

ACAS

Adaptive Control of Inference-Time Activation Sparsity for On-Device LLMs

Ethan Lin¹, Youngmin Yi², Hoeseok Yang¹

¹ Santa Clara University, Santa Clara, CA, USA

² Sogang University, Seoul, Republic of Korea



Code
Available



Code
Reviewed



Code
Reproducible



Santa Clara
Engineering



IEEE/ACM International Conference on Codesign of Embedded Systems (CODES) 2026

ESWEEK, Barcelona, Spain, October 5, 2026



Agenda

- 1 Motivation
- 2 Background
- 3 Proposed Technique: ACAS
- 4 Experimental Results
- 5 Conclusion

Agenda

Motivation

Background

Proposed Technique: ACAS

Experimental Results

Conclusion

AI Is Moving onto Devices



Phones



Cars



Robots



Edge boards



Private



Works offline



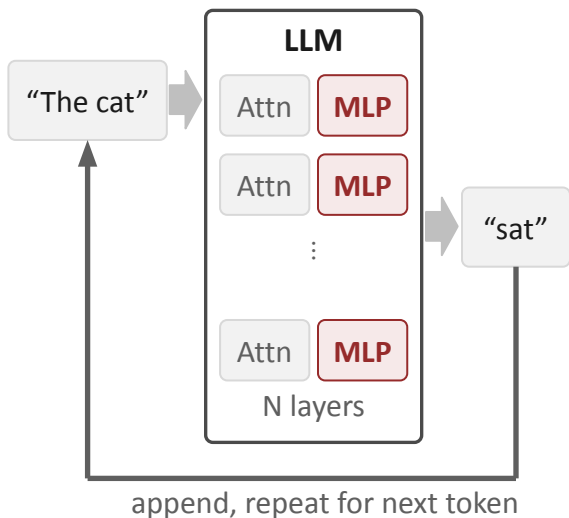
Instant



No cloud bill

But devices have little memory bandwidth and power.

Why It's Slow



An LLM writes one word at a time, and for every word it re-reads the whole model from memory.



Like re-reading an entire encyclopedia to write each word.

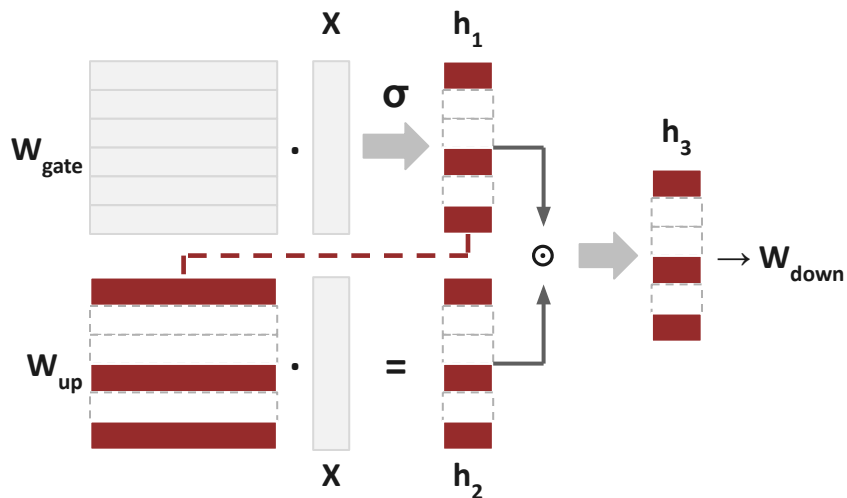
≈12 words/s

7B model on a Jetson AGX Orin

Memory-bound: memory accesses, not computation, sets the speed.

Most of That Reading Is Wasted

For each word, most MLP neurons output ≈ 0 and contribute nothing. Skip them: read less, go faster.



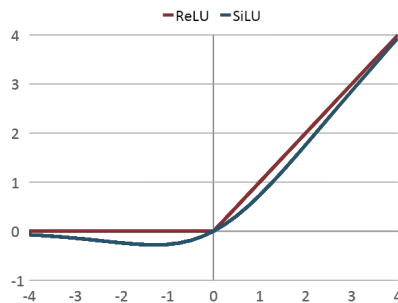
This is **activation sparsity**.

62%

of decode time is the MLP
Llama2-13B, Jetson Orin (SparseInfer)

~90%

of MLP activations are zero
ProSparse-LLaMA-2-7B (ReLU)



ReLU $\rightarrow$ exact 0

SiLU $\rightarrow$ near 0

Agenda

Motivation

Background

Proposed Technique: ACAS

Experimental Results

Conclusion

Existing Methods Guess What to Skip

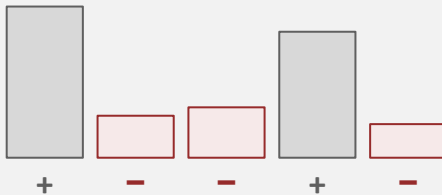
SparseInfer DATE '25



3 negative > 2 positive $\rightarrow$ skip

Guess from the signs.

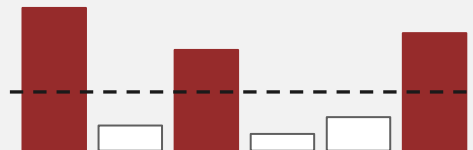
Grasp DAC '25



big positives win $\rightarrow$ keep

Signs, weighted by size.

CATS COLM '24



below the line $\rightarrow$ set to 0 and skip

Cut values below a threshold.

All three predict which neurons are ≈ 0 , then skip their weights.

Every Method Has a Hyperparameter

SparseInfer: α Grasp: β CATS: θ



Skip less

accurate, but slower

Skip more

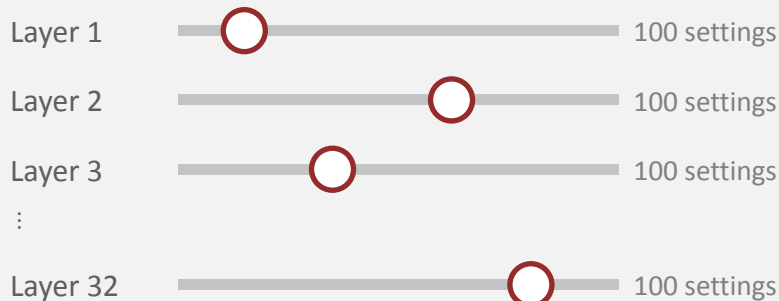
faster, but less accurate

It trades speed for accuracy, and the best setting for a given goal isn't obvious.

Today it is set offline: by search, calibration, or by hand.

Choosing the Hyperparameters Is Hard

32 layers *one hyperparameter each*



Real values are continuous: practically infinite.

Even with only 100 choices per layer:

$$100 \times 100 \times \dots \times 100 \\ = 100^{32} = 10^{64}$$

possible configurations

Exhaustive search is infeasible.

✗ Heuristic search

approximate, fixed per model
evolutionary search: R-Sparse (ICLR '25)

✗ Manual tuning

one value, picked by hand
SparseInfer (DATE '25)

✗ Calibration

one fixed setting per model
CATS (COLM '24), Grasp (DAC '25)

✗ Learned in training

fixed once trained
DejaVu (ICML'23), ProSparse (COLING'25)

Can these hyperparameters be tuned while the model runs?

Agenda

Motivation

Background

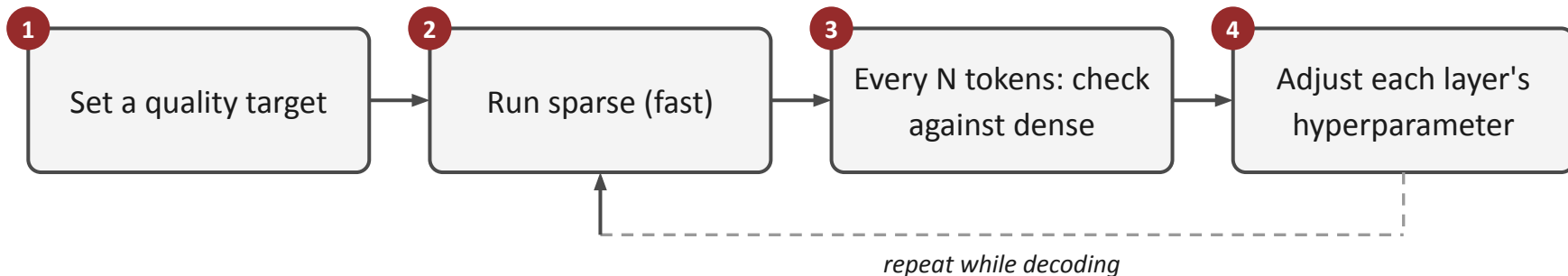
Proposed Technique: ACAS

Experimental Results

Conclusion

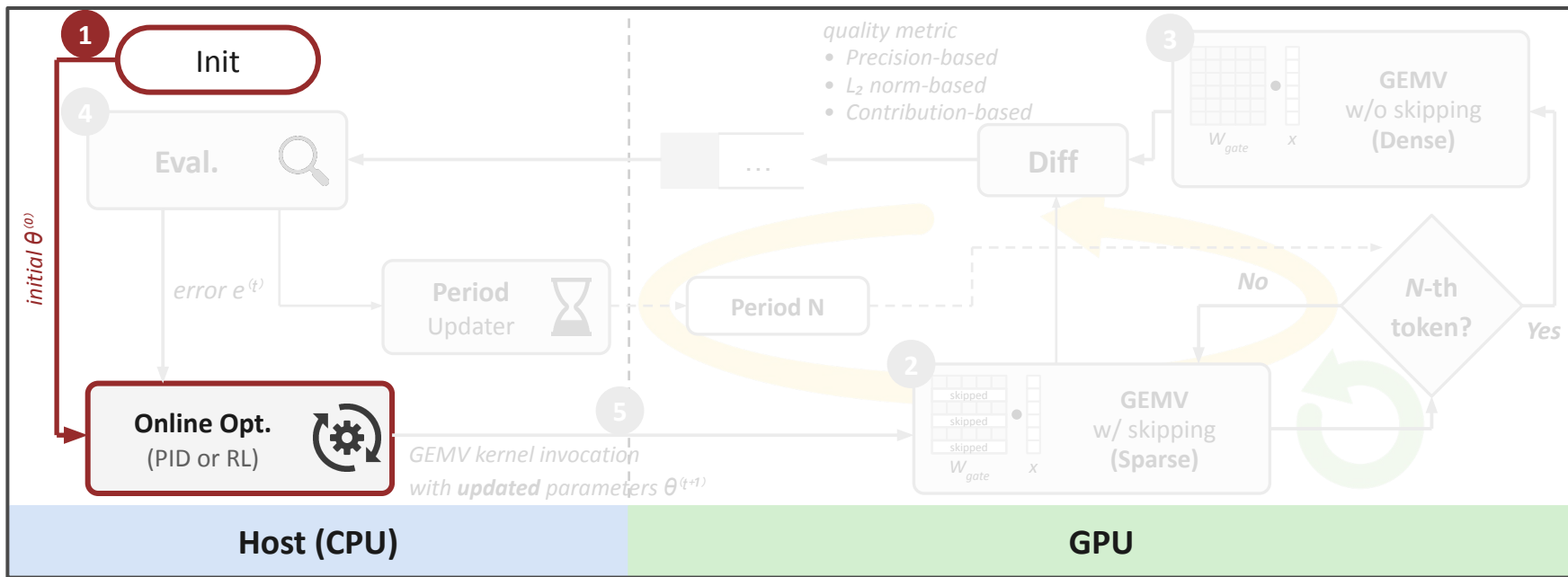
ACAS: A Thermostat for Sparsity

Like a thermostat: you set the temperature you want, and it keeps adjusting the heater. ACAS does this for sparsity.



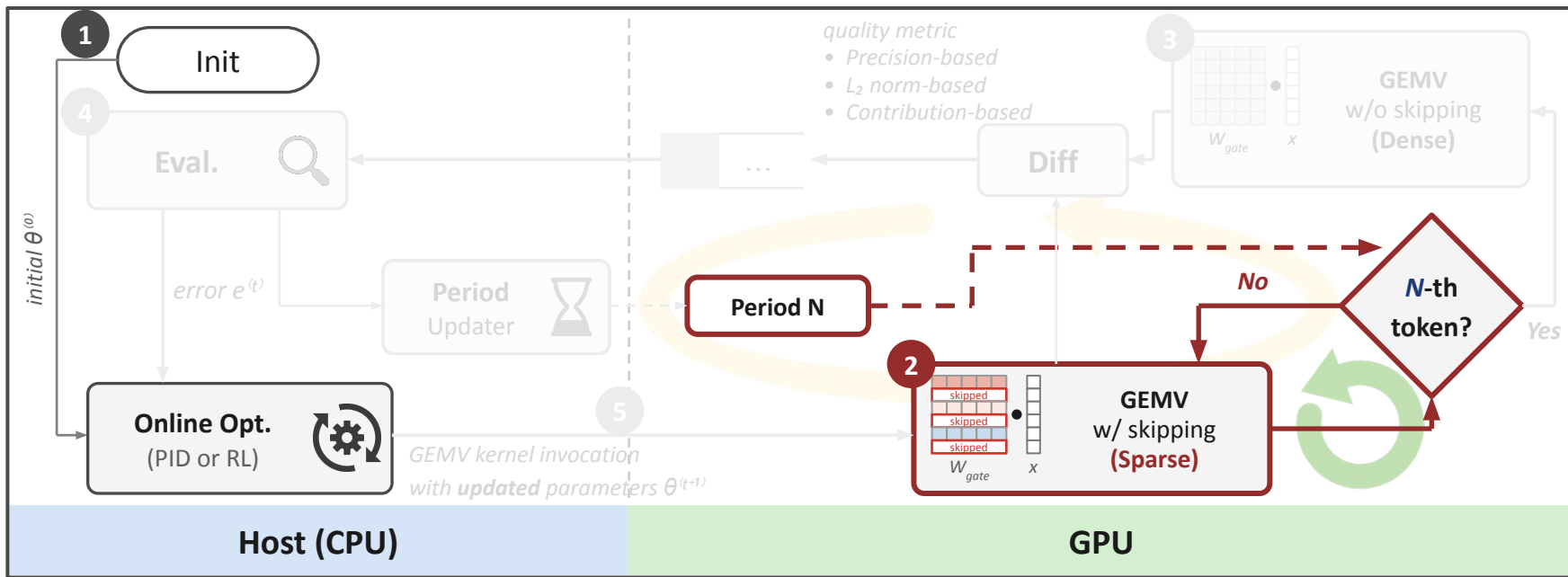
No offline tuning: the hyperparameters follow the workload, layer by layer.

ACAS Overview



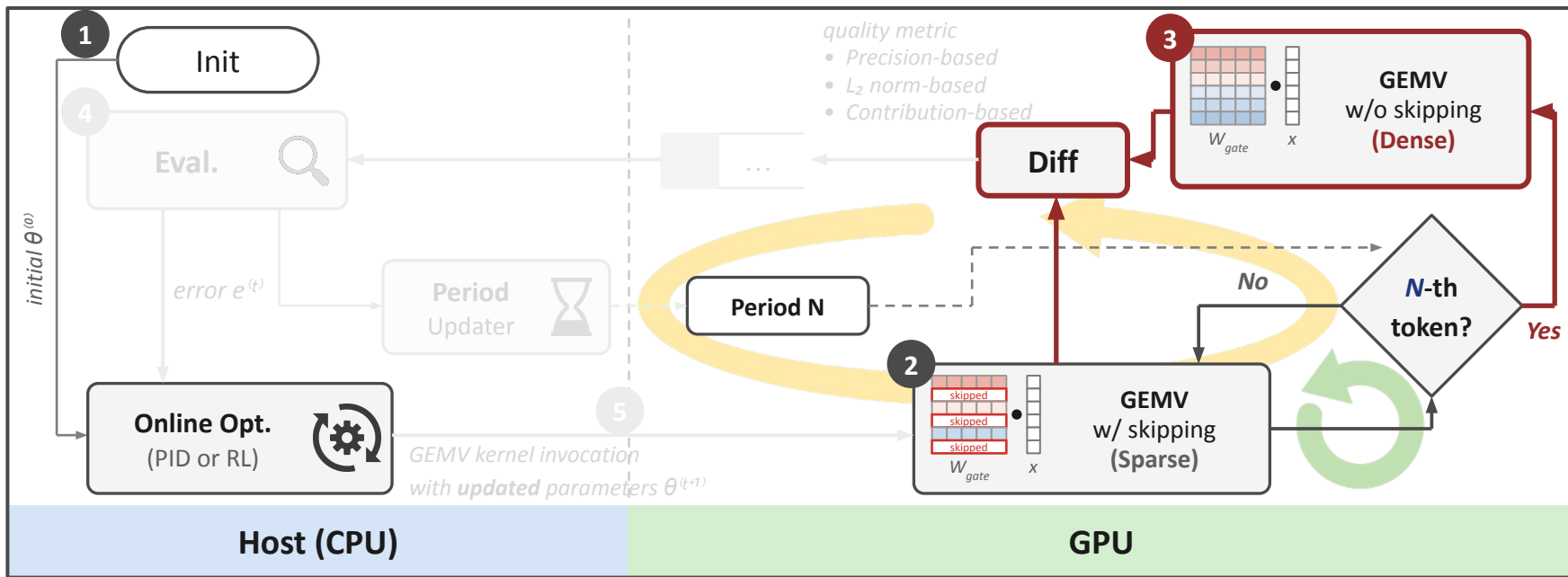
1 Start: load the initial hyperparameters.

ACAS Overview



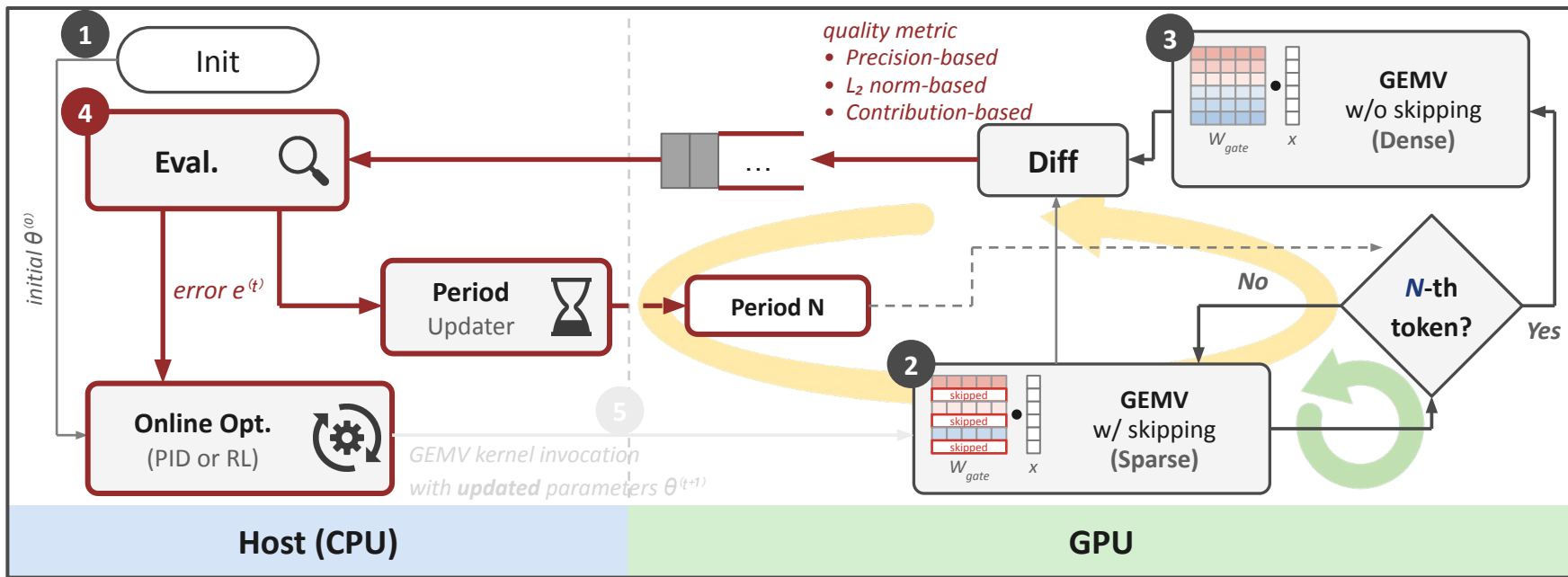
2 Every token: run the fast sparse kernel.

ACAS Overview



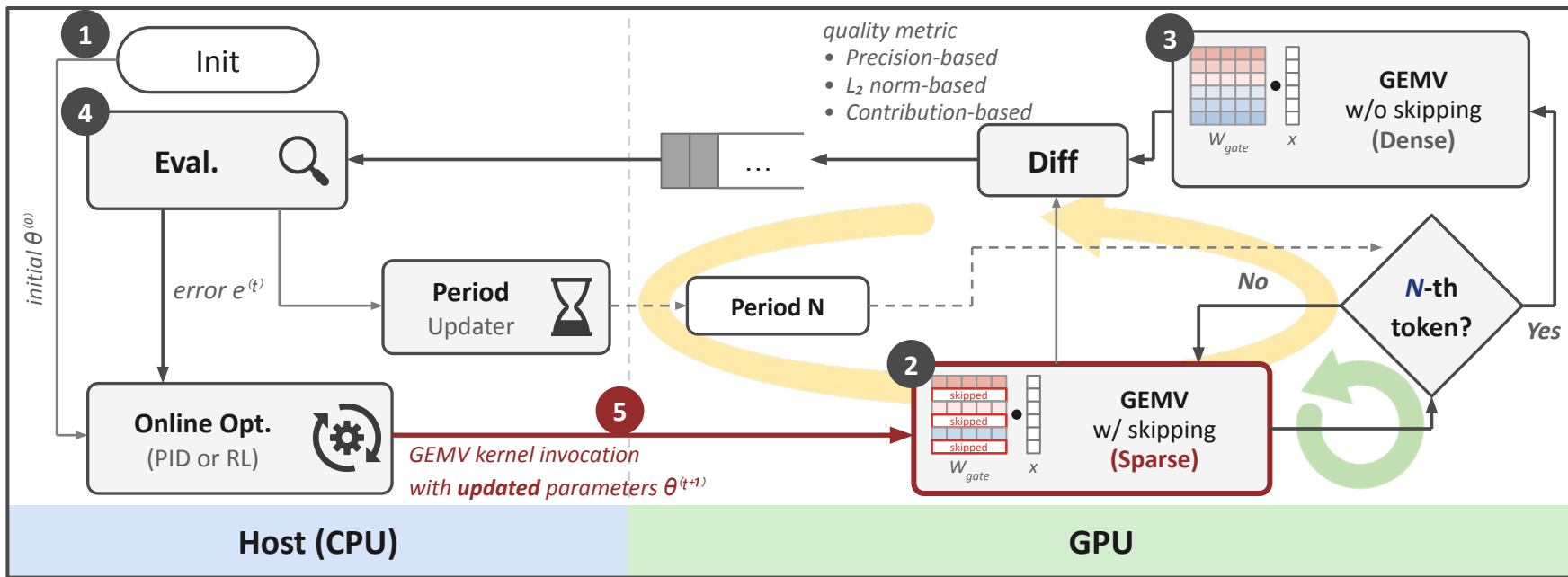
3 Every N-th token: also run dense and compare the two outputs.

ACAS Overview



4 On the CPU: turn the difference into an error and pick a new setting.

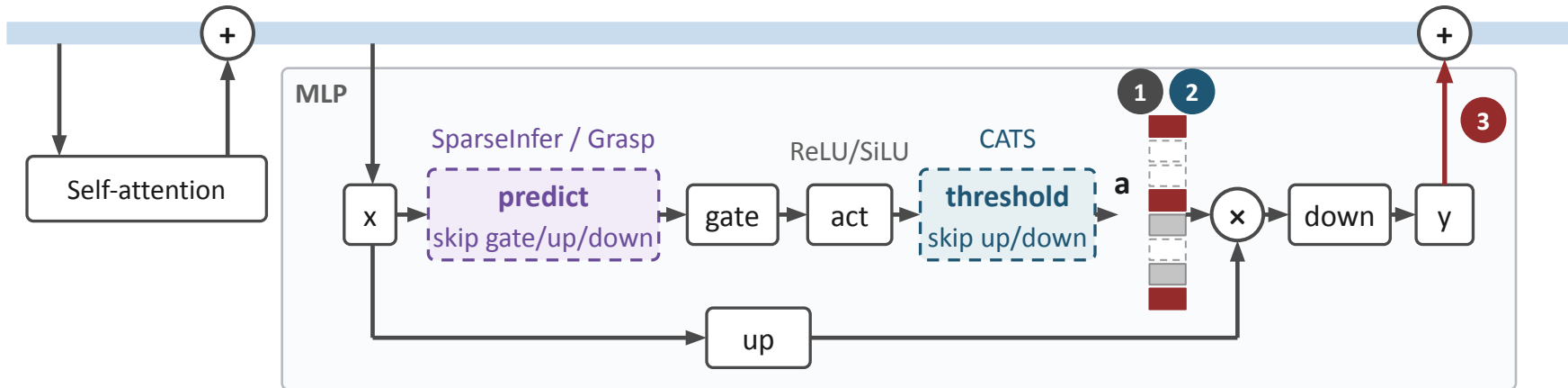
ACAS Overview



5 Repeat: the new hyperparameters go to the next tokens.

Quality Metrics

r = residual stream



1 Precision

Of the neurons skipped, how many were truly ≈ 0 ?

$$TP / (TP + FP)$$

2 Activation change

How much did skipping change the MLP activations?

$$1 - \|a - \hat{a}\| / \|a\|$$

3 Output change

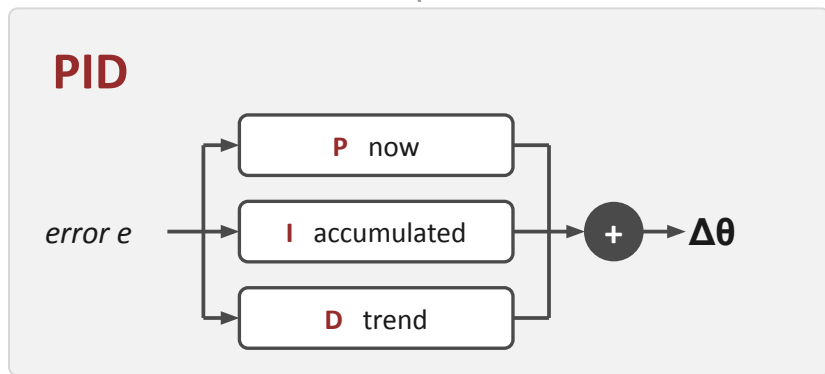
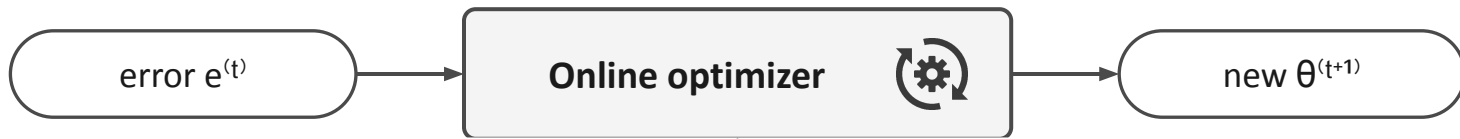
How much did skipping change what the MLP adds to the model?

$$1 - \|y - \hat{y}\| / \|r\|$$

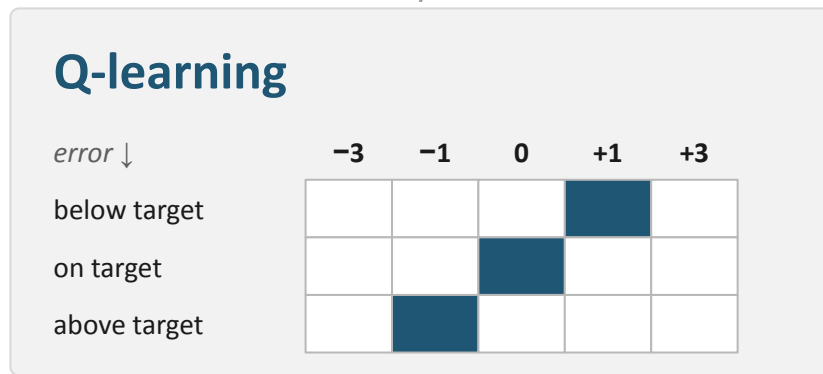
Non-intrusive: all three reuse the periodic dense check, so no extra pass is needed.

Online Optimizer

Either controller turns the error into the next hyperparameter.

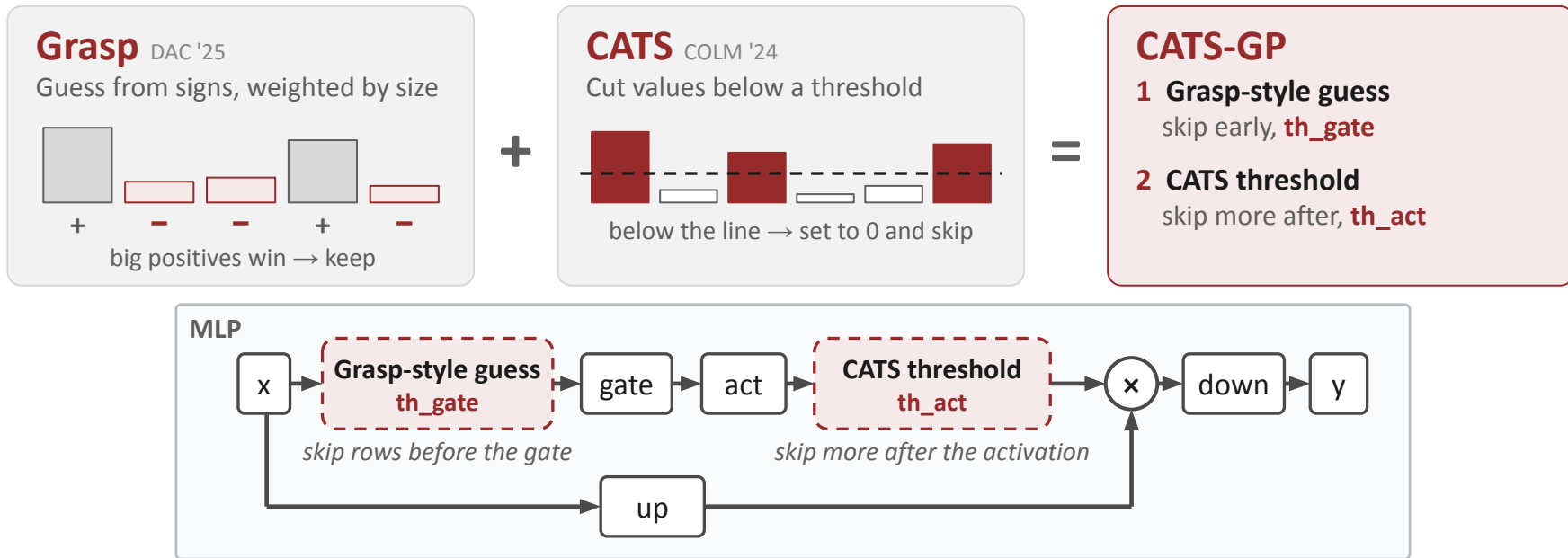


Fixed rule: reacts to the error now, its history, and its trend.



Learned rule: picks the step that worked best before for this error.

CATS-GP: Grasp + CATS



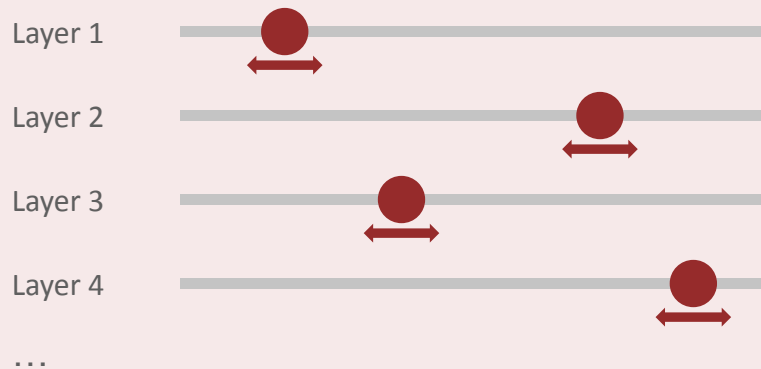
CATS-GP = Grasp's early skip + CATS's threshold, for SiLU models.

ACAS Works with Every Method

Fixed (today) *set once, offline*



With ACAS *moves while the model runs*



SparseInfer α · Grasp β · CATS th · CATS-GP th_gate, th_act

ACAS moves the sliders: each layer's hyperparameter, while the model runs.

Agenda

Motivation

Background

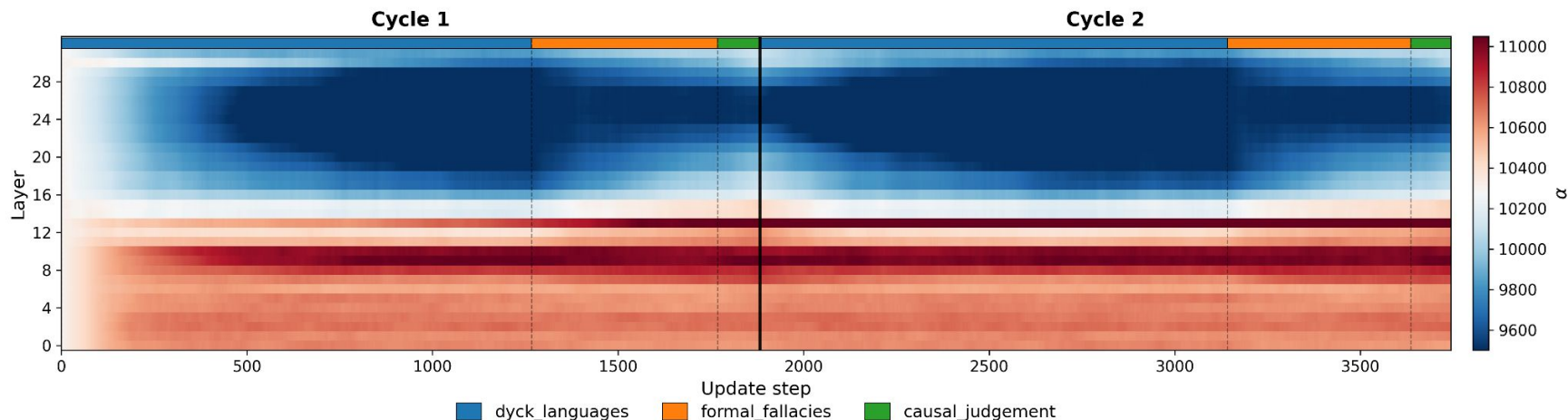
Proposed Technique: ACAS

Experimental Results

Conclusion

Why Runtime Adaptation Is Necessary

x: time y: layer top bar: current task color: ■ skips less ■ skips more

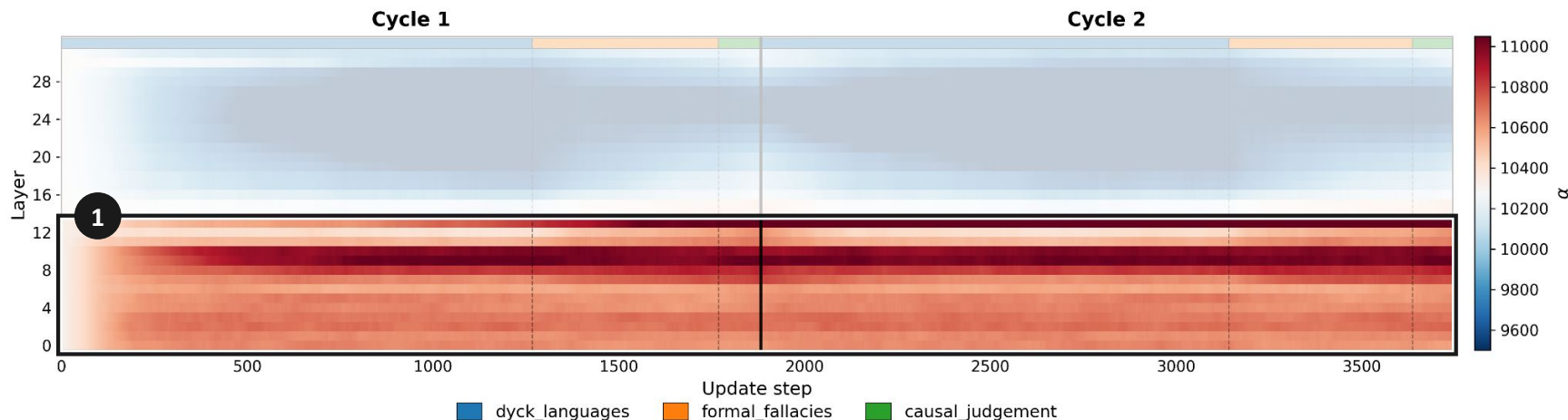


ACAS tunes α for every layer while it runs

Three BBH tasks, run twice. Every layer starts at the same α .

Why Runtime Adaptation Is Necessary

x: time y: layer top bar: current task color: ■ skips less ■ skips more

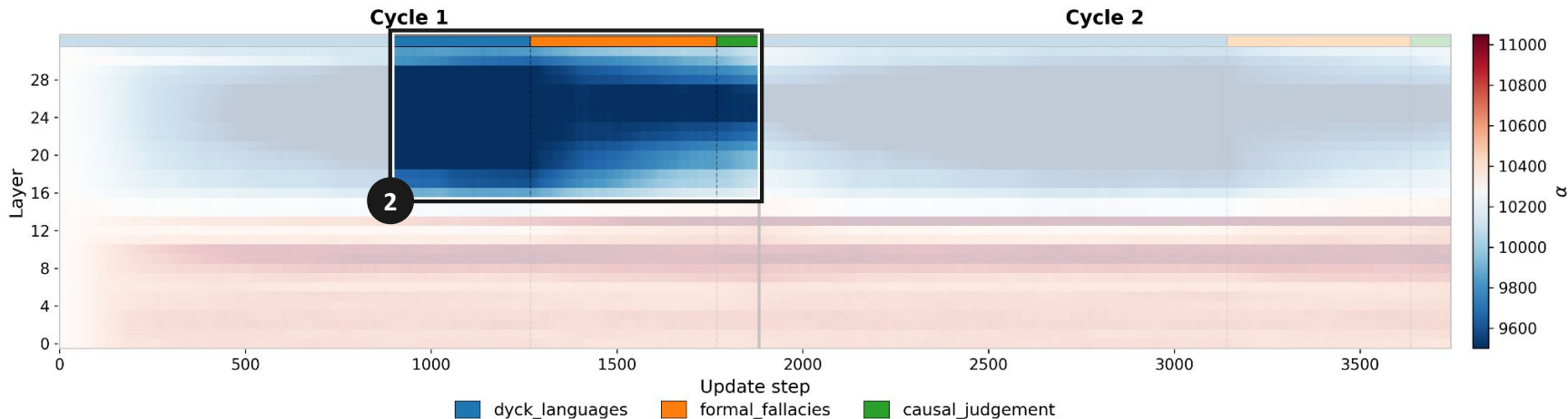


1 Layers need different settings

Early layers keep a high α and skip less; later layers skip more.

Why Runtime Adaptation Is Necessary

x: time y: layer top bar: current task color: ■ skips less ■ skips more

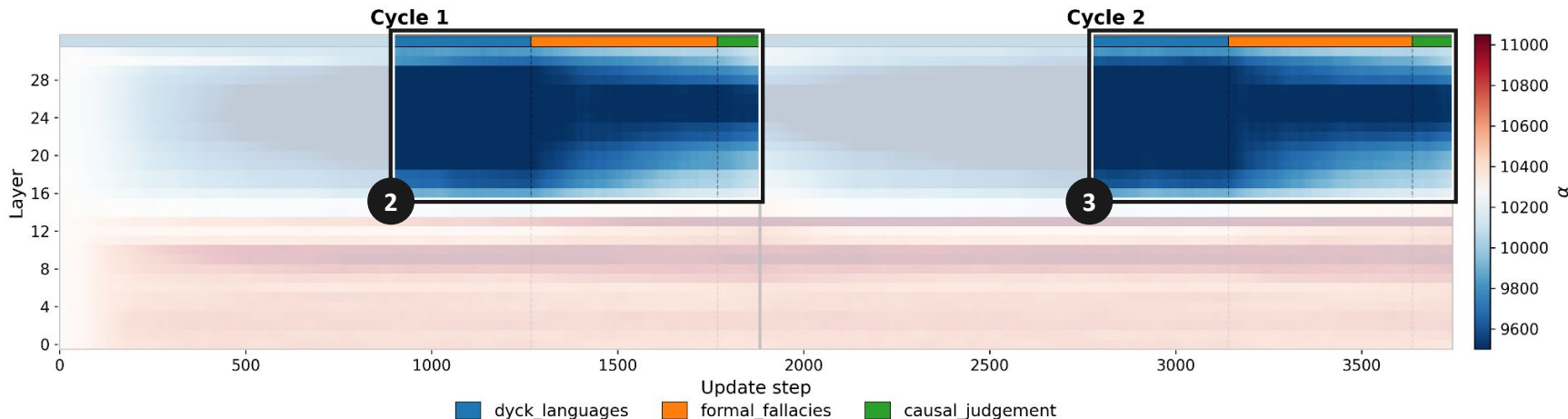


2 Tasks need different settings

When the task changes, the later layers become more careful.

Why Runtime Adaptation Is Necessary

x: time y: layer top bar: current task color: ■ skips less ■ skips more



3 ACAS finds them consistently

Same task again, same pattern: ACAS finds it every time.

No single fixed setting fits every workload.

Accuracy and Latency

ProSparse-LLaMA-2-7B (ReLU-fied)

ACAS-SparseInfer

vs. SparseInfer

Accuracy

25.45 → **26.17**

Speedup

1.54× → **1.57×**

ACAS setting: Prec. PID*

ACAS-Grasp

vs. Grasp

Accuracy

25.08 → **25.15**

Speedup

1.70× → **1.79×**

ACAS setting: Contrib. RL

LLaMA-3.1-8B (SiLU)

ACAS-CATS

vs. CATS-50%

Accuracy

60.85 → **67.99**

Speedup

1.10× → **1.10×**

ACAS setting: Contrib. PID

ACAS-CATS-GP

vs. CATS-50%

Accuracy

60.85 → **65.89**

Speedup

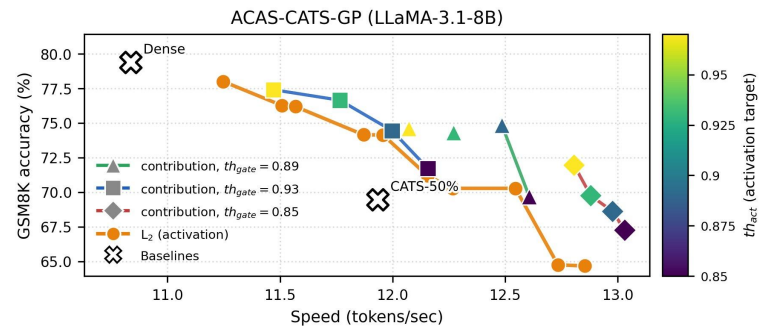
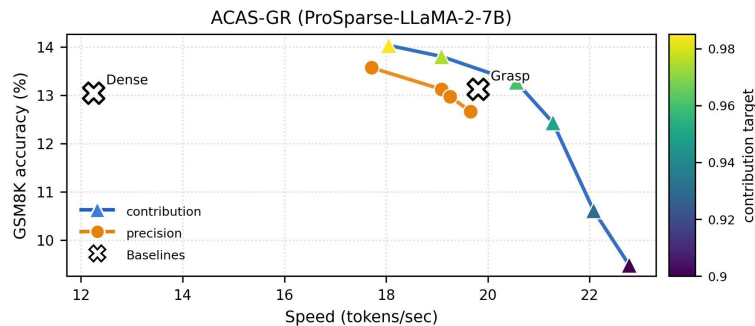
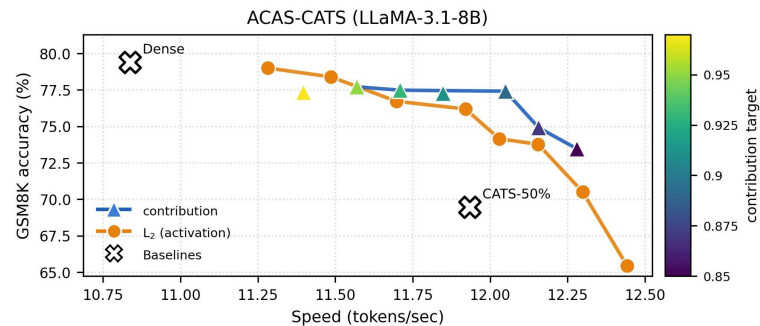
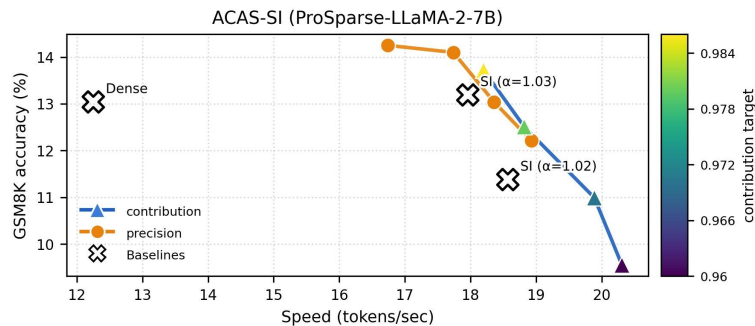
1.10× → **1.14×**

ACAS setting: Contrib. PID

Higher accuracy at the same or better speed, for all four methods.

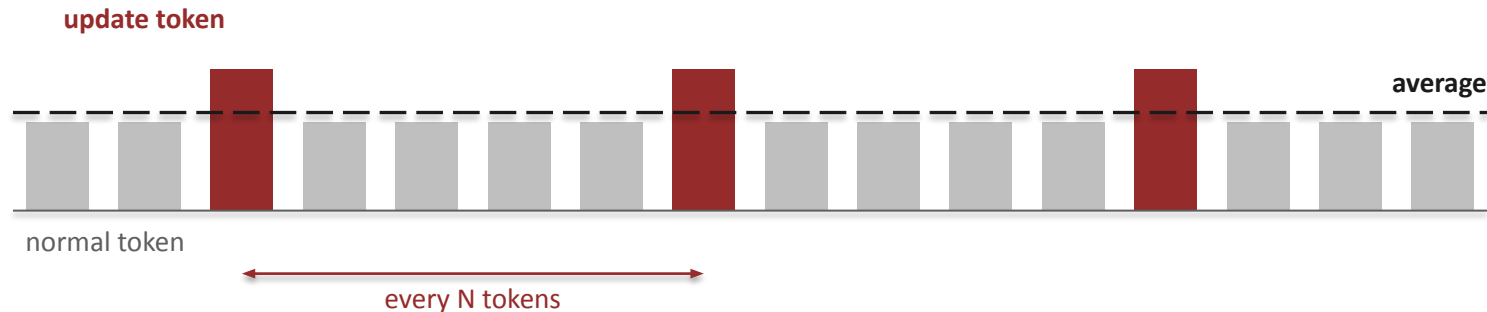
Acc. = average of GSM8K, BBH, and MBPP. Speedup vs. no sparsity, Jetson AGX Orin.

ACAS as a Design-Space Exploration Tool



Varying the quality target traces the speed and accuracy curve, beating every fixed setting.

Runtime Overhead: Small and Amortized



Update tokens cost $\sim 1.6\times$ (at most $2.4\times$), but come only once every N tokens.

Average latency $\approx$ static baselines, faster for 3 of 4 methods.

Power: comparable, no spikes at updates.

Agenda

Motivation

Background

Proposed Technique: ACAS

Experimental Results

Conclusion

Takeaways



Runtime adaptation is necessary

The best hyperparameters differ by layer and change with the task.



Offline tuning can't keep up → ACAS does it at runtime

Per-layer, per-task settings make offline search infeasible.
ACAS finds them during inference, no calibration.



A new design-space exploration tool

With no search burden, varying one quality target explores fine-grained, per-layer speed/accuracy trade-offs.

One quality target replaces all offline tuning.

Questions?

Code: github.com/SCU-CoAI/acas

Contact: enlin@scu.edu



Code
Available



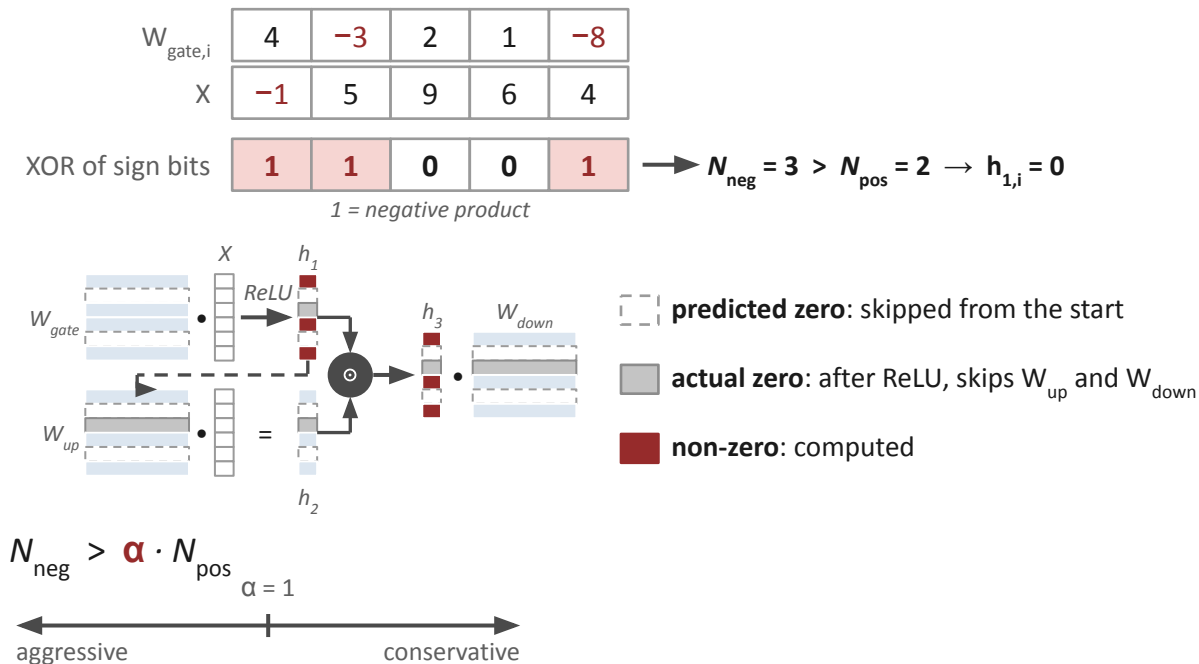
Code
Reviewed



Code
Reproducible

Appendix

SparseInfer



ReLU models

1 Sign bits, then XOR

XOR of the sign bits shows which products are negative.

2 Count and decide

Mostly negative products $\rightarrow$ ReLU will likely zero it, so skip the row.

3 The parameter α

α sets aggressiveness: lower α skips more, higher α skips less.

Grasp

						S			count	S · count
$W_{gate,i}$	1	-2	7	-1	6	G1	large × large	4	+2	+8
X	-2	1	5	5	4	G2	mixed	2	-1	-2
XOR of sign bits	1	1	0	1	0	G3	small × small	1	-2	-2
size group	G3	G3	G1	G2	G1					
bold = large		1 = negative product								
$N_{total} = \sum S \cdot \text{count}$						+4				
S: avg product vs. G3, calibrated offline										
count: $N_{pos} - N_{neg}$ within the group										

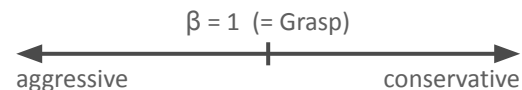
Signs only: 3 negative vs 2 positive → skip (SparseInfer)

Weighted by size: $N_{\text{total}} = +4 \rightarrow \text{keep}$ (Grasp)

Grasp: predict zero if $N_{\text{total}} < 0$

With β : predict zero if $N_{\text{total}} < (1 - \beta) \cdot d/2$

d = hidden dimension (length of X)



ReLU models

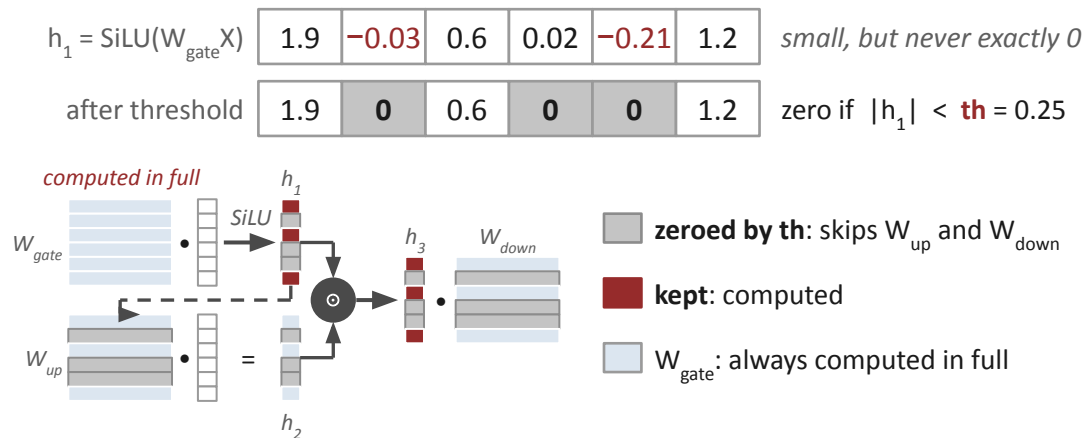
1 Magnitude groups

Products are grouped by size; each group's count is weighted by a scale S.

2 The parameter β

β sets aggressiveness: lower β skips more, higher β skips less.

CATS



CATS: th fixed offline, from calibration

CATS-50%: th chosen so half the values are zeroed

higher **th** → more zeros → more skipping

SiLU models

1 No zeros to predict

SiLU outputs are small, but never exactly zero, so there is nothing to predict.

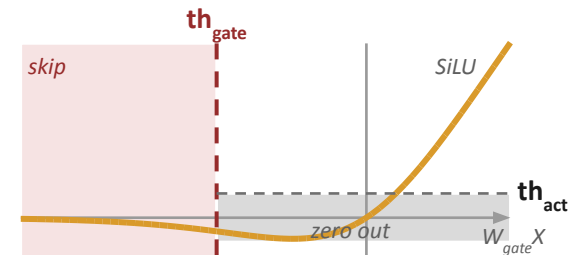
2 Threshold, then skip

The gate is computed in full; values under th are zeroed and their rows skipped.

3 The parameter th

th sets aggressiveness: higher th skips more, lower th skips less.

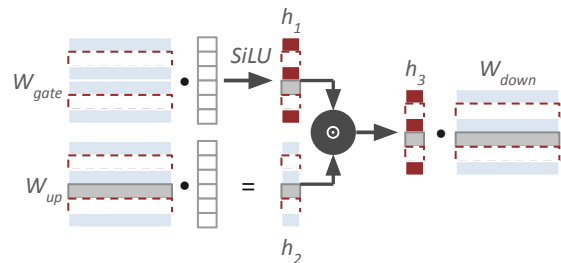
ACAS-CATS-GP



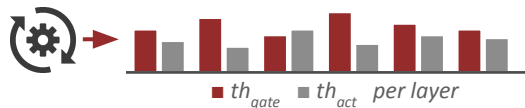
skipped by th_{gate} : gate, up, down

skipped by th_{act} : up, down

kept: computed



ACAS sets both, per layer



CATS-GP = CATS + a threshold before the gate

SiLU models

1 Skip before the gate

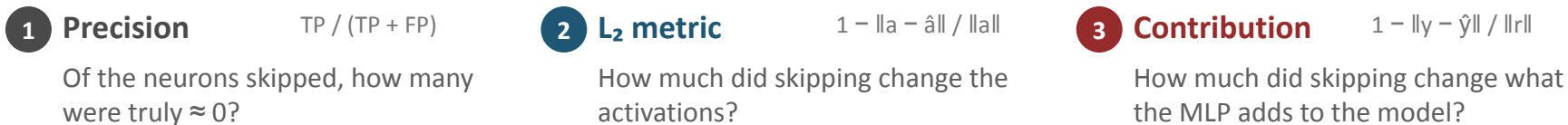
Grasp's group-based estimate is a proxy for the gate output. Below th_{gate} : skip gate, up, down.

2 Skip after the gate

Otherwise compute the gate. If its output is below th_{act} : skip up, down (as in CATS).

3 ACAS controls both

th_{gate} and th_{act} , per layer, every N tokens.



 predicted non-zero
  predicted sparse (skipped)
  actual sparse (found after act)
 $\hat{a}, \hat{y}$ = with neurons skipped

Experimental Setup

	Details
Hardware	NVIDIA Jetson AGX Orin (latency, power, energy); batch size 1, decode
Power	On-board INA3221 sensor, sampled every 25 ms; energy/token = power $\times$ TPOT
Models	ProSparse-LLaMA-2-7B (ReLU-fied), LLaMA-3.1-8B (SiLU)
Benchmarks	GSM8K (math), BBH (reasoning), MBPP (code); Acc. = average of the three
Baselines	No sparsity; SparseInfer ($\alpha = 1.03$); Grasp; CATS-50%
ACAS signal	Precision or contribution (ReLU); L_2 or contribution (SiLU)
Optimizer	PID or tabular Q-learning (RL)
Update period	Every N tokens; * = adaptive period with $N_{\min} / N_{\max} = 50 / 500$

Accuracy and Latency

ProSparse-LLaMA-2-7B (ReLU-fied)

Method	Metric / optimizer	Acc.	TPOT ms (speedup)
No sparsity		25.98	84.1 (1.00×)
SparseInfer		25.45	54.5 (1.54×)
ACAS-SI	Prec. PID	26.07	54.2 (1.55×)
	Prec. PID*	26.17	53.7 (1.57×)
	Contrib. PID	25.86	51.9 (1.62×)
	Contrib. PID*	25.98	52.3 (1.61×)
	Contrib. RL	25.98	52.2 (1.61×)
	Contrib. RL*	25.33	53.1 (1.59×)
Grasp		25.08	49.6 (1.70×)
ACAS-GR	Prec. PID	25.55	54.0 (1.56×)
	Prec. PID*	25.50	52.9 (1.59×)
	Contrib. PID	24.87	47.2 (1.78×)
	Contrib. PID*	24.49	47.5 (1.77×)
	Contrib. RL	25.15	47.1 (1.79×)
	Contrib. RL*	23.82	48.1 (1.75×)

ACAS-SI: 5 of 6 settings beat SparseInfer on both

ACAS-GR: up to +0.5 pts over Grasp; Contrib. RL beats it on both

static baseline

better than its baseline

bold best sparse setting

* adaptive update period

Acc. average of GSM8K, BBH, and MBPP

TPOT time per output token, with speedup over no sparsity

LLaMA-3.1-8B (SiLU)

Method	Metric / optimizer	Acc.	TPOT ms (speedup)
No sparsity		69.54	92.3 (1.00×)
CATS-50%		60.85	83.8 (1.10×)
ACAS-CATS	L ₂ PID	67.49	85.1 (1.08×)
	L ₂ PID*	67.06	85.2 (1.08×)
	Contrib. PID	67.99	83.8 (1.10×)
	Contrib. PID*	66.60	85.6 (1.08×)
	Contrib. RL	68.35	84.6 (1.09×)
	Contrib. RL*	67.92	85.9 (1.07×)
ACAS-CATS-GP	L ₂ PID	62.40	83.4 (1.11×)
	L ₂ PID*	61.84	82.6 (1.12×)
	Contrib. PID	65.89	81.0 (1.14×)
	Contrib. PID*	62.67	81.6 (1.13×)
	Contrib. RL	66.40	82.1 (1.12×)
	Contrib. RL*	62.99	82.8 (1.11×)

ACAS-CATS: 5.8–7.5 pts above CATS-50%, at most 2.1 ms slower per token

ACAS-CATS-GP: all 6 settings beat CATS-50% on both

Runtime Overhead

Latency per token (ms)

	Regular	Update	Worst
ACAS-SI (Prec.)	54.1	75.0 (1.39×)	82.7 (1.53×)
ACAS-SI (Contrib.)	51.5	109.8 (2.13×)	118.1 (2.29×)
ACAS-GR (Prec.)	53.9	71.5 (1.33×)	78.1 (1.45×)
ACAS-GR (Contrib.)	46.9	105.4 (2.25×)	111.8 (2.38×)
ACAS-CATS (L_2)	85.1	91.6 (1.08×)	94.1 (1.11×)
ACAS-CATS (Contrib.)	83.6	131.5 (1.57×)	133.9 (1.60×)
ACAS-CATS-GP (L_2)	83.2	107.7 (1.29×)	115.5 (1.39×)
ACAS-CATS-GP (Contrib.)	80.6	153.1 (1.90×)	155.6 (1.93×)

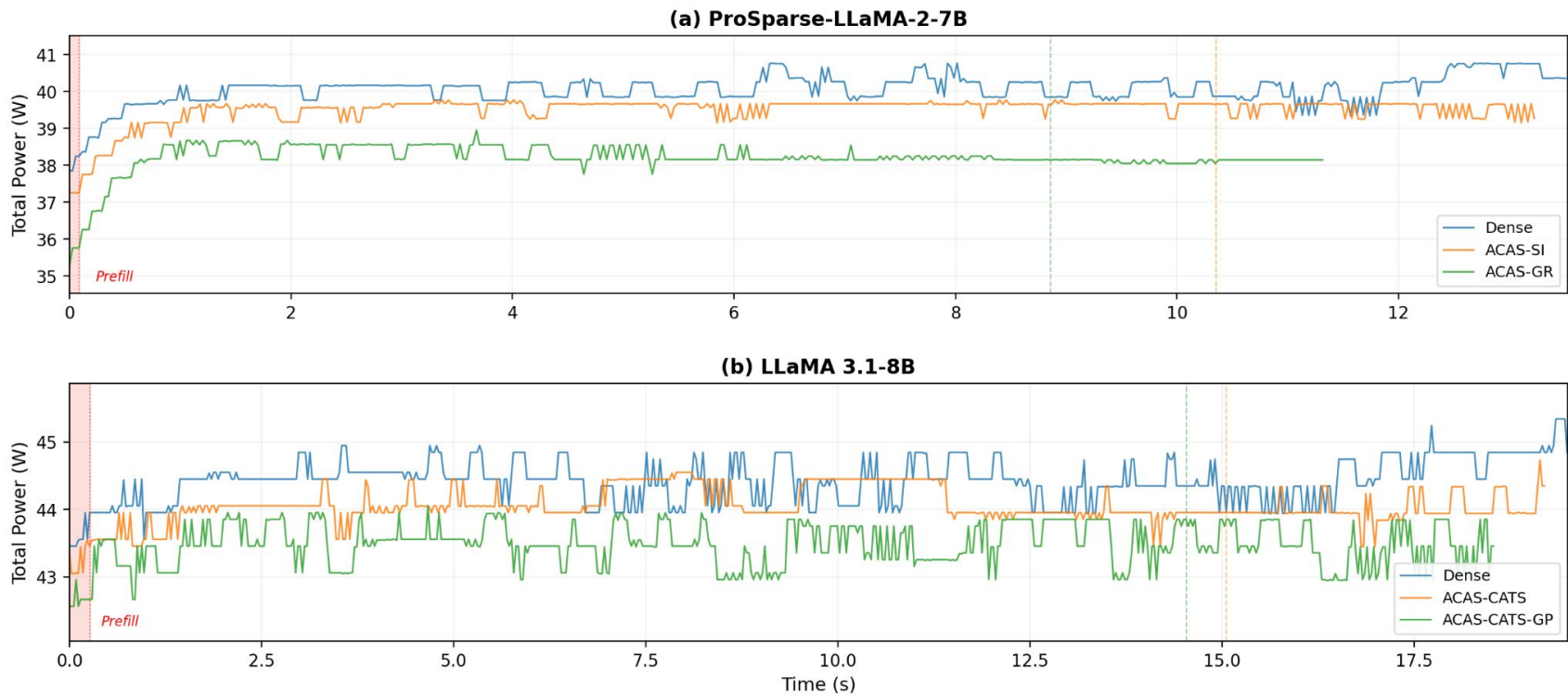
Slower update tokens, but rare

An update token takes 1.6× a regular token on average (2.4× at most). It happens once every N tokens, so average latency stays close to the baselines.

Power and energy

Comparable to the sparse baselines, with no power spikes at update events. Details in the paper.

Instantaneous Power Traces During Inference



Adaptive update period: sensitivity to its bounds

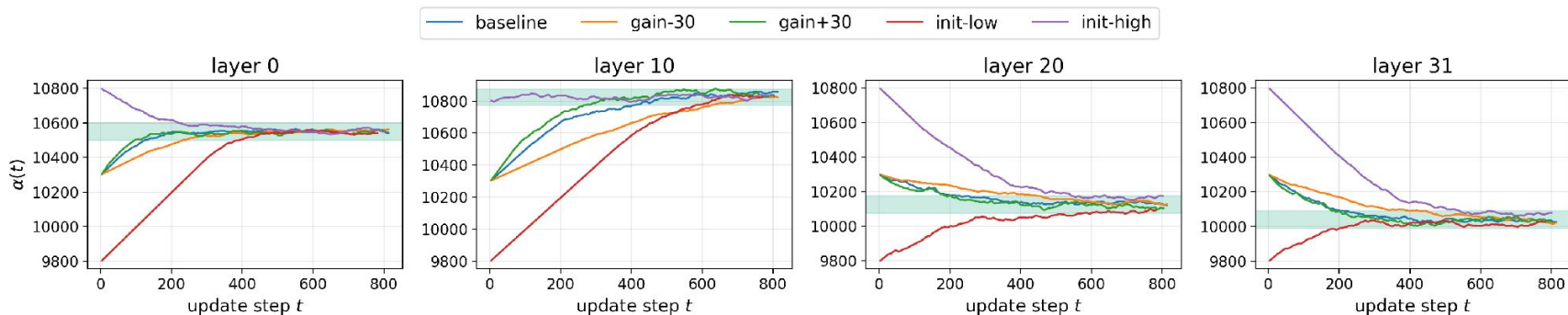
ProSparse-LLaMA-2-7B, ACAS-SI (Contrib., PID*)

	$N_{\min} / N_{\max}$	GSM8K	BBH	MBPP	Avg.	TPOT (ms)
Default	50 / 500	14.03	35.71	28.20	25.98	52.29
Vary $N_{\min}$	25 / 500	14.48	35.56	28.80	26.28	53.34
	100 / 500	14.78	35.99	28.60	26.46	52.21
Vary $N_{\max}$	50 / 250	13.72	35.57	29.00	26.10	52.80
	50 / 1000	14.25	35.69	28.40	26.12	52.95
	50 / 2000	14.86	35.49	29.20	26.52	52.84

Barely changes

Across all six settings, average accuracy stays within 0.5 points and latency within 1.1 ms.

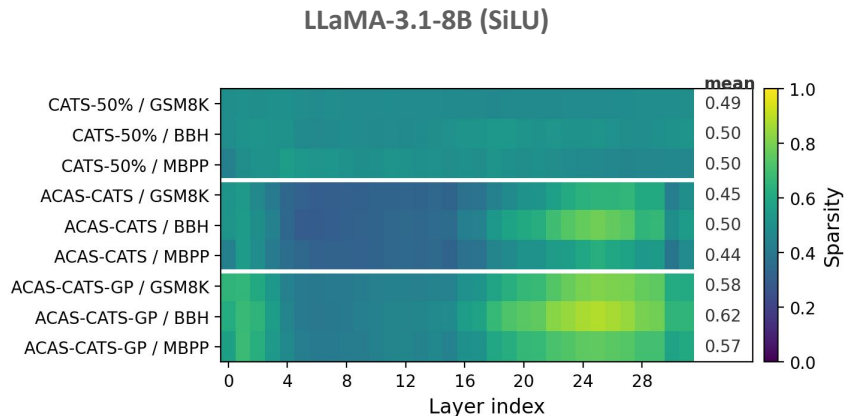
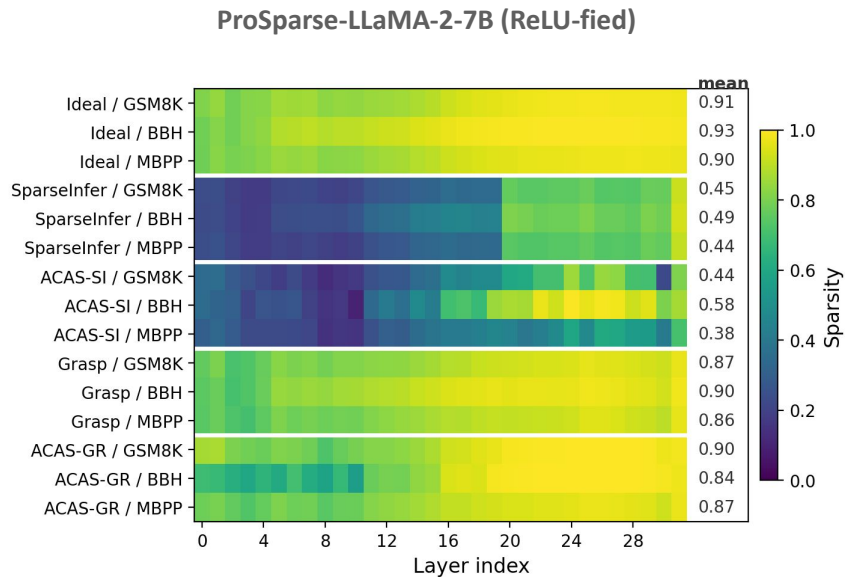
Controller robustness to gains and initial values



Per-layer $\alpha(t)$ for layers 0, 10, 20 and 31 under five conditions: baseline, all PID gains +30% and -30%, and low or high initial α .

Every condition converges into the same band, so the result does not depend on hand-tuned gains or starting values.

Where each method skips: sparsity by layer



Fraction of MLP neurons skipped per layer (right column: mean). ACAS spends its skips unevenly across layers instead of one fixed ratio.