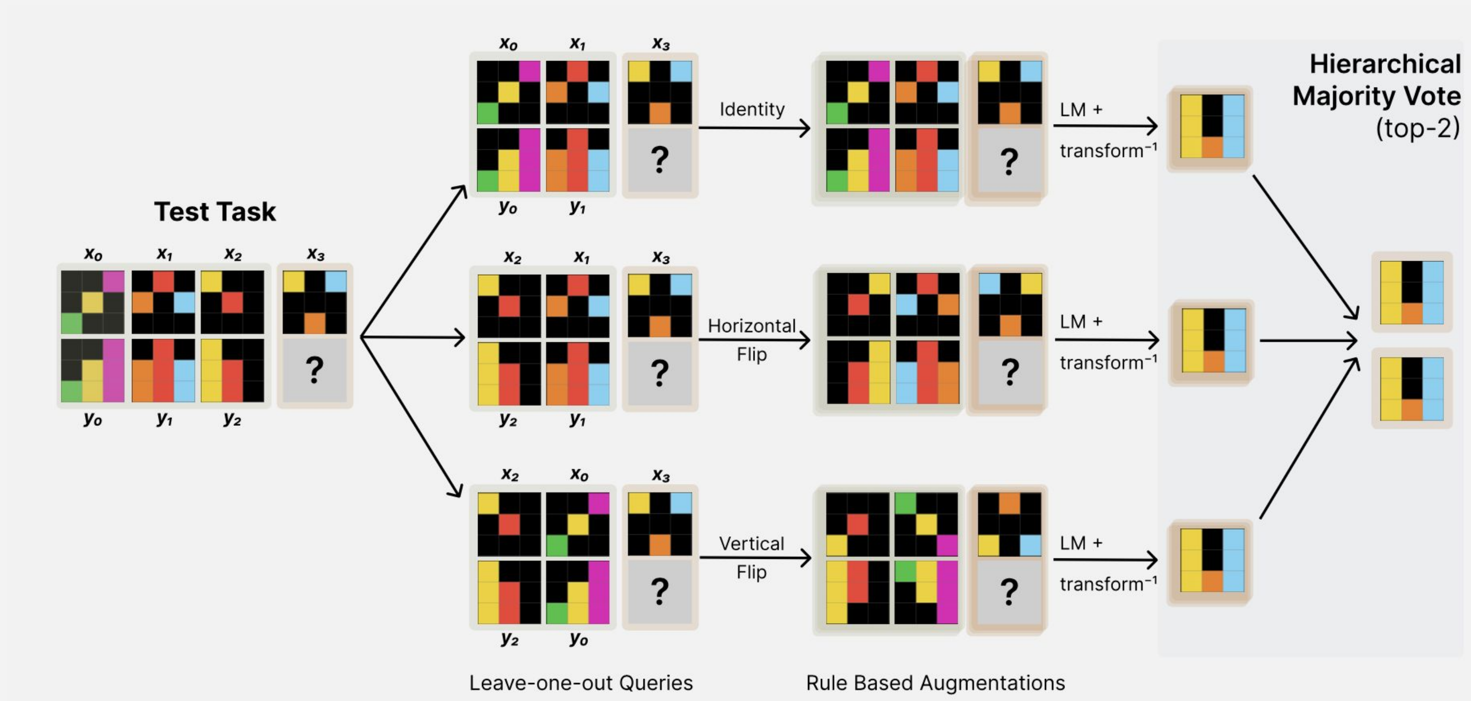
- Initial capabilities of the base model significantly influence its final performance.
- Used fine-tuning based on synthetic training data to enhance the base model's abstract reasoning capabilities.

Inference

- How can we get a “self-consistency” type boost of performance?
- Need to get multiple outputs.
- Can it be done through temperature scaling?

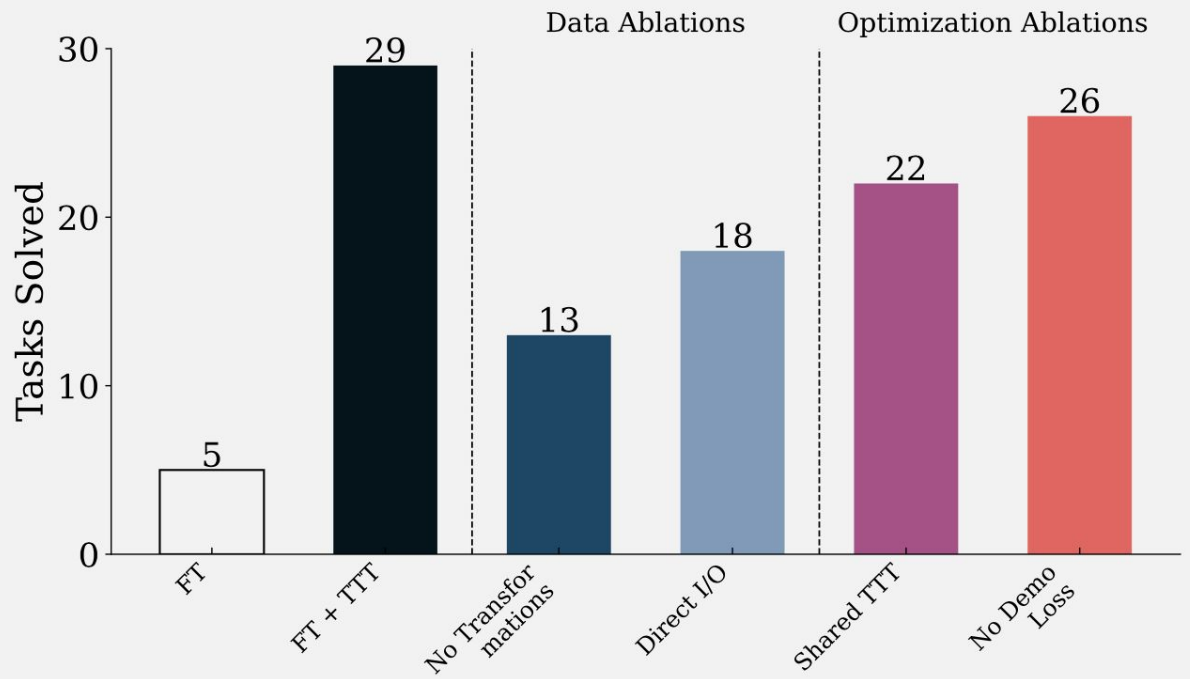
Inference

Use geometrical transformations.

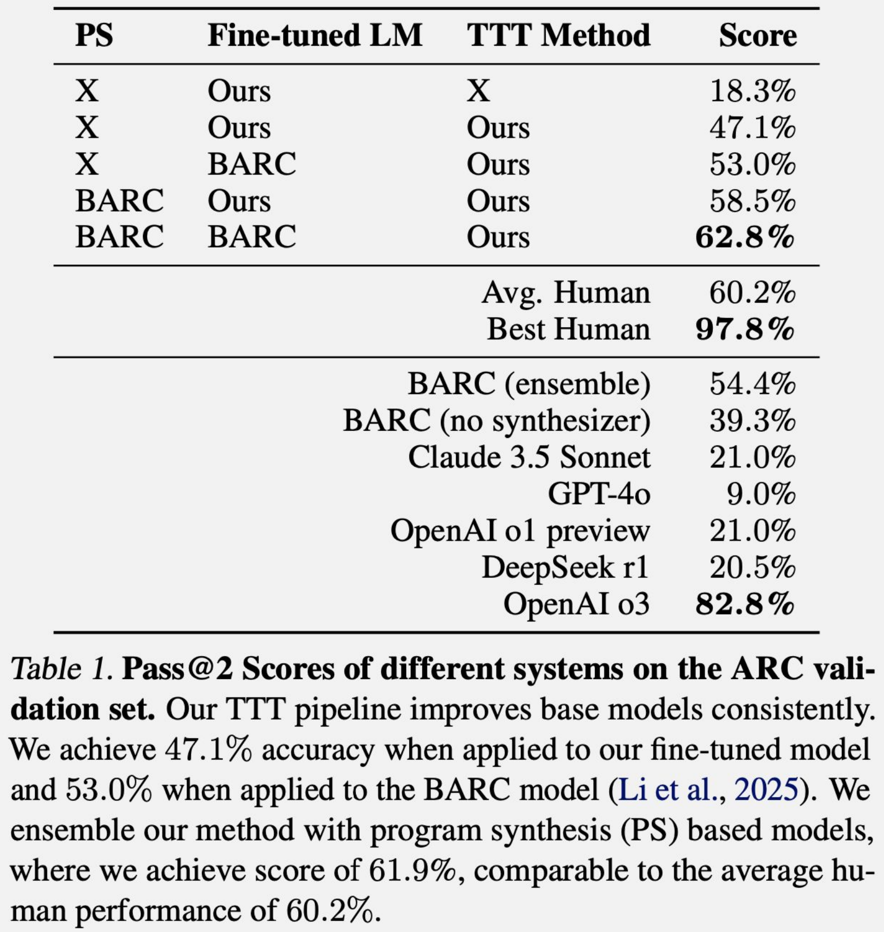


Results

LLaMA 3.2 1B as the base model



Results



PS	Fine-tuned LM	TTT Method	Score
X	Ours	X	18.3%
X	Ours	Ours	47.1%
X	BARC	Ours	53.0%
BARC	Ours	Ours	58.5%
BARC	BARC	Ours	**62.8%**
		Avg. Human	60.2%
		Best Human	**97.8%**
		BARC (ensemble)	54.4%
		BARC (no synthesizer)	39.3%
		Claude 3.5 Sonnet	21.0%
		GPT-4o	9.0%
		OpenAI o1 preview	21.0%
		DeepSeek r1	20.5%
		OpenAI o3	**82.8%**

Table 1. **Pass@2 Scores of different systems on the ARC validation set.** Our TTT pipeline improves base models consistently. We achieve 47.1% accuracy when applied to our fine-tuned model and 53.0% when applied to the BARC model (Li et al., 2025). We ensemble our method with program synthesis (PS) based models, where we achieve score of 61.9%, comparable to the average human performance of 60.2%.

Results on Semi-Private ARC

- Semi-private evaluation ARC-AGI challenge provides a hidden “semi-private dataset”.
- Used to rank submissions.
- The ensemble solution reaches 47.5% accuracy.

Big Bench Hard

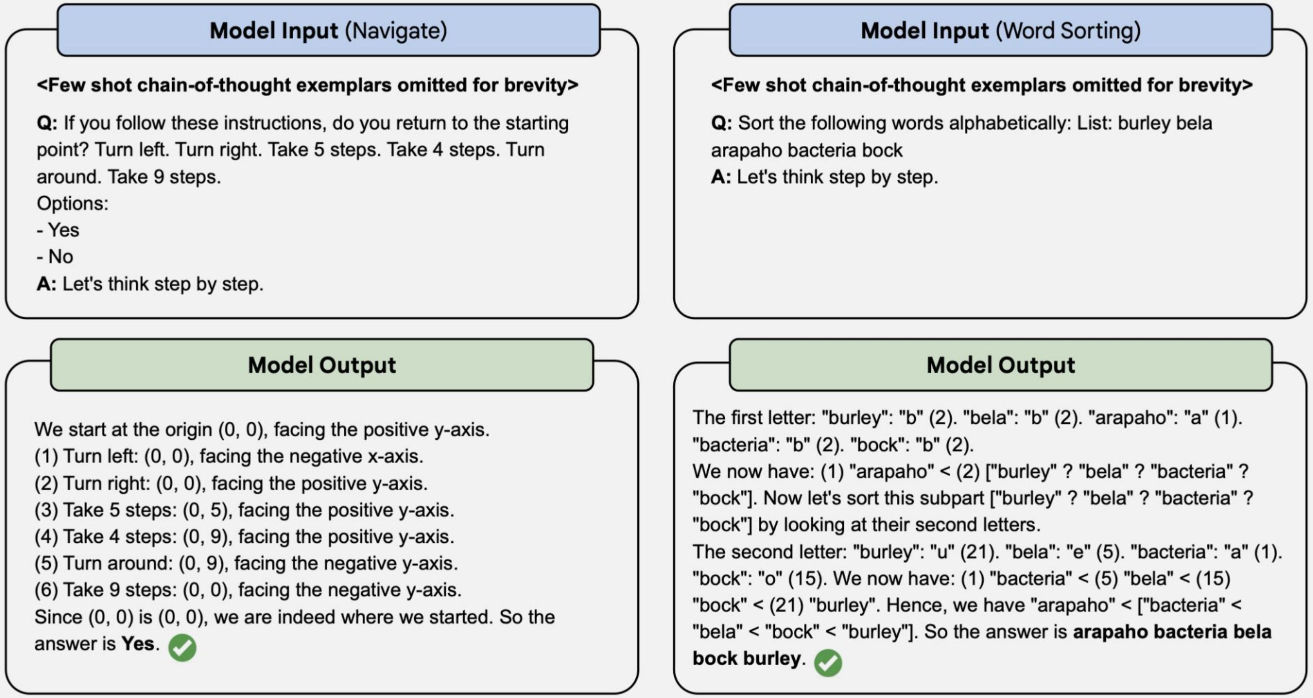
- A suite of 23 challenging BIG-Bench tasks which we call BIG-Bench Hard (BBH).
- Tasks for which prior language model evaluations did not outperform the average human-rater.

Challenging BIG-Bench tasks and whether chain-of-thought can solve them

Mirac Suzgun[π] Nathan Scales Nathanael Schärli Sebastian Gehrmann
Yi Tay Hyung Won Chung Aakanksha Chowdhery Quoc V. Le
Ed H. Chi Denny Zhou Jason Wei

Google Research [π]Stanford University

Results on BIG-Bench Hard


Model Input (Navigate)

<Few shot chain-of-thought exemplars omitted for brevity>

Q: If you follow these instructions, do you return to the starting point? Turn left. Turn right. Take 5 steps. Take 4 steps. Turn around. Take 9 steps.
Options:
- Yes
- No

A: Let's think step by step.

Model Output

We start at the origin (0, 0), facing the positive y-axis.
(1) Turn left: (0, 0), facing the negative x-axis.
(2) Turn right: (0, 0), facing the positive y-axis.
(3) Take 5 steps: (0, 5), facing the positive y-axis.
(4) Take 4 steps: (0, 9), facing the positive y-axis.
(5) Turn around: (0, 9), facing the negative y-axis.
(6) Take 9 steps: (0, 0), facing the negative y-axis.
Since (0, 0) is (0, 0), we are indeed where we started. So the answer is **Yes**. ✓

Model Input (Word Sorting)

<Few shot chain-of-thought exemplars omitted for brevity>

Q: Sort the following words alphabetically: List: burley bela arapaho bacteria bock
A: Let's think step by step.

Model Output

The first letter: "burley": "b" (2). "bela": "b" (2). "arapaho": "a" (1). "bacteria": "b" (2). "bock": "b" (2).
We now have: (1) "arapaho" < (2) ["burley" ? "bela" ? "bacteria" ? "bock"]. Now let's sort this subpart ["burley" ? "bela" ? "bacteria" ? "bock"] by looking at their second letters.
The second letter: "burley": "u" (21). "bela": "e" (5). "bacteria": "a" (1). "bock": "o" (15). We now have: (1) "bacteria" < (5) "bela" < (15) "bock" < (21) "burley". Hence, we have "arapaho" < ["bacteria" < "bela" < "bock" < "burley"]. So the answer is **arapaho bacteria bela bock burley**. ✓

Figure 2: Examples of two tasks—*Navigate* (left) and *Word Sorting* (right)—from BBH with real model outputs.

Evaluation on BBH

- 27 Tasks
- Each has 10 in-context examples given at the test time
- Used LLaMA 3.2 8B without any FT as the base
- LoRA of rank 64 over 40 randomly permuted demonstrations.

Results

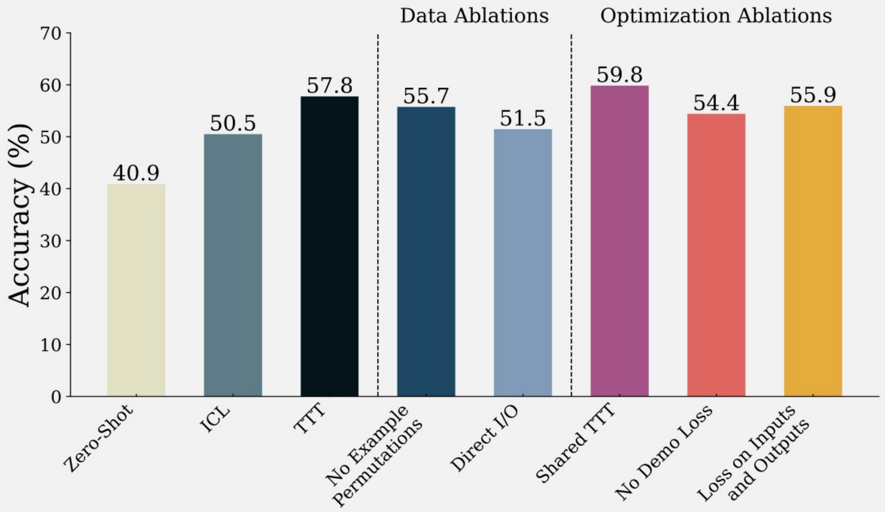


Figure 8. **Overall BIG-Bench Hard Results.** TTT outperforms standard in-context learning by 7.3 absolute percentage points, from 50.5% to 57.8%. Our performance improvement over direct input-output data shows that using in-context leave-one-out tasks is crucial. Not taking demonstration loss or taking loss on inputs results in a performance decrease. Unlike with ARC, using a shared adapter across all tasks improves performance.

Task Specific Analysis of BBH

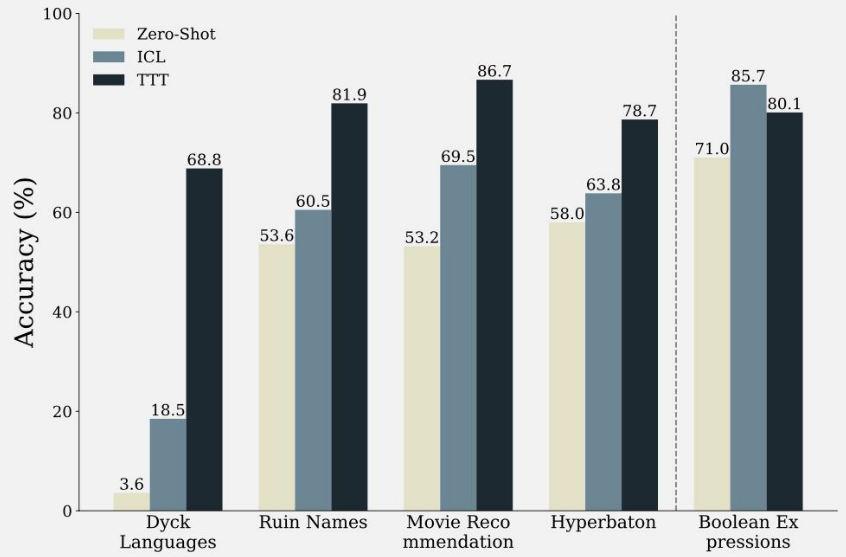
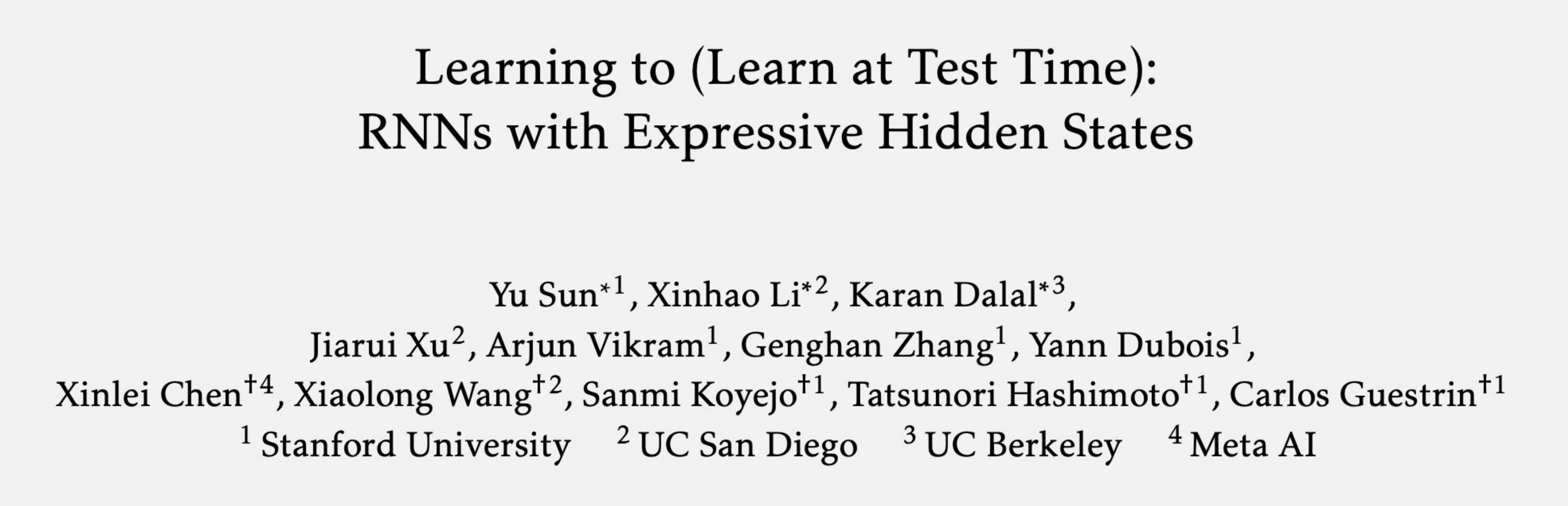


Figure 9. **BIG-Bench Hard results for tasks with the largest TTT-ICL score differences.** The four tasks on the left show the most significant improvements with TTT over ICL, while the task on the right has the lowest TTT score relative to ICL. Full task-specific results are given in Appendix F.2.

TTT for Making RNNs with Expressive Hidden States

- Self attention takes quadratic memory for the attention matrix.
- RNNs don’t have enough capacity to store the entire context in fixed length vec.
- Can TTT help in making the representation more flexible?



Learning to (Learn at Test Time):
RNNs with Expressive Hidden States

Yu Sun[*1], Xinhao Li[*2], Karan Dalal[*3],
Jiarui Xu[2], Arjun Vikram[1], Genghan Zhang[1], Yann Dubois[1],
Xinlei Chen[†4], Xiaolong Wang[†2], Sanmi Koyejo[†1], Tatsunori Hashimoto[†1], Carlos Guestrin[†1]
[1] Stanford University [2] UC San Diego [3] UC Berkeley [4] Meta AI

Overview

$$\begin{tikzcd}
\text{Output tokens} & & z_1 & & z_{t-1} & z_t = f(x_t; W_t) & \text{Output rule} \\
\text{Hidden state} & W_0 \arrow[r] & W_1 \arrow[r] \arrow[u] & \dots \arrow[r] & W_{t-1} \arrow[r] \arrow[u] & W_t = W_{t-1} - \eta \nabla \ell(W_{t-1}; x_t) \arrow[u] & \\
\text{Input tokens} & & x_1 \arrow[u] & & x_{t-1} \arrow[u] & x_t \arrow[u] & \text{Update rule}
\end{tikzcd}$$

Figure 1. All sequence modeling layers can be expressed as a hidden state that transitions according to an update rule. Our key idea is to make the hidden state itself a model f with weights W, and the update rule a gradient step on the self-supervised loss ℓ. Therefore, updating the hidden state on a test sequence is equivalent to training the model f at test time. This process, known as Test-Time Training (TTT), is programmed into our TTT layers.

Unifying Perspective

$$\begin{tikzcd}
\text{Output tokens} & & z_1 & & z_{t-1} & z_t = \texttt{output}(s_t, x_t) & \text{Output rule} \\
\text{Hidden state} & s_0 \arrow[r] & s_1 \arrow[u] \arrow[r] & \dots \arrow[r] & s_{t-1} \arrow[u] \arrow[r] & s_t = \texttt{update}(s_{t-1}, x_t) \arrow[u] & \text{Update rule} \\
\text{Input tokens} & & x_1 \arrow[u] & & x_{t-1} \arrow[u] & x_t \arrow[u] &
\end{tikzcd}$$

	Initial state	**Update rule**	**Output rule**	**Cost**
Naive RNN	$s_0 = \texttt{vector()}$	$s_t = \sigma(\theta_{ss} s_{t-1} + \theta_{sx} x_t)$	$z_t = \theta_{zs} s_t + \theta_{zx} x_t$	$O(1)$
Self-attention	$s_0 = \texttt{list()}$	$s_t = s_{t-1}.\text{append}(k_t, v_t)$	$z_t = V_t \texttt{softmax}\left(K_t^T q_t\right)$	$O(t)$
Naive TTT	$W_0 = f.\texttt{params()}$	$W_t = W_{t-1} - \eta \nabla \ell(W_{t-1}; x_t)$	$z_t = f(x_t; W_t)$	$O(1)$

Two Loops of Training

Inner loop: Optimize W (or context embedding) to minimize a reconstruction loss

$$\ell(W; x_t) = \|f(\tilde{x}_t; W) - x_t\|^2. \qquad \ell(W; x_t) = \left\|f(\theta_K x_t; W) - \theta_V x_t\right\|^2.$$

Outer loop: Optimize model parameters (thetas), such that the resulting W from the inner loop produces the desired output.

$$z_t = f\left(\theta_Q x_t; W_t\right).$$

Why doesn’t this collapse?

```python
class TTT_Layer(nn.Module):
  def __init__(self):
    self.task = Task()

  def forward(self, in_seq):
    state = Learner(self.task)
    out_seq = []
    for tok in in_seq:
      state.train(tok)
      out_seq.append(state.predict(tok))
    return out_seq

class Task(nn.Module):
  def __init__(self):
    self.theta_K = nn.Param((d1, d2))
    self.theta_V = nn.Param((d1, d2))
    self.theta_Q = nn.Param((d1, d2))

  def loss(self, f, x):
    train_view = self.theta_K @ x
    label_view = self.theta_V @ x
    return MSE(f(train_view), label_view)
```

```python
class Learner():
  def __init__(self, task):
    self.task = task
    # Linear here, but can be any model
    self.model = Linear()
    # online GD here for simplicity
    self.optim = OGD()

  def train(self, x):
    # grad function wrt first arg
    # of loss, which is self.model
    grad_fn = grad(self.task.loss)
    # calculate inner-loop grad
    grad_in = grad_fn(self.model, x)

    # starting from current params,
    # step in direction of grad_in,
    self.optim.step(self.model, grad_in)

  def predict(self, x):
    test_view = self.task.theta_Q @ x
    return self.model(test_view)
```

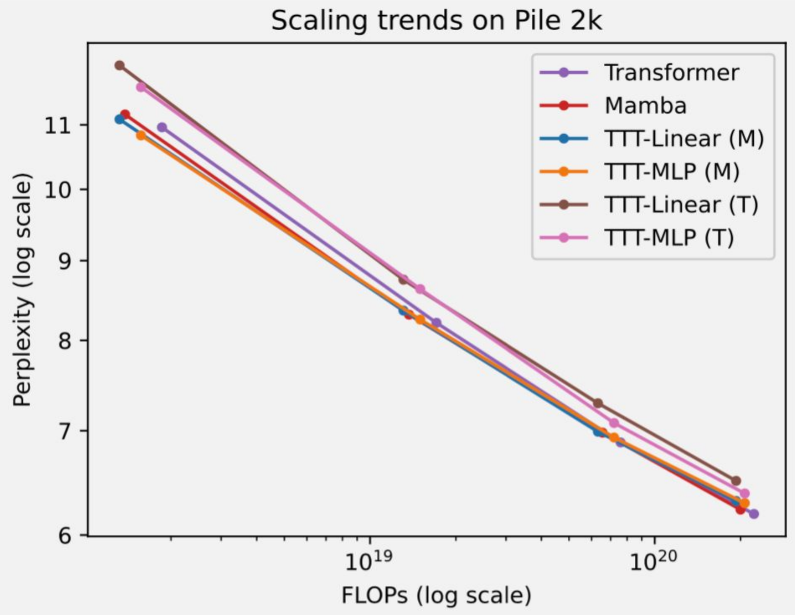
Making its Training Efficient

$$G_t = \nabla l(W_{t-1}; x_t).$$

$$W_t = W_{t-1} - \eta\, G_t = W_0 - \eta \sum_{s=1}^{t} G_s,$$

To parallelize G_t for $t = 1, \ldots, T$, we can take all of them w.r.t. W_0. This variant with $G_t = \nabla\ell(W_0; x_t)$ is known as *batch gradient descent*, since $\sum_{s=1}^{t} \nabla\ell(W_0; x_s)$ is the same as the gradient w.r.t. W_0 over $x_1, \ldots, x_t$ as a batch. However, in batch GD, W_t is effectively only one gradient step away from W_0,

Results



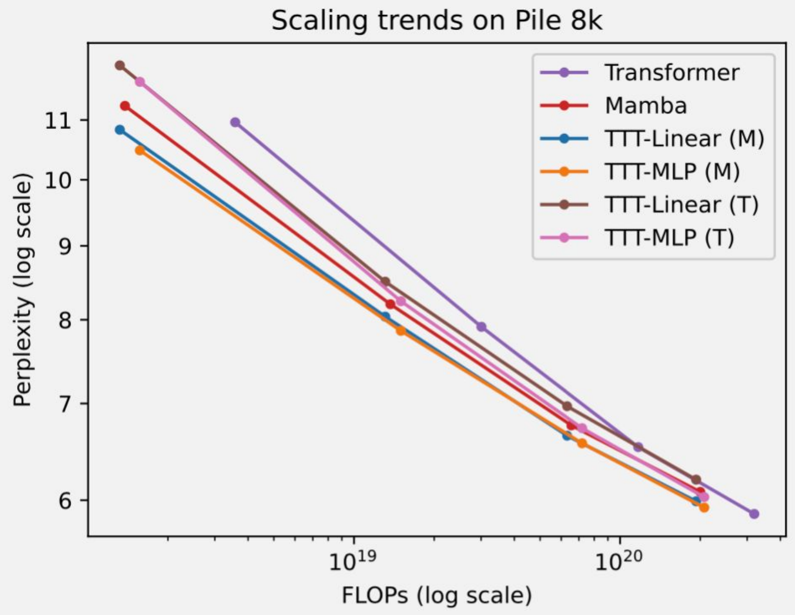


Figure 10. Evaluations for context lengths 2k and 8k on the Pile. Details in Subsection 3.1. TTT-Linear has comparable performance as Mamba at 2k context, and better performance at 8k.

Results

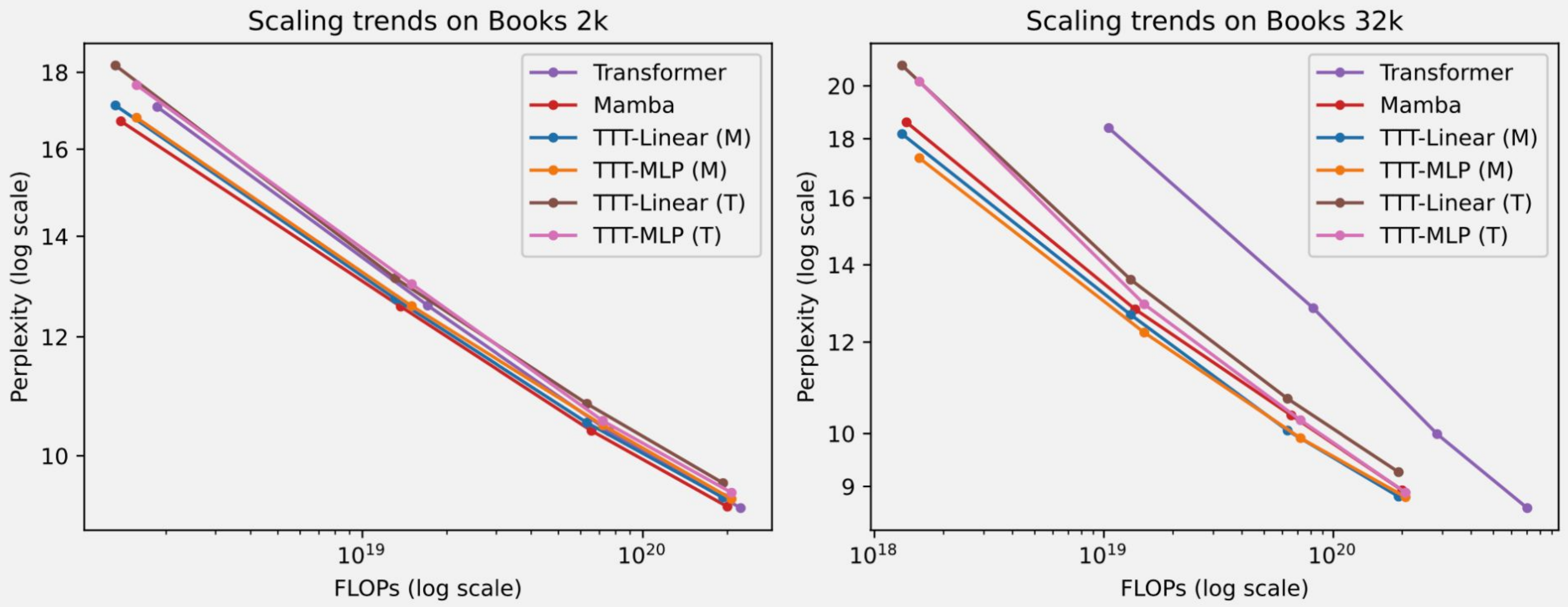


Figure 11. Evaluations for context lengths 2k and 32k on Books. Details in Subsection 3.2. Our complete results for context lengths 1k, 2k, 4k, 8k, 16k, 32k, including Transformer finetuning, are in Figure 15 (in Appendix). Most observations from the Pile still hold.

Results

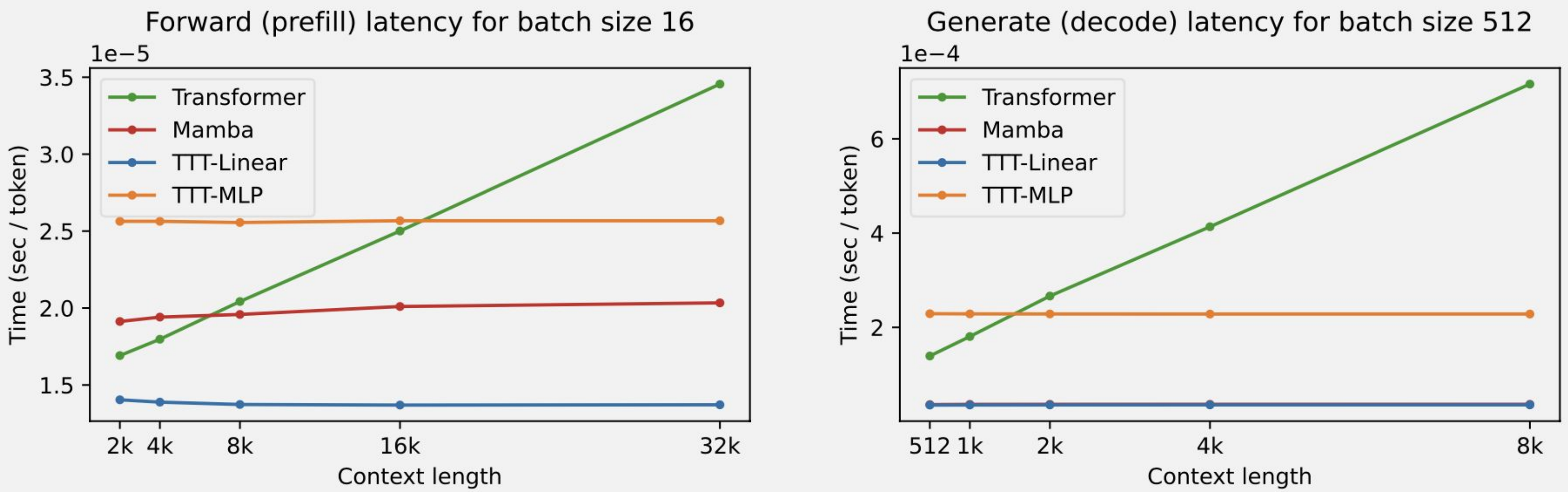


Figure 12. Latency on an NVIDIA A100 GPU with 80G HBM and PCIe connections.