Exposing Attention Glitches with Flip-Flop Language Modeling

Bingbin Liu
Kempner Institute
(work done at CMU)

Jordan T. Ash
MSR NYC

Surbhi Goel
UPenn

Akshay Krishnamurthy
MSR NYC

Cyril Zhang
MSR NYC

Occasional errors of language models

(“hallucination”)

GPT-4o: $S = \left(\frac{1}{2} \times 0\right) + \left(\frac{1}{2} \times \frac{1}{2}\right) + (0 \times \frac{1}{2}) = 0.$

This work: one explanation of occasional errors, exposed using a naïve task.

(attention glitches) (flip-flop)

- Revealing inherent limitations of attention.
- Mitigations & next steps?

How Transformers perform sequential reasoning?

background

↘

Capabilities?

Transformer is capable to solve a broad class of reasoning problems.

How to formalize?

today's focus

↙

Limitations?

Transformer struggles to find an optimal solution in practice.

Formalizing sequential reasoning

```
    1
× (-1)
× (-1)
   ×1
   …
```

arithmetic

[Nogueira et al. 21, Jelassi et al. 23]

Bobo is a dog.

A dog is a mammal.

Is Bobo a mammal?

deduction

[Wei et al. 22, Saparov et al. 22]

```
x = 1
for _ in range(10):
  x = x**2 + 1
print(x)
```

program

[Wang et al. 22, Nijkamp et al. 22]

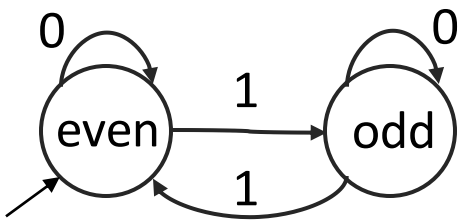
→ Sequential state transitions

Formalizing sequential reasoning

arithmetic

$$1 = (-1)^0,$$
$$-1 = (-1)^1,$$

for the exponents:



aka. **parity counter**

$Q = \{\text{even}, \text{odd}\}$
$\Sigma = \{0, 1\}$

Finite-state automata

$$\mathcal{A} = (Q, \Sigma, \delta)$$

states **inputs** **transitions**

$$q_t = \delta(q_{t-1}, \sigma_t)$$

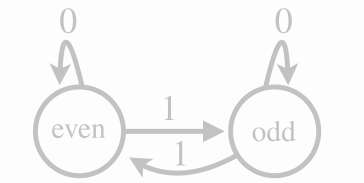
1 state transition
↔ 1 reasoning step

Transformer can simulate automata

$$\mathcal{A} = (Q, \Sigma, \delta),$$
$$q_t = \delta(q_{t-1}, \sigma_t).$$

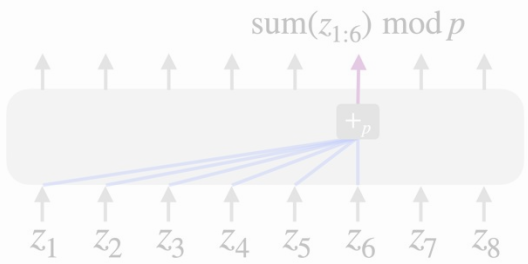
Krohn-Rhodes: (a subset of) $\mathcal{A}$ can be *decomposed* into 2 **base factors**:

- Each factor representable by 1 Transformer layer.

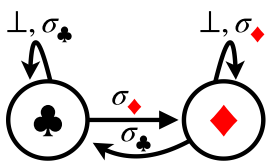


$\delta(q, \sigma) = q + \sigma \bmod p$

mod counter

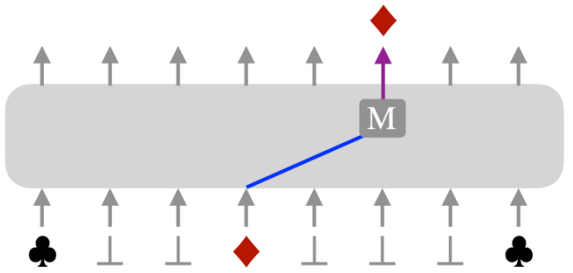


uniform attention



$\delta(q, \sigma) = \sigma,$
$\delta(q, \perp) = q.$

memory unit



sparse attention

Focus of today

How Transformers perform sequential reasoning?

Capabilities?

Transformer is capable to solve
a broad class of reasoning problems.

~ *simulating automata*
(two “base units”)

Limitations?

Transformer struggles to find
an optimal solution in practice.

… even for the base unit.

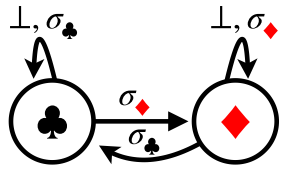
A simple(st) language based on the memory unit

Recall: one (of the 2) *base factor* of automata decomposition.

"flip-flop"

1-bit memory unit

$Q = \{0,1\}, \Sigma = \{\sigma_0, \sigma_1, \perp\}$



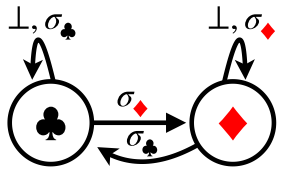
Flip-Flop Language (FFL): sequences of instruction-value pairs.

- 3 instructions: `w` (write), `i` (ignore) , `r` (read).
- 2 values: {0, 1} – ***Constraint***: the value for `r` must be the same as the last `w`.

Why Flip-Flop?

1-bit memory unit

$Q = \{0,1\}, \Sigma = \{\sigma_0, \sigma_1, \perp\}$



Flip-Flop Language (FFL)

`w 1 i 0 r 1 w 0 i 1 r 1`

1. An atomic unit embedded in many reasoning tasks (e.g. automata).

- **`ignore`**: Irrelevant context / distractors
  [Shi et al. 23]

```
Alice put the keys
on the table.
Bob came in later.
….
Bob left and took
the keys from ___ .
```

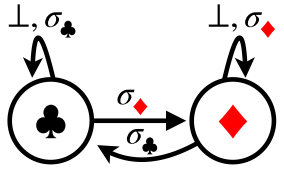
- **`write`**: Updated code semantics
  [Miceli-Barone et al. 23]

```
def f():
  sum = len
  . . .
  x = [1,2,3]
  . . .
  assert(sum(x))==3
```

Why Flip-Flop?

1-bit memory unit

$Q = \{0,1\}, \Sigma = \{\sigma_0, \sigma_1, \perp\}$

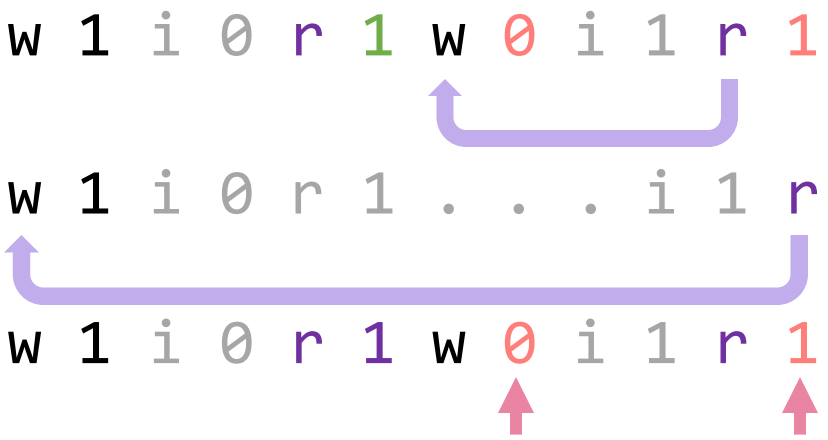


Flip-Flop Language (FFL)

w 1 i 0 r 1 w 0 i 1 r 1

1. An atomic unit embedded in many reasoning tasks (e.g. automata).

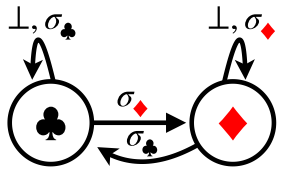
- Induction head [Olsson et al. 22]
- Long-range dependency [Tay et al. 20]
- Closed-domain hallucination [Dziri et al. 22, OpenAI 23]



Why Flip-Flop?

1-bit memory unit

$Q = \{0,1\}, \Sigma = \{\sigma_0, \sigma_1, \perp\}$



Flip-Flop Language (FFL)

`w 1 i 0 r 1 w 0 i 1 r 1`

2. The cleanest, interesting setting of failure, compared to prior work:

- Less interesting: *fixed attention patterns*. e.g. Parity [Hahn 20, Barak 22], FIRST [Chiang & Cholak 22]
- More complicated:
  - Formal languages [Bhattamishra 20, Yao 21, van der Poel 23].
  - Algorithmic reasoning [Nogueira 21, Anil 22, Zhang 22, Dziri 23, Zhang 23].
  - Code [Chen 21, Miceli-Barone 23], math [Cobbe 21, Lu 22, Shi 23].
  - Natural languages: QA [Rogers 21], long-range tasks [Tay 20], multi-step reasoning [Saparov 22].

Flip-Flop Language Modeling (FFLM)

Flip-Flop Language (FFL): instruction-value pairs; `r` recalls the most recent `w`.

Task: supervise & evaluate only on the values following `r`.

- Deterministic task; training signals not "drawn" by irrelevant tokens.

Data distribution: FFL(p_i), where p_i can vary across train/test.

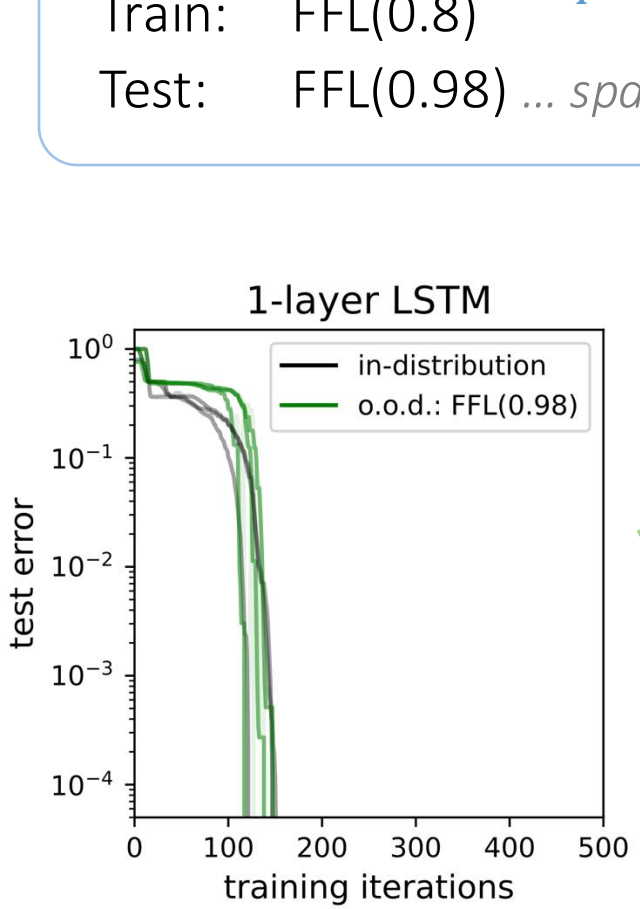
- $p_w = p_r = (1 - p_i)/2$, $p_0 = p_1 = 0.5$. Fix length $T = 512$.

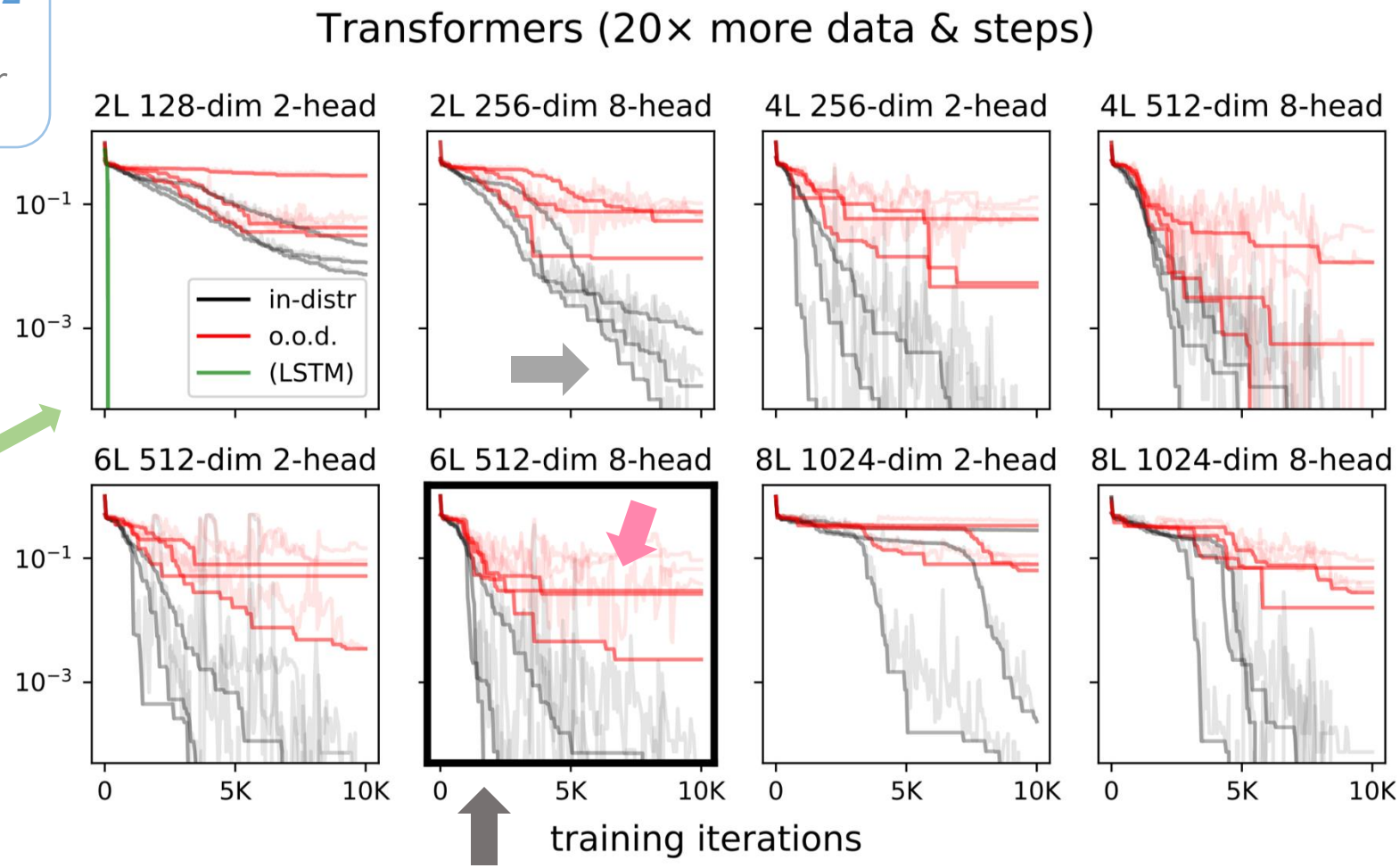
FFLM Results

Train: FFL(0.8)
Test: FFL(0.98) … *sparser*

$T = 512$

Attention glitches



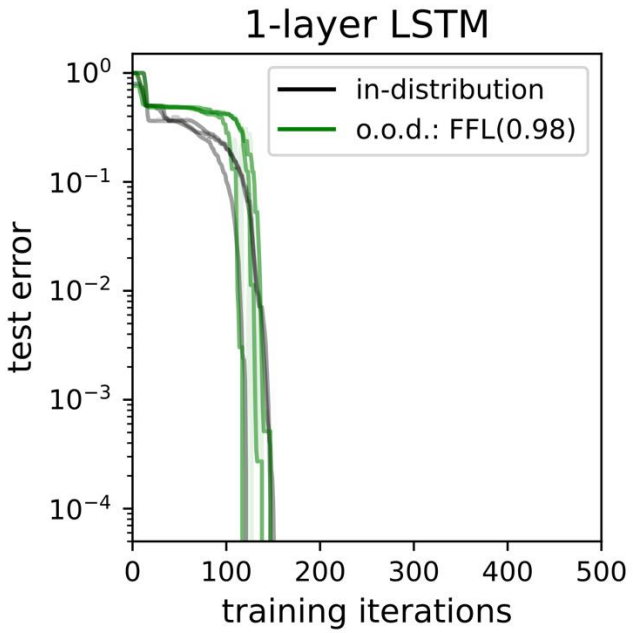


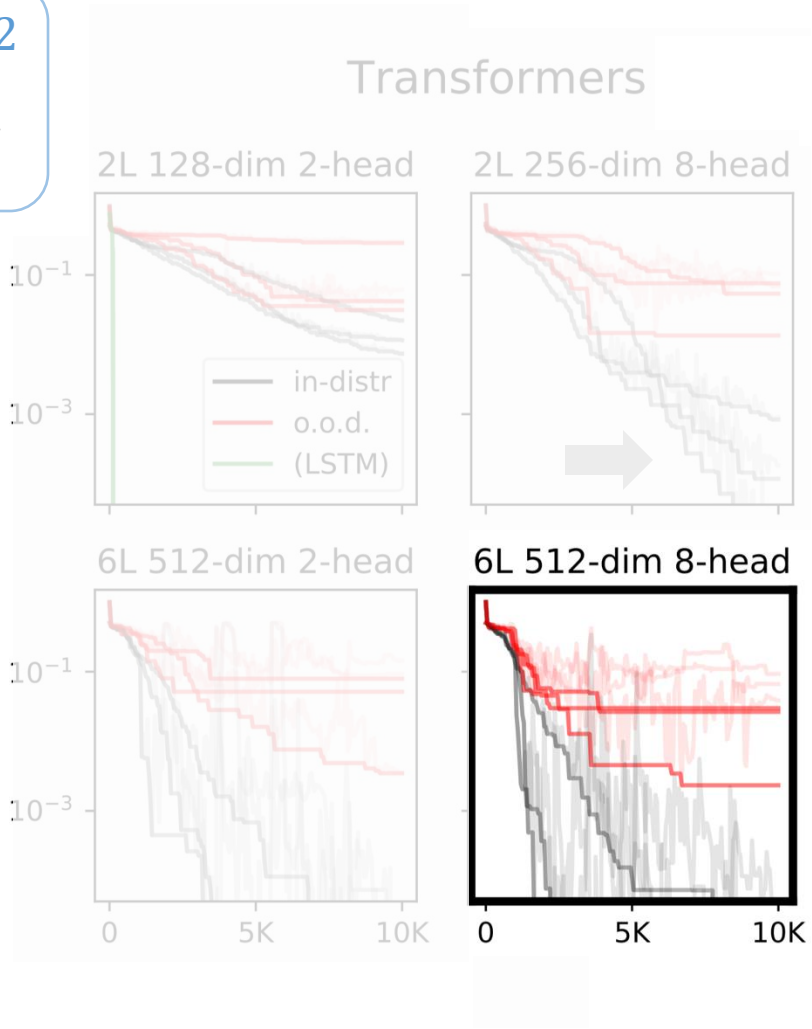
FFLM Results

Train: FFL(0.8)
Test: FFL(0.98) … *sparser*

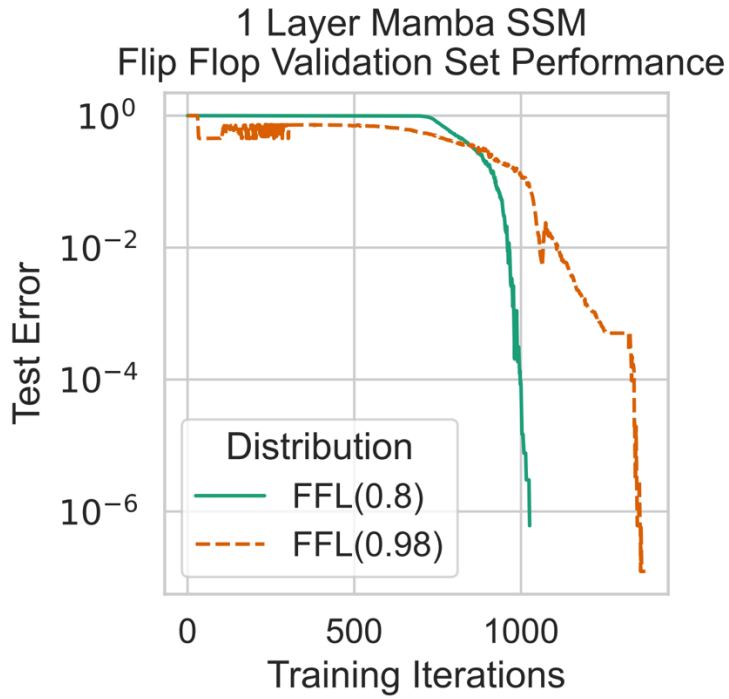
$T = 512$

Attention glitches





State-Space Models (SSMs)?



[Sarrof et al. 24]

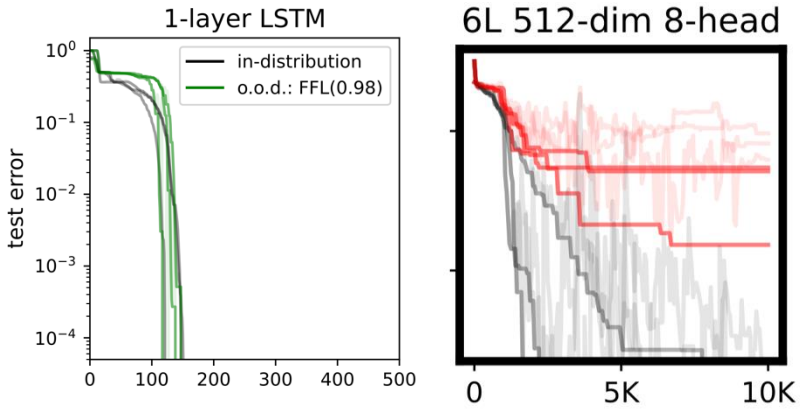
Attention Glitches

$\text{FFL}(p_i)$: $p_w = p_r = (1 - p_i)/2.$

Def: *imperfect hard retrieval*.

- Transformers exhibit a long tail of errors.
- 1-layer LSTMs extrapolate *perfectly*.

Even proprietary models are not robust.



User: Hi, let's play a game. There are 3 instructions: "write", "read", and "ignore".

. . .
For example, ...

Now, please answer the following sequence: …

GPT 4o: ~50% acc
1, 0, 0, 1, 1, 0, 0, 1, 1, 0

GPT o1: ~90% acc

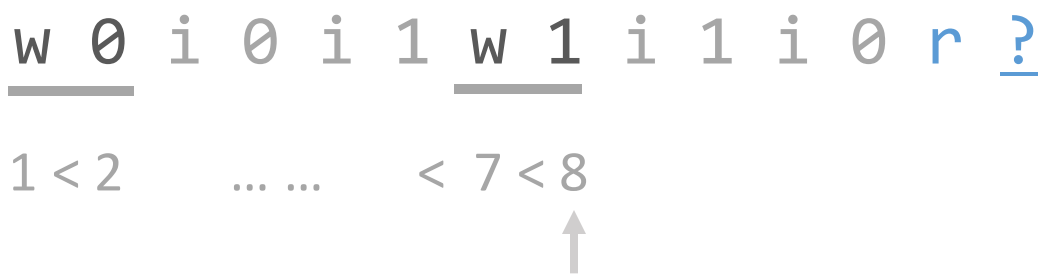
(based on 10 trials)

Cause of attention glitches?

FFL(p_i): $p_w = p_r = (1 - p_i)/2$.

Not due to representation power.

- Solvable with ***2-layer 1-head*** . → see also: induction head [Bietti et al. 23, Sanford et al. 24]
  - 1st layer: locate `w`'s and the associated values.
  - 2nd layer: tie-break using position encoding.



Cause of attention glitches?

FFL(p_i): $p_w = p_r = (1 - p_i)/2$.

Not due to representation power: *2-layer 1-head* suffices (Bietti et al. 23, Sanford et al. 24).

2 potential causes, each related to 1 type of OOD error.

Diluted soft attention: caused by more items (e.g. *denser* `w`) in the softmax.

$$a_{\max} = \frac{\exp(z_{\max})}{\underbrace{\exp(z_1)+\cdots+\exp(z_t)}_{\text{to be ignored}}+\exp(z_{max})}$$

- Also identified in prior work [Hahn 20, Chiang & Cholak 22].
- Possible mitigation: Switching to hard attention.

Cause of attention glitches?

FFL(p_i): $p_w = p_r = (1 - p_i)/2$.

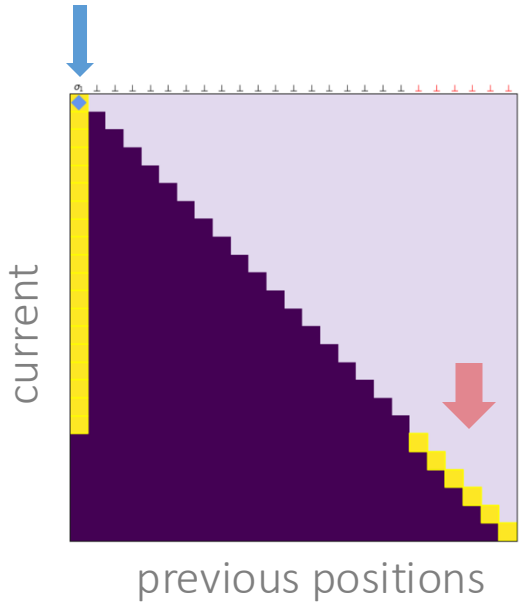
Not due to representation power: *2-layer 1-head* suffices (Bietti et al. 23, Sanford et al. 24).

2 potential causes, each related to 1 type of OOD error.

Diluted soft attention: caused by more items (e.g. *denser* `w`) in the softmax.

Position over content: lead to wrong argmax.
(e.g. *sparser* `w`, length gen)

- "Implicit" length generalization: w-r dist can increase.



Mitigating attention glitches

Data & scaling

- Incorporating OOD data.

  Performance ceiling; a few samples can help.
  e.g. “priming” [Jelassi et al. 23]

- Resource scaling: larger, train for longer.

  Fresh samples → better coverage.

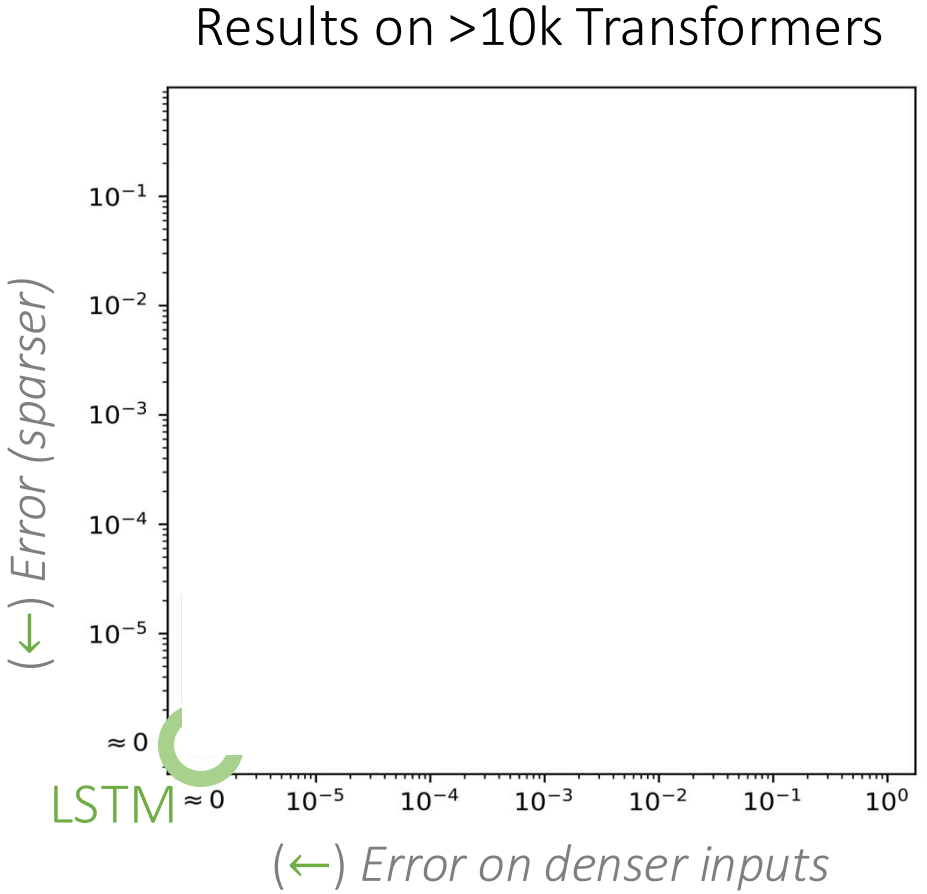
Algorithmic control

- Regularization

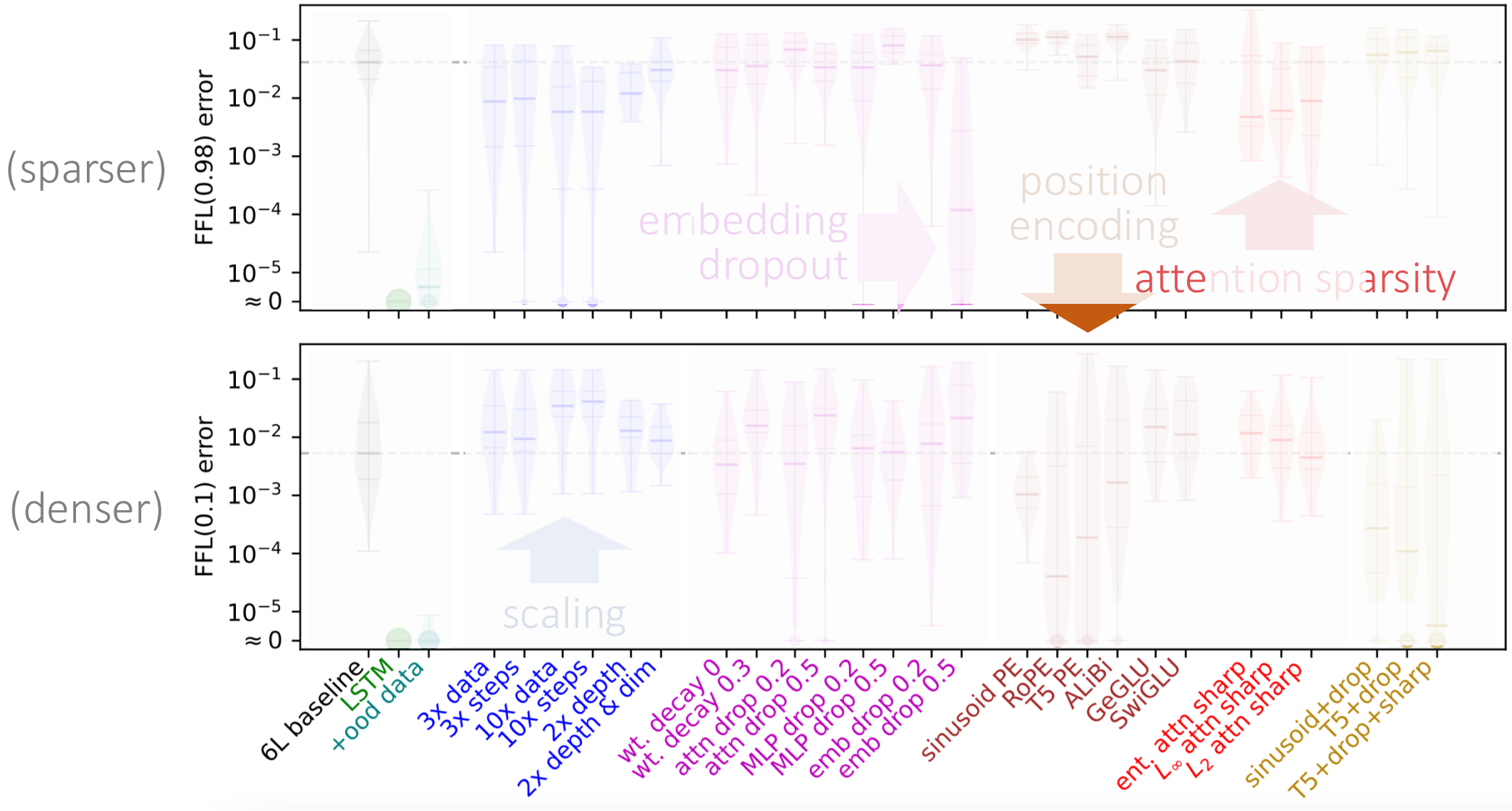
  weight decay, dropout, attention sparsity.

- Architectural choices

  position encoding, activation.



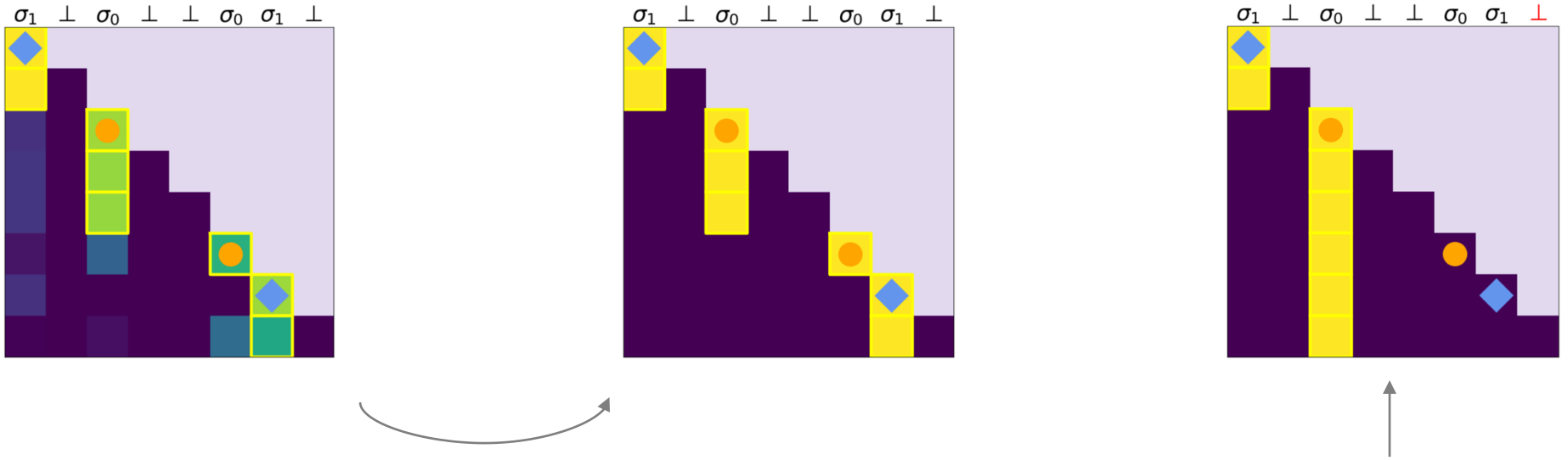
Surprisingly hard to fix: no mitigation helps with *both*



Zoom-in: attention sparsity regularization

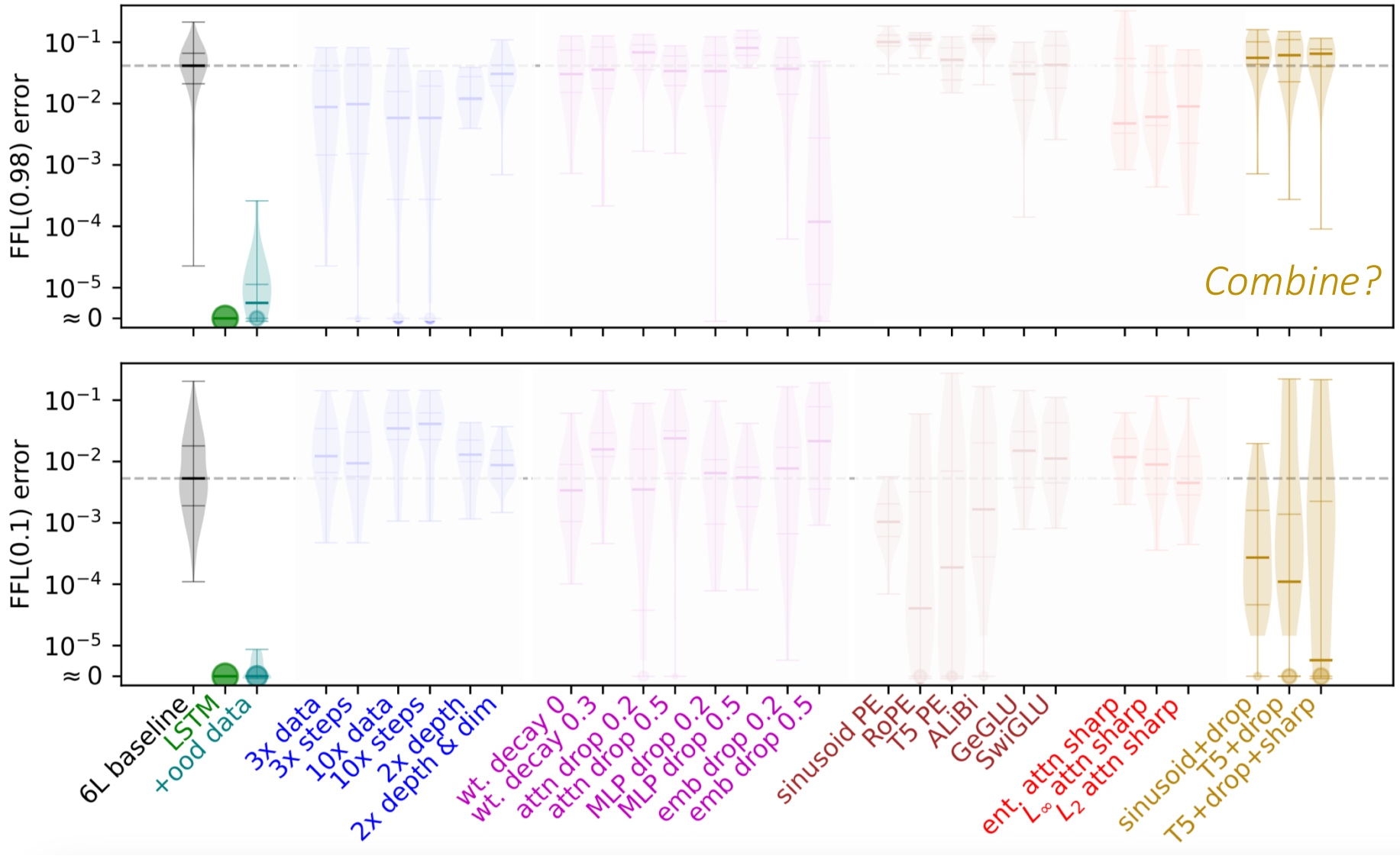
Regularize for sparsity: $\sum_i \alpha_i \log \alpha_i$ (entropy), $-|\alpha|_\infty$ (L_∞), $-|\alpha|_2$ (L_2).

1-layer 1-head model:



** In general, use caution with attention-based interpretability.* [Wen et al. 23]

Surprisingly hard to fix: no mitigation helps with *both*



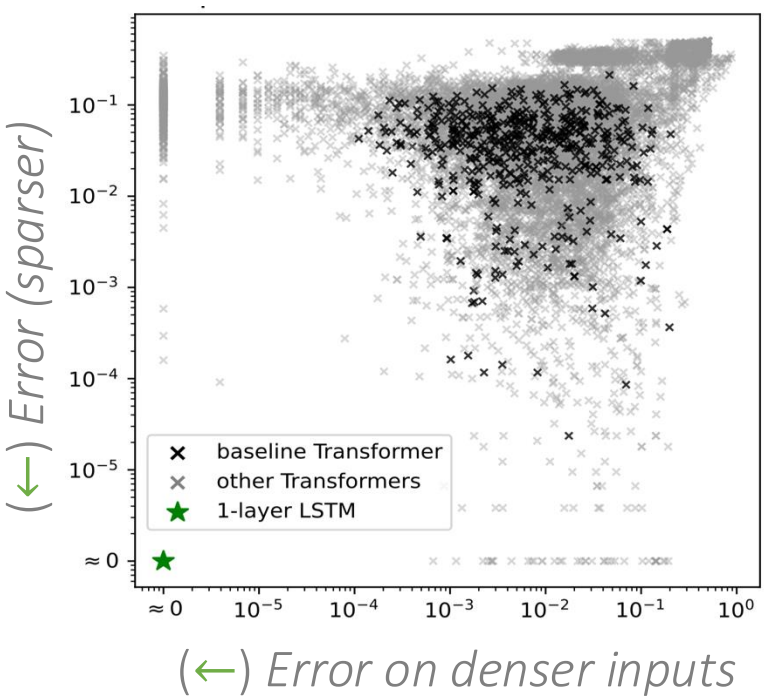
Attention glitches with Flip-flop

One source of Transformer's occasional errors: imperfect hard retrieval.

- What: Transformer's long tail of errors, even on an *extremely simple* task.
  - *Goal: learn as well as LSTM?*
- Why: Two inherent limitations of attention.
  - Imply various errors; hard to mitigate.
- How: Scaling is no panacea. Data matters.
  - *… recurring theme recently.*

Flip-Flop Language (FFL)

`w 1 i 0 r 1 w 0 i 1 r 1`



How Transformers perform sequential reasoning?

Capabilities?

Transformer is capable to solve a broad class of reasoning problems.

- Formalized as simulating automata.
- Parallel solution of $o(\mathbf{length})$ steps.

Limitations?

Transformer struggles to find an optimal solution in practice.

- Occasional errors on the naïve flip-flop.
- Exposing inherent limitations of attention.

Discussion: Towards glitch-free models

Data: collection/selection (“priming”), format, usage (curriculum, repetition).

Architecture: recurrence/hybrid, positional encoding, tokenization.

(RoPE, FIRE, Abacus)

- Different notions of error: average-case (L_2) vs worst-case (L_∞) .
- Do we really need to be stringent?

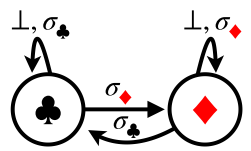
> **Turing** (1947)*: “If a machine is expected to be infallible, it cannot also be intelligent.”*

Still: awareness & caution when there are other priorities (e.g. safety, robustness).

Discussion: Using small-scale “sandboxes”

“Sandbox for the Blackbox”

Sandboxes: minimal, representative abstractions of complex systems.



- Why: formal analyses, rapid iteration; controllable & accessible.

Can be used for…

- Scientific understanding

  e.g. methods, training dynamics, mech interp.

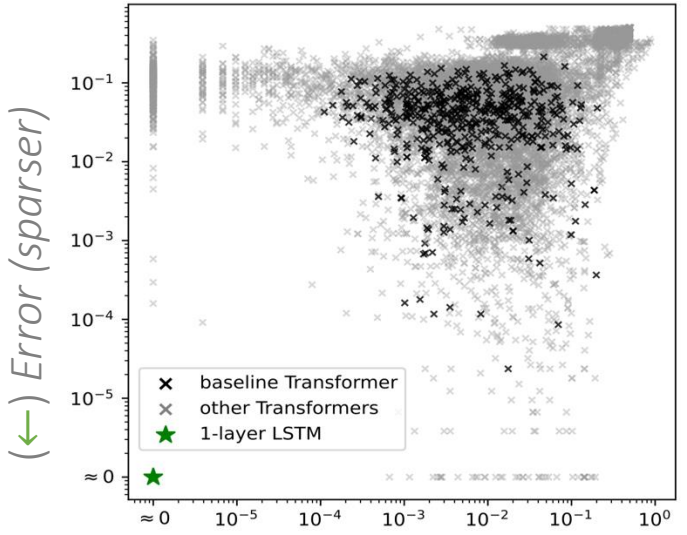
- Evaluation and diagnosis

  e.g. flip-flop, arithmetic, needle-in-a-haystack.

- Efficient algorithmic innovations

  e.g. architecture, training recipe, inference procedure.

Results on 10k Transformers



Discussion: Scaling (up)

Scaling is not a panacea…

- e.g. inverse scaling challenges; environmental/accessibility considerations.

… but it does help.

- Representation: larger models are more expressive.
- Optimization: wider models are more stable to train.
- Favorable properties: generalization; “emergent” abilities *… how to evaluate?*

∴ Be mindful of it when making design choices. (The Bitter Lesson – Sutton 19)

How Transformers perform sequential reasoning?

Capabilities?

Transformer is capable to solve a broad class of reasoning problems.

- Formalized as simulating automata.
- Parallel solution of $o(\text{length})$ steps.

Limitations?

Transformer struggles to find an optimal solution in practice.

- Occasional errors on the naïve flip-flop.
- Exposing inherent limitations of attention.

Thank you! Questions? 

Appendix

Thanks for wanting to know more! ☺

Part I
Transformers learn shortcut to automata

Reasoning as simulating automata

$\mathcal{A} = (Q, \Sigma, \delta)$

states, inputs, transitions

Simulating $\mathcal{A}$: learn a *seq2seq function* for sequences of length T.

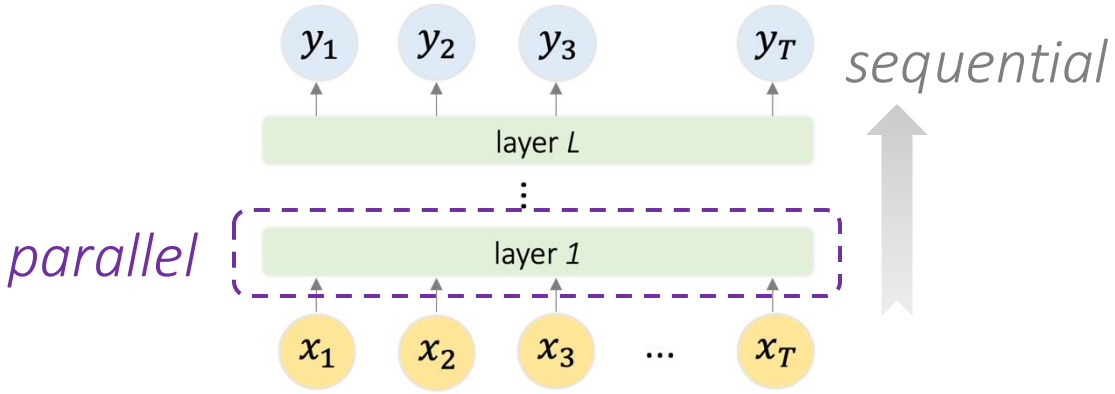
Output: q_1 q_2 $\cdots$ q_T $\subset Q^T$ (states)

Model (Transformer/RNN)

Input: σ_1 σ_2 $\cdots$ σ_T $\subset \Sigma^T$ (alphabet)

Solutions with much fewer sequential steps?

Transformers

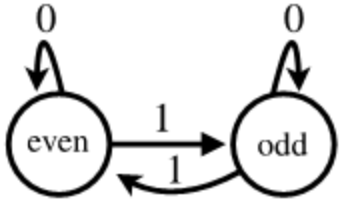


#layers ≪ #positions

Why are $o(T)$-step solutions possible

$\mathcal{A} = (Q, \Sigma, \delta)$

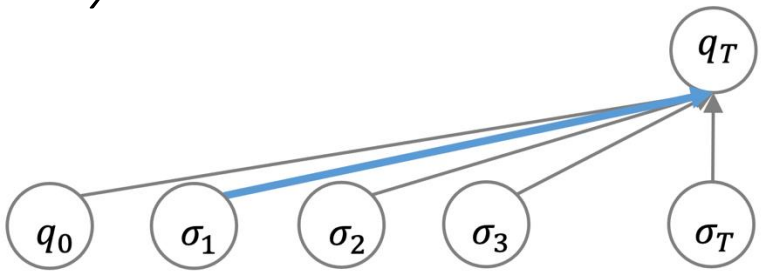
states, inputs, transitions

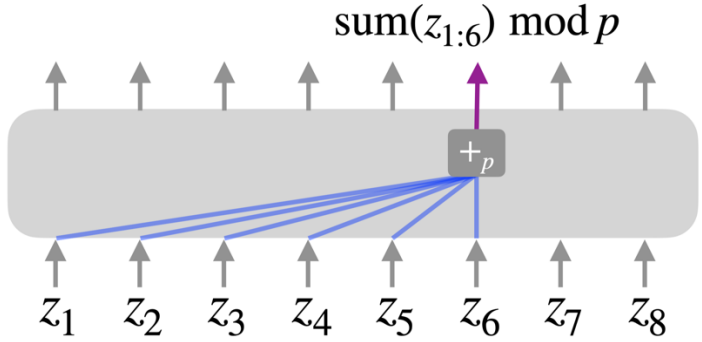
e.g. parity 

… T-step solution: $q_t = \delta(q_{t-1}, \sigma_t), \forall t \in [T]$.

1 step solution: $q_t = (\sum_{\tau \le t} \sigma_\tau) \bmod 2$.

no dependency on T



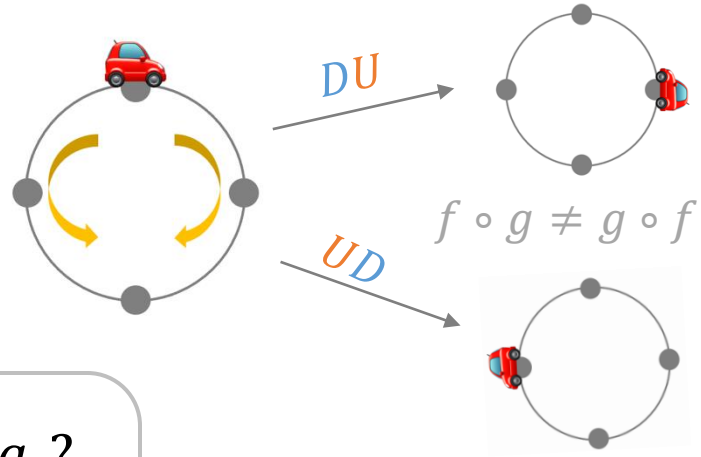


uniform attention

Decomposition: example

$$\mathcal{A} = (Q, \Sigma, \delta), \quad q_t = \delta(q_{t-1}, \sigma_t).$$

$Q = \{$ 🚗 (1), 🚙 (-1) $\} \times \{0,1,2,3\}$, $\Sigma = \{D\,(\text{drive}), U\,(\text{U-turn})\}$.



	q_0	D	D	D	U	D	D	U	U	D $\rightarrow q_t$?
Parity:	1	1	1	1	-1	-1	-1	1	-1	-1 $\rightarrow$ 🚗
Signed sum:	0	1	1	1	0	-1	-1	0	0	-1 $\rightarrow$ 0

$O(1)$ layer each

1. Direction = parity (sum) of U. (parity: {1, -1} $\leftrightarrow$ {0, 1})
2. Position = signed sum mod 4 : sign = parity of U.

Transformers can simulate automata in practice

19 automata, across various depths.

- Good in-distribution accuracy.
- Deeper factorization → more layers.
  - Rows ordered by #factorization steps.

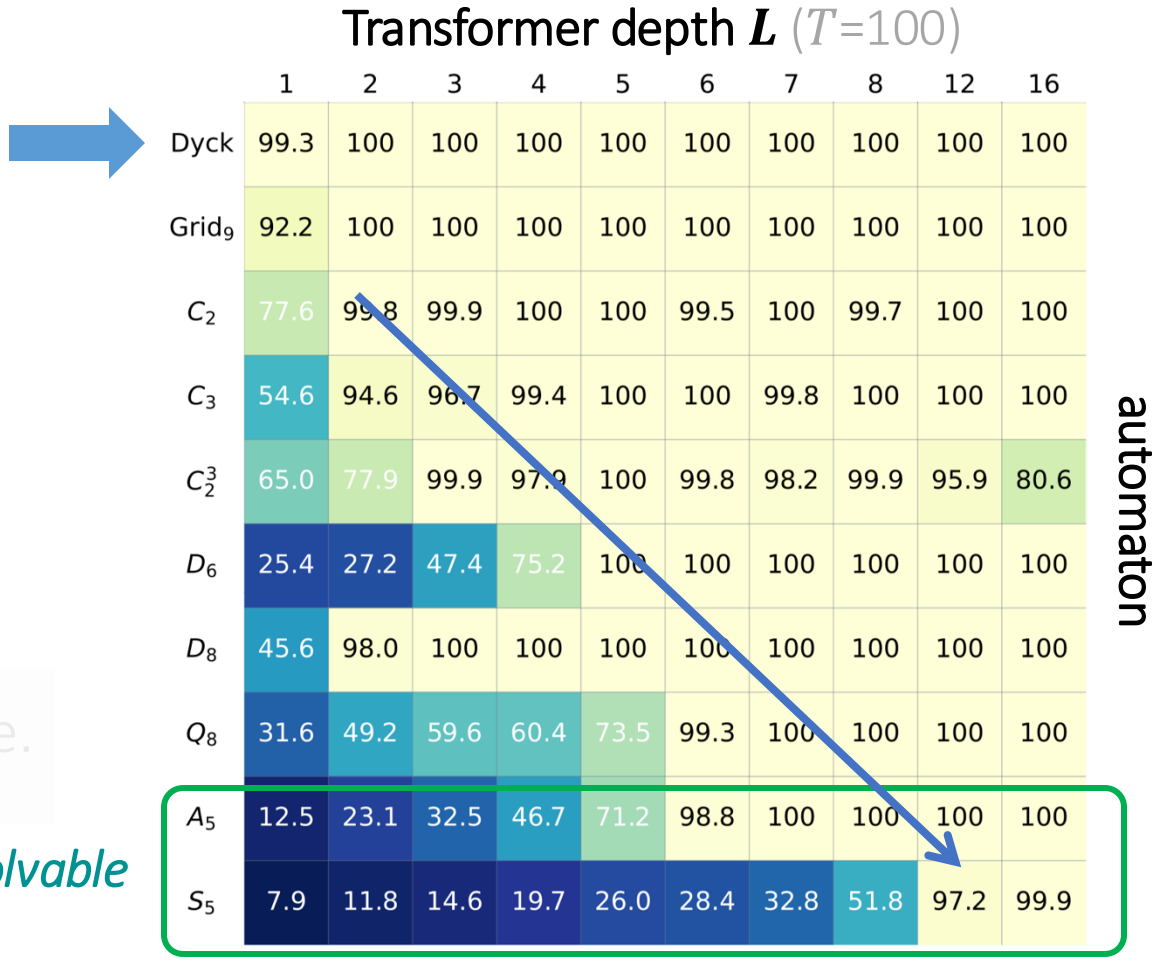
Constructions ≠ empirical solutions

There are multiple constructions.

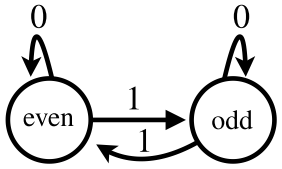
- $O(\log|Q|)$ layers suffice; $O(|Q|^2)$ if solvable.

Infinitely many

non-solvable



OOD Generalization - Parity



$(T = 40)$

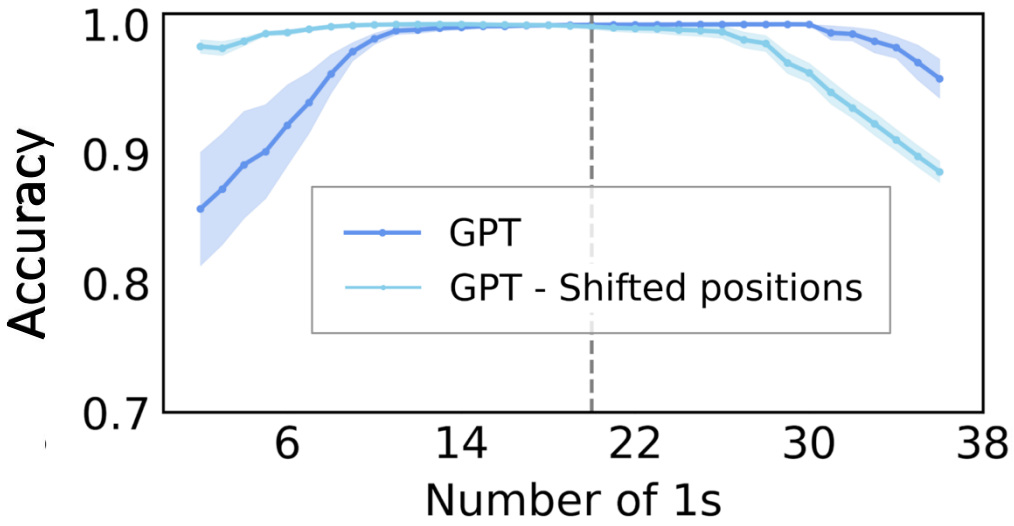
- train: $p(1) = 0.5$
- test: other $p(1)$.

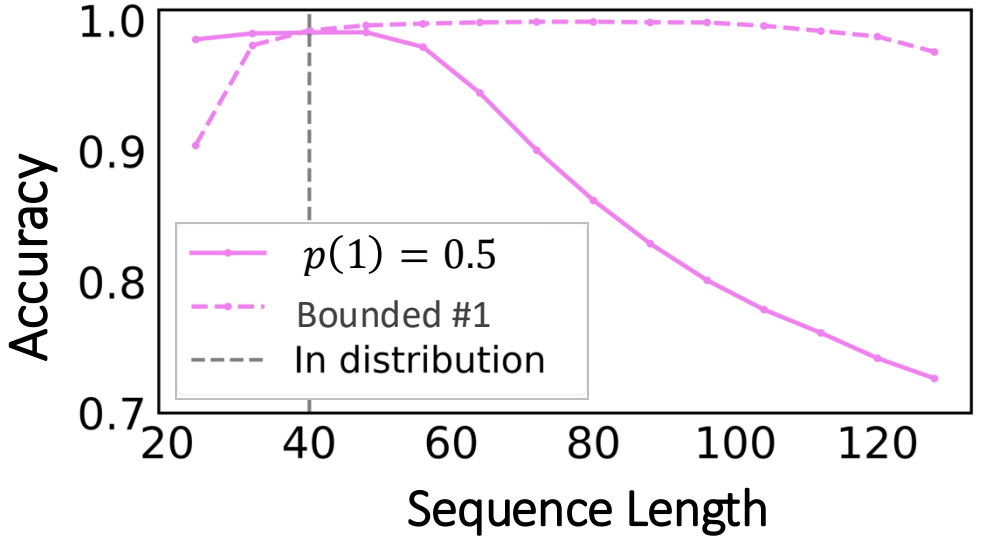
$$q_t = \left(\sum_{i\in[t]} \sigma_i\right) \bmod 2$$

concentrates $\approx 0.5t$

memorized by MLP

→ fail at unseen $\left(\sum_{i\in[t]} \sigma_i\right)$.

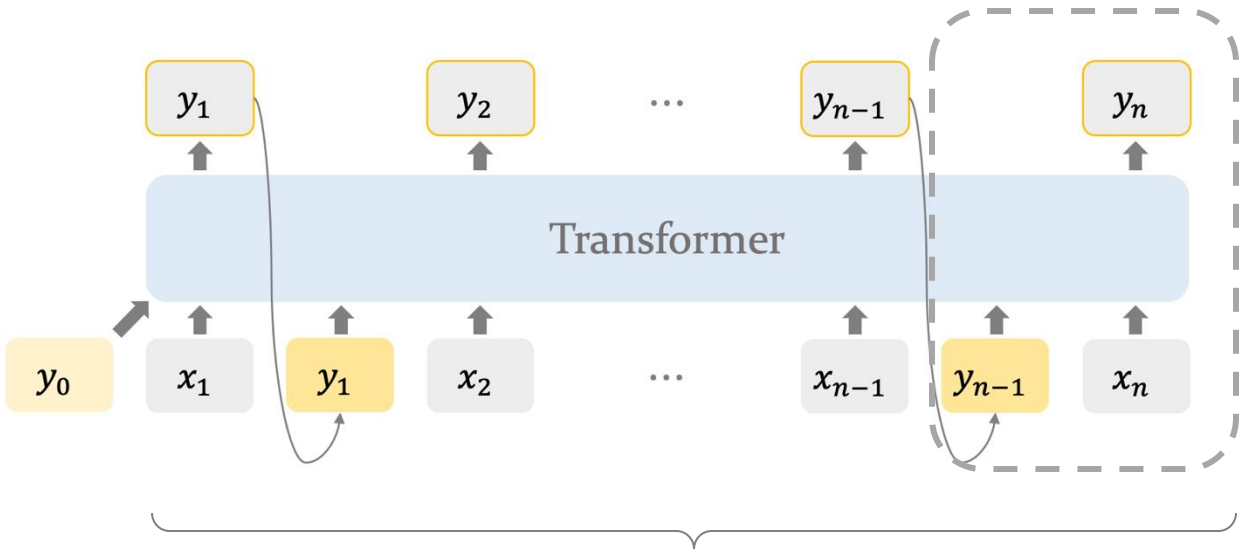




Autoregressive mode of Transformers

Fix to OOD: iterative/autoregressive solutions: use q_{t-1} as inputs.

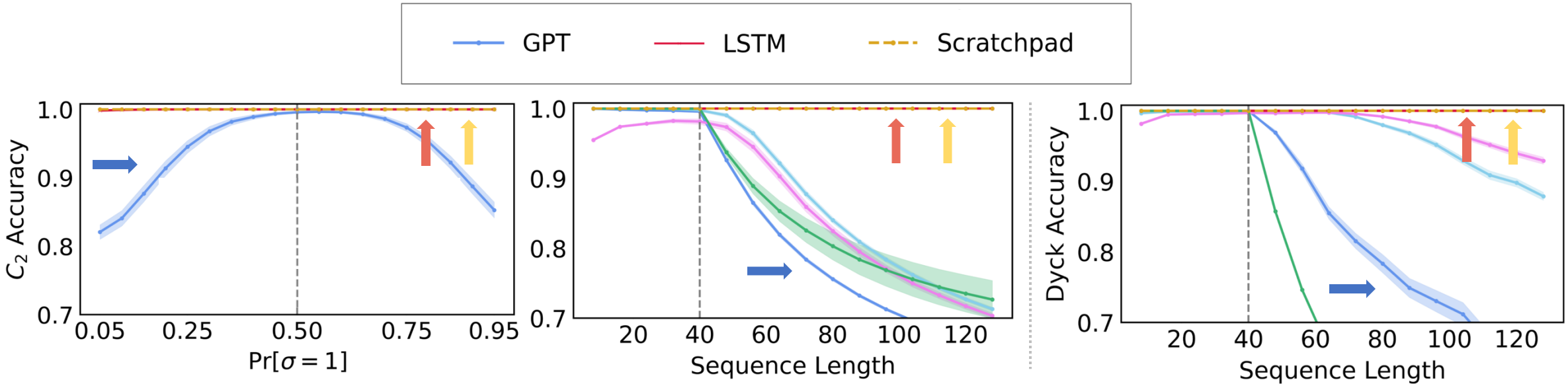
Scratchpad (Nye et al. 21) $q_t = \delta(q_{t-1}, \sigma_t)$



Not shortcut

No longer parallel across positions

Autoregressive mode of Transformers



Transformers generalize, when made autoregressive with scratchpad [Nye et al. 22].

→ Open: *Can we learn shortcuts that generalize?*

Training with limited supervision

Less ideal setups?

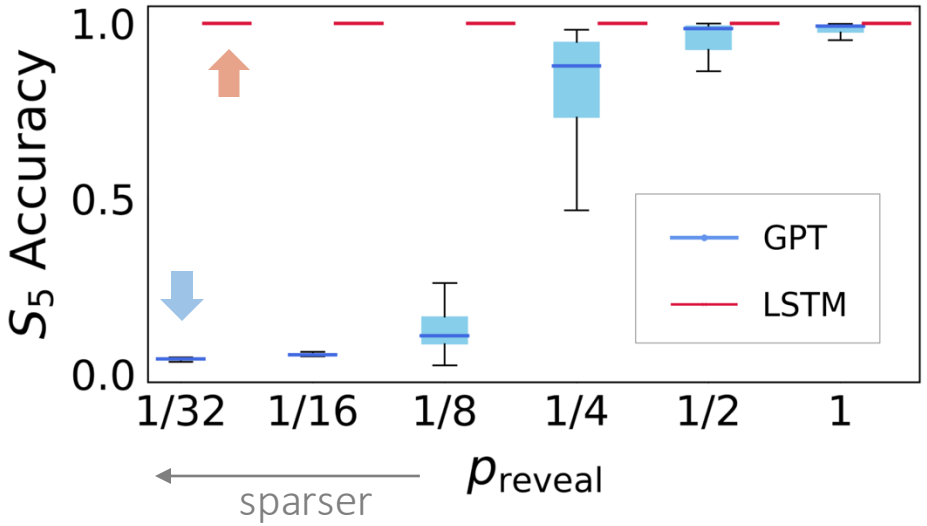
Indirect supervision

train & test on a function of q_t.

$\text{Dyck}_{4,8}$	Grid_9	S_5	C_4	D_8
stack top	$\mathbb{1}_{\text{boundary}}$	$\pi_{1:t}(1)$	$\mathbb{1}_{0 \bmod 4}$	location
100.0	99.8	99.8	99.7	99.8

Incomplete supervision

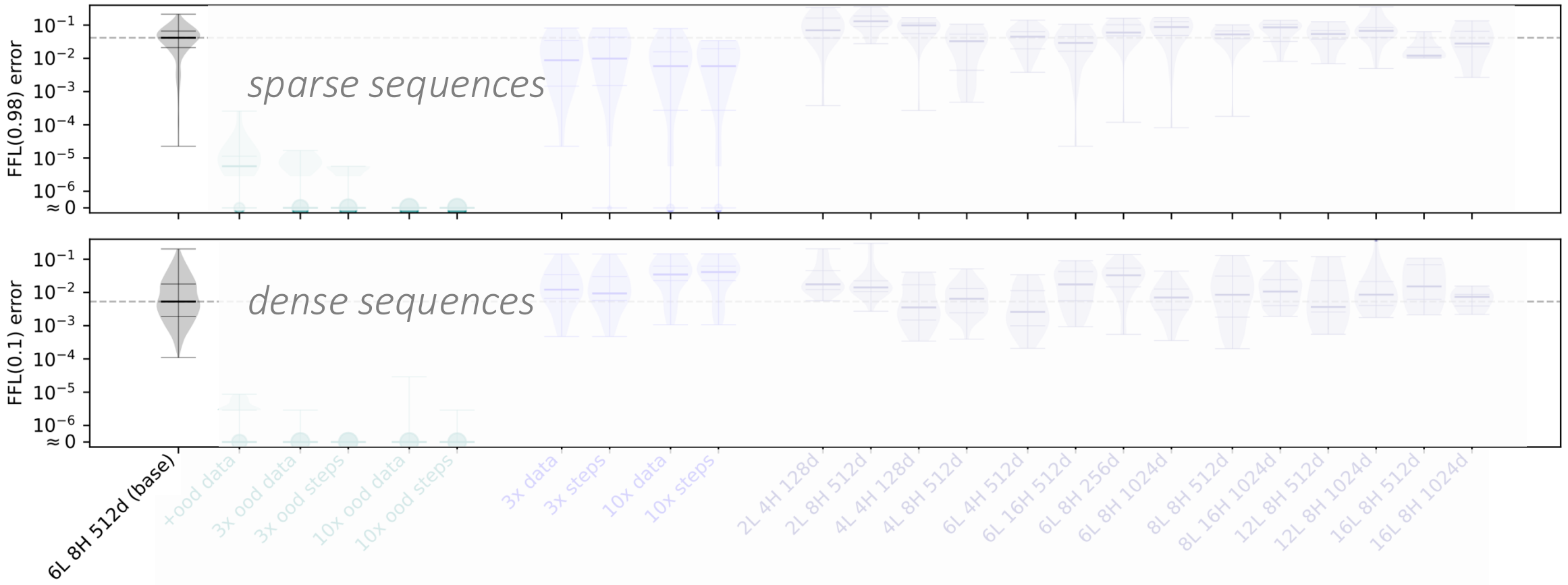
q_t is revealed w.p. $p_{\text{reveal}} \in [0,1]$.



LSTM is always 100% → Open: *How to improve Transformer training?*

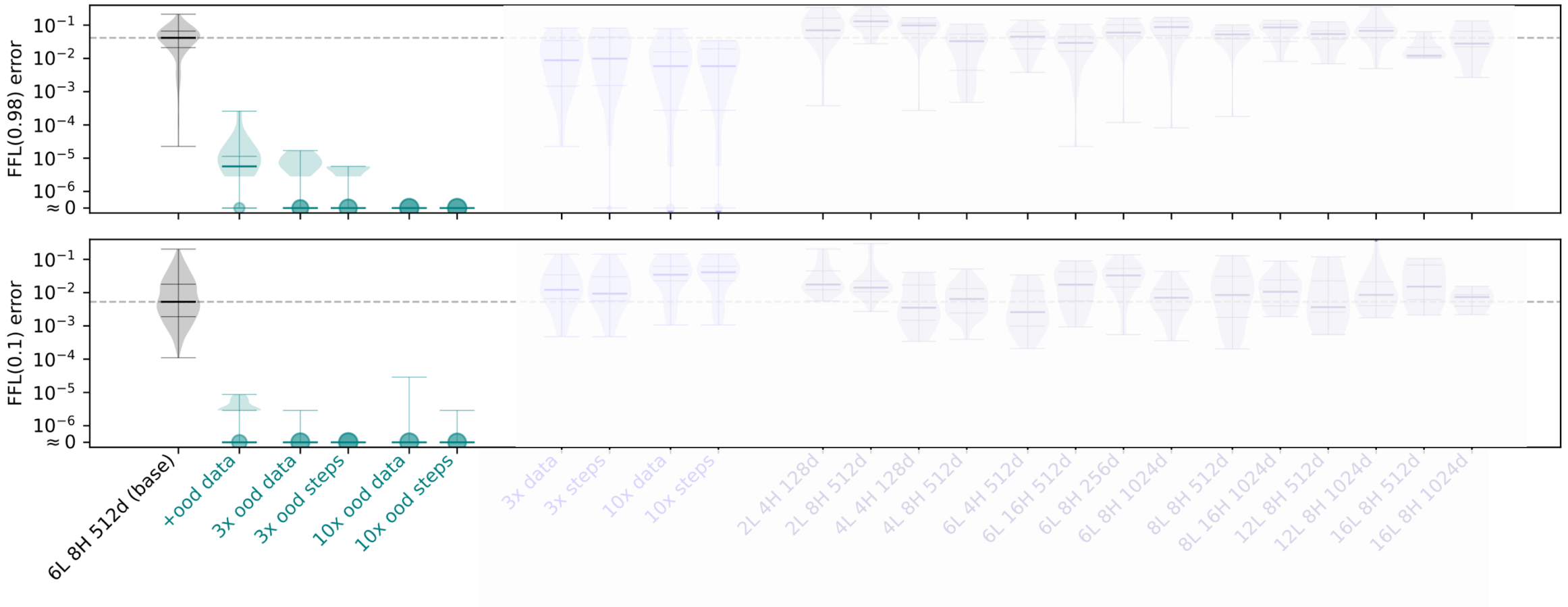
Part II
Exposing attention glitches with FFLM

Mitigations



Mitigations – Data

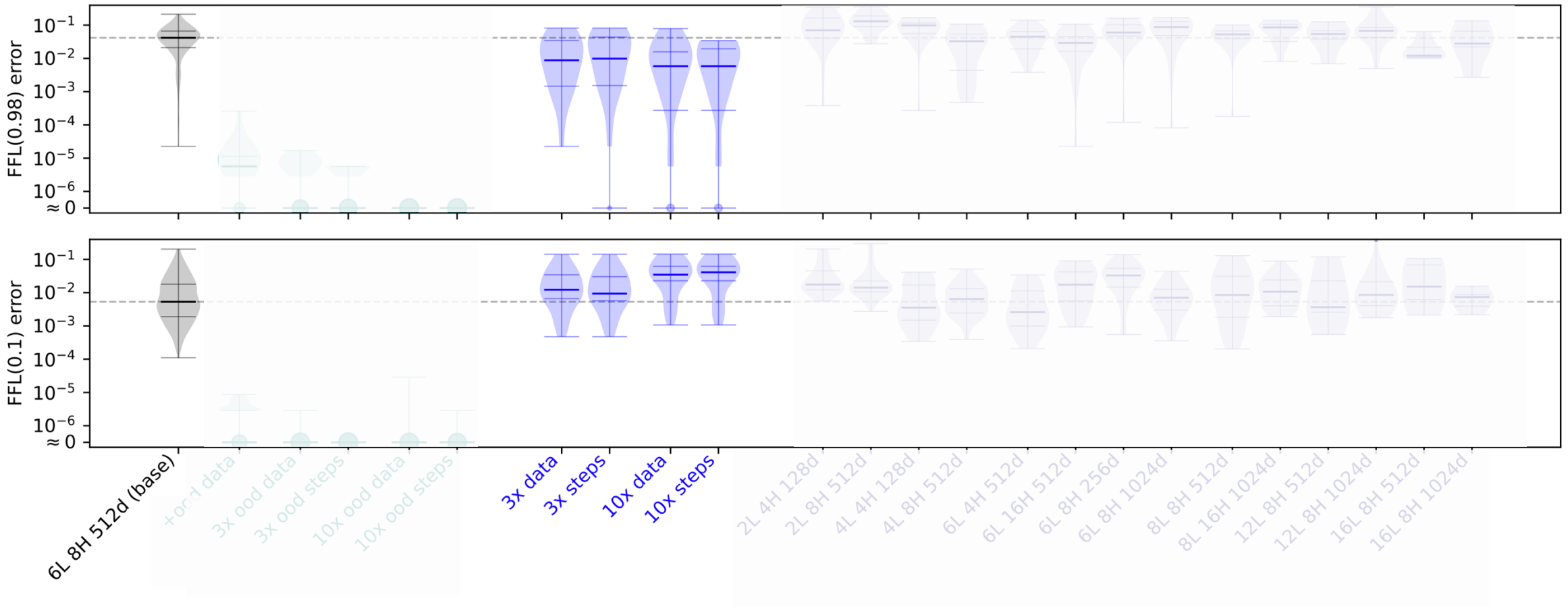
(R4) Incorporating OOD data works the best, by far.



Mitigations – Scaling

More data/iterations

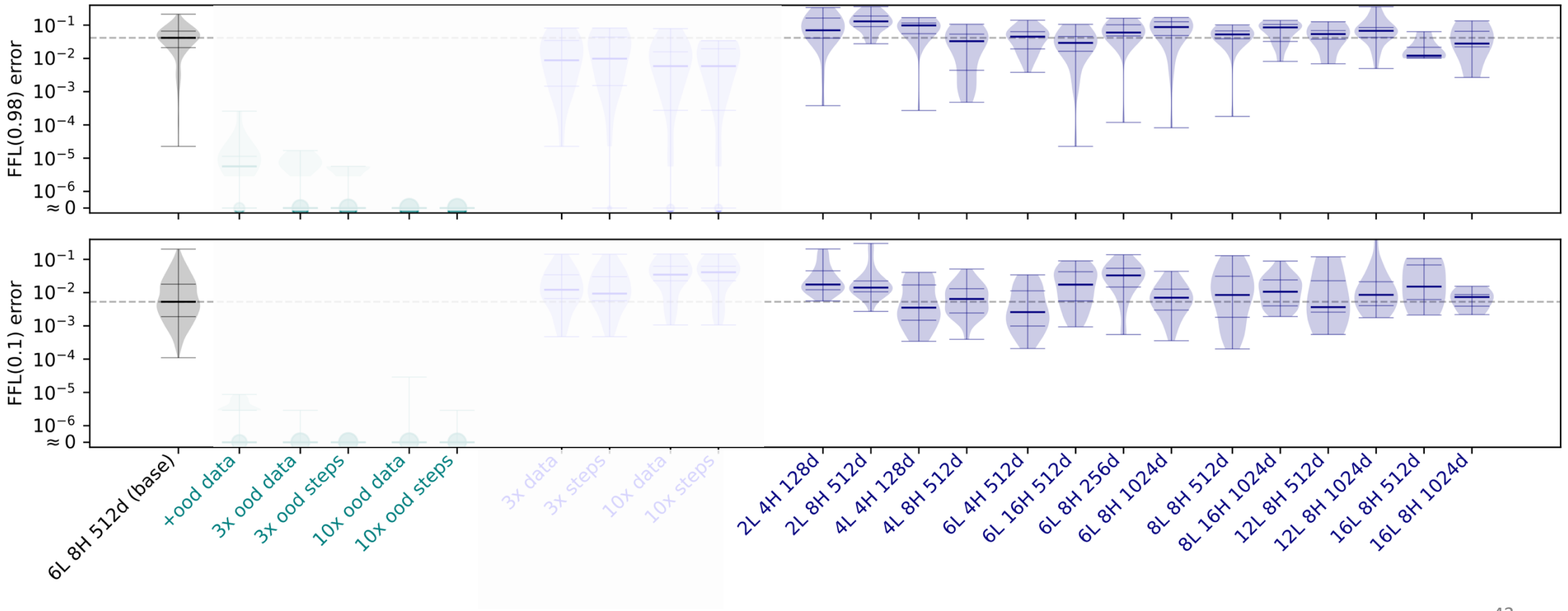
(R5) Resource scaling helps, but there's a **dense-sparse trade-off**.



Mitigations – Scaling

Larger models

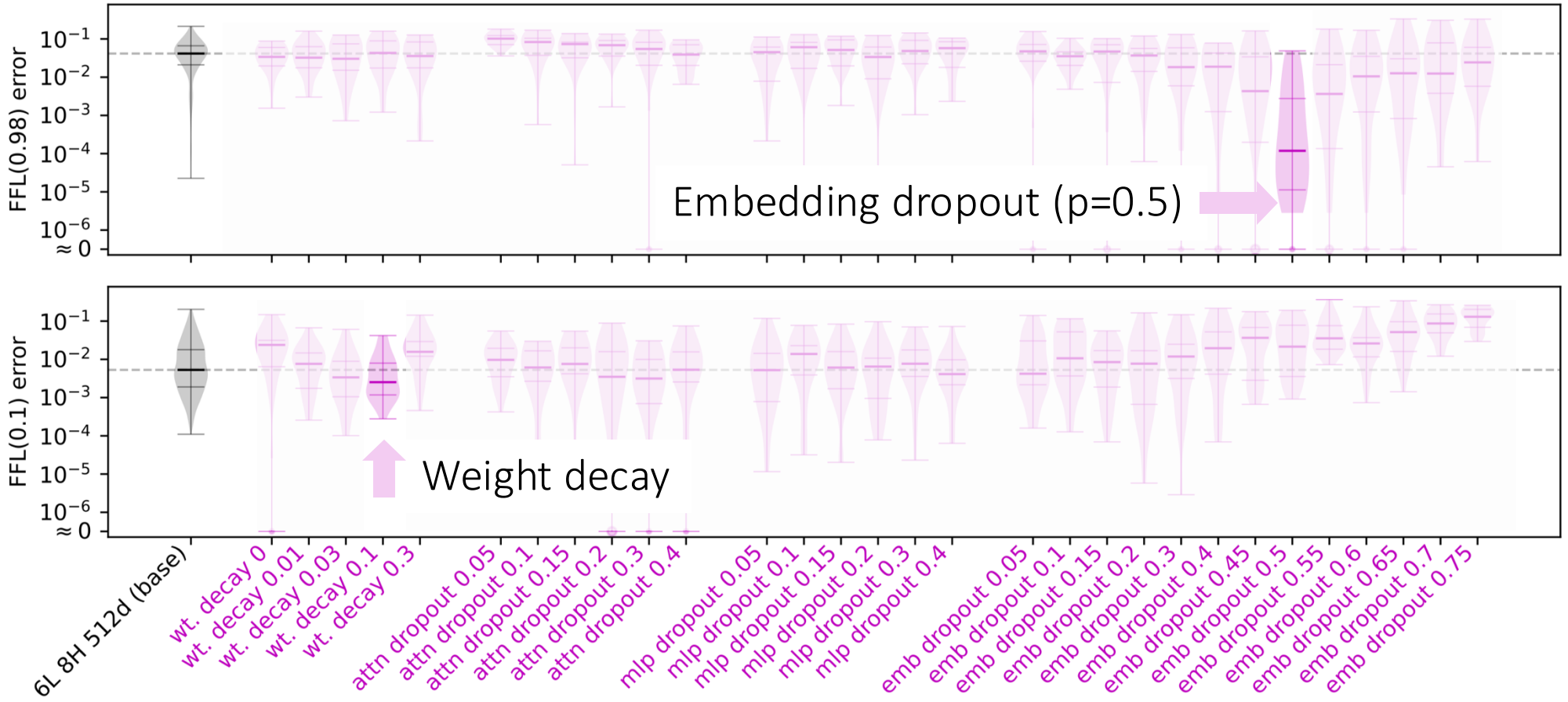
(R5) Resource scaling helps, but there's a **dense-sparse trade-off**.



Mitigations – Regularization

- Weight decay
- Dropout (attention, MLP, embedding)

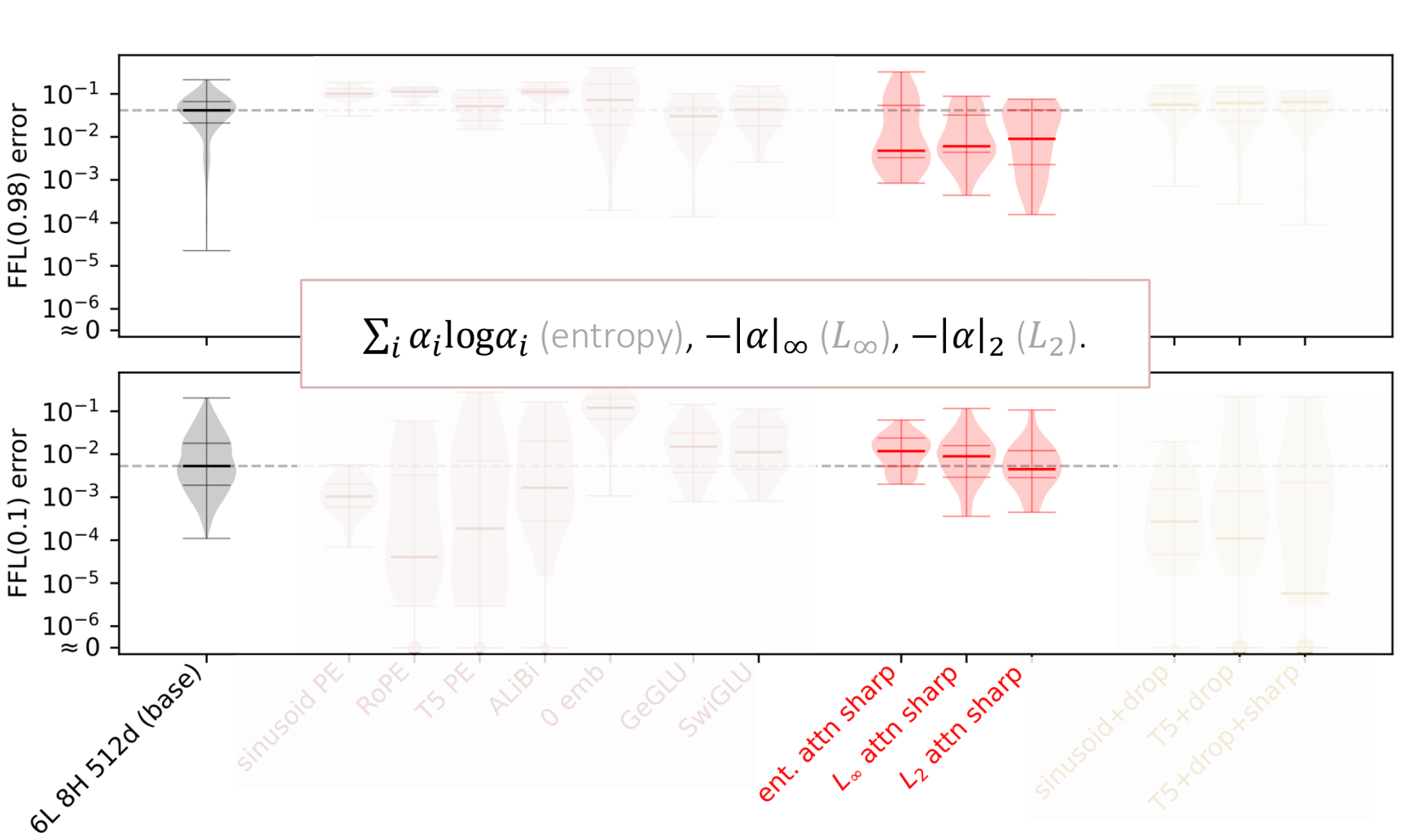
(R6) Standard regularizations have various influences.

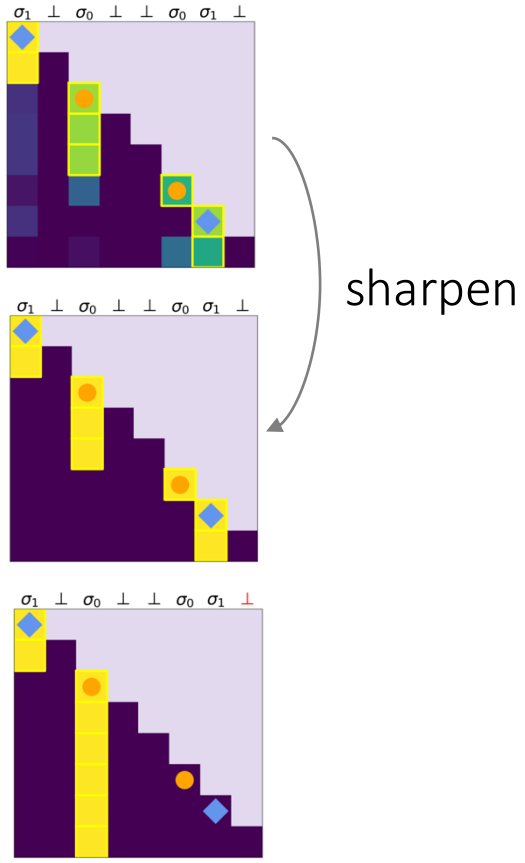


Mitigations – Regularization

Attention-sharpening regularization

(R6) Standard regularizations have various influences.

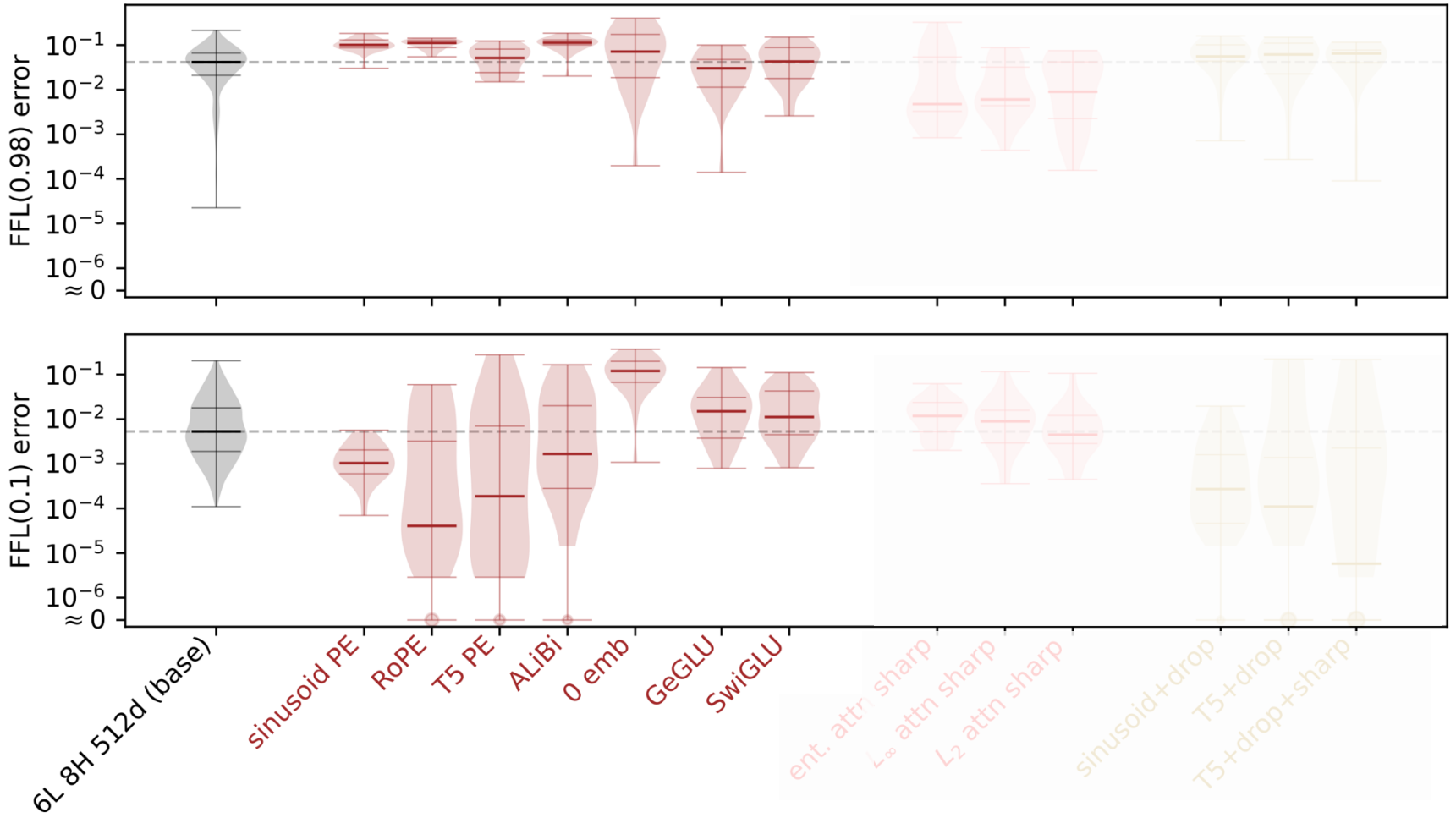




Mitigations – Architectural details

- Positional encoding
- Activation function

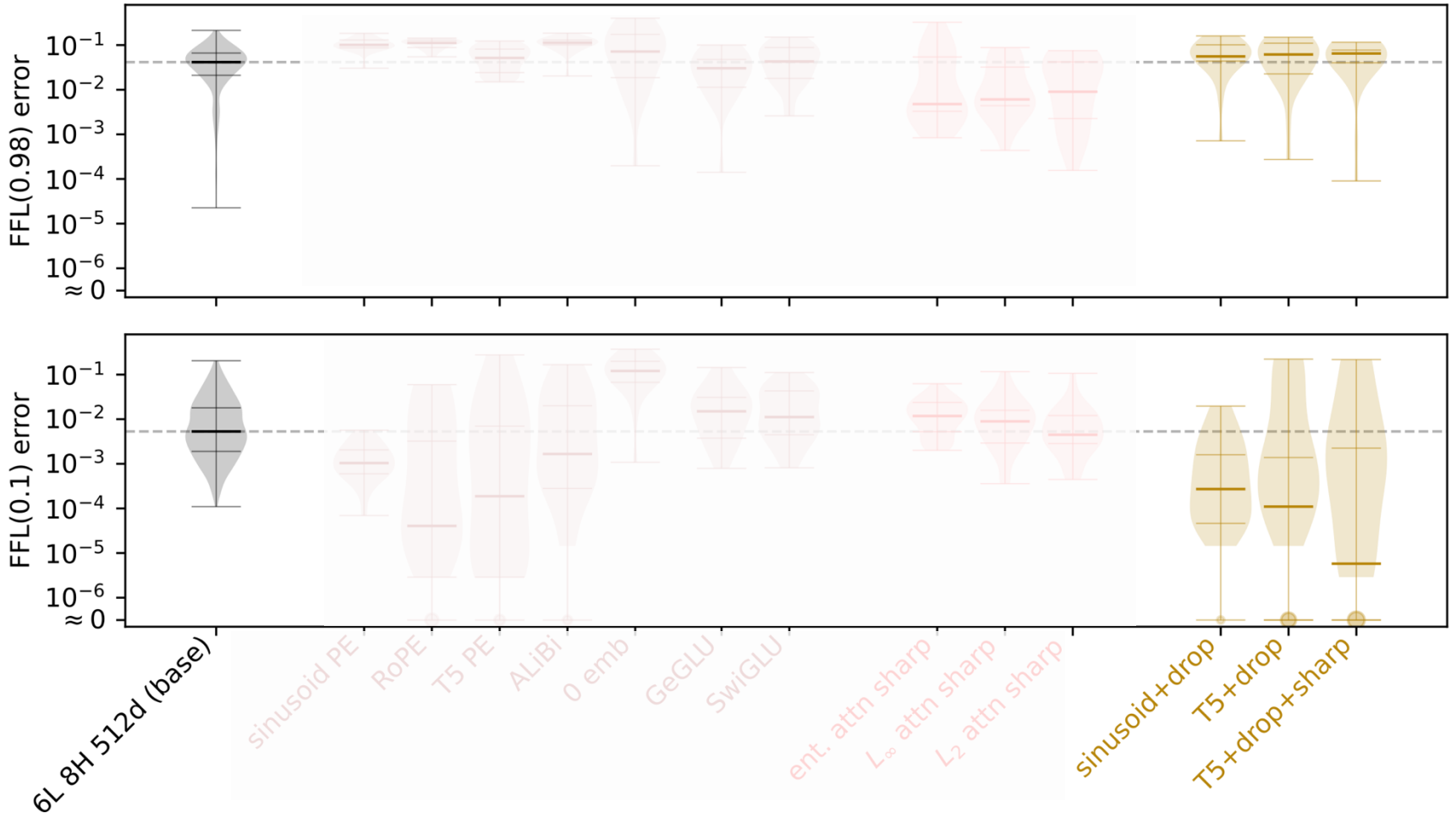
(R6) Standard regularizations have various influences.



Mitigations – Combination?

Combining multiple indirect controls

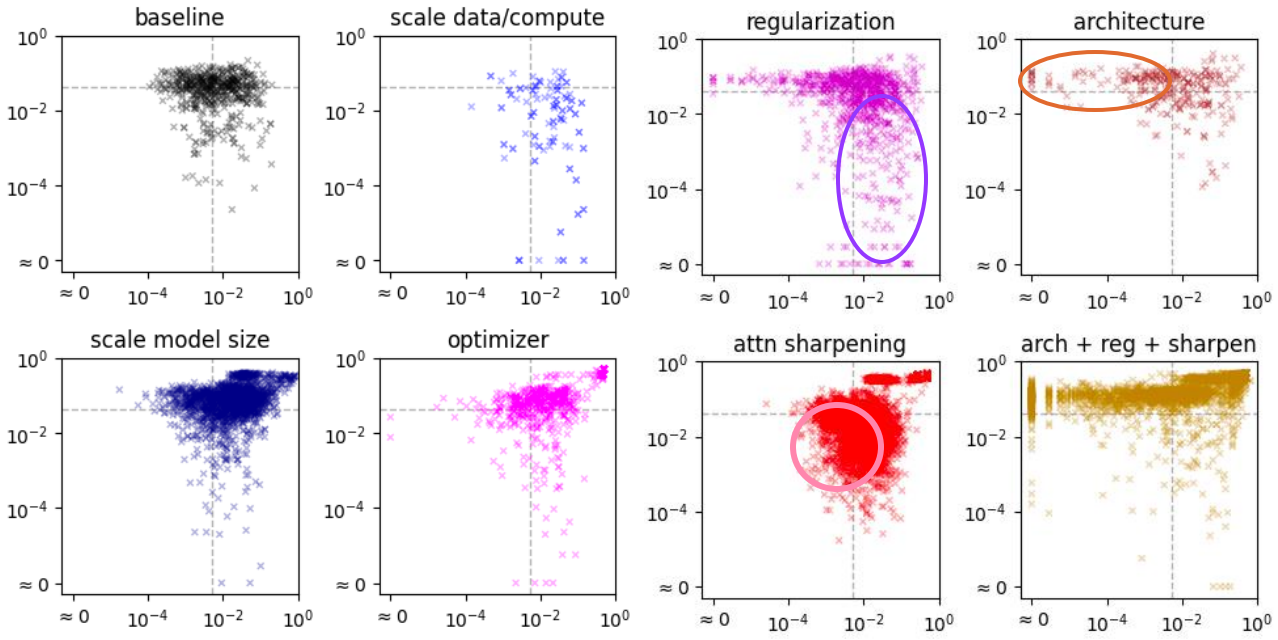
(R6) Standard regularizations have various influences.

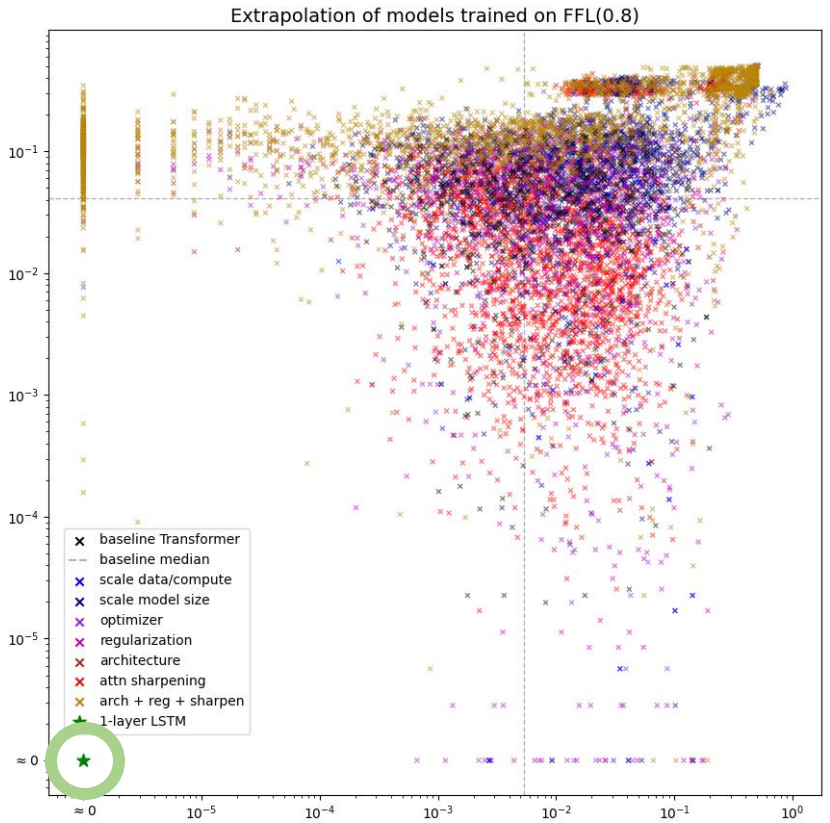


Trade-off: 2 types of OOD errors

Dense vs sparse: seldom improve both.

dense = x-axis, sparse = y-axis.





LSTM