AWQ and TinyChat: Efficient LLMs on the Edge

Ji Lin

jilin@mit.edu
https://linji.me

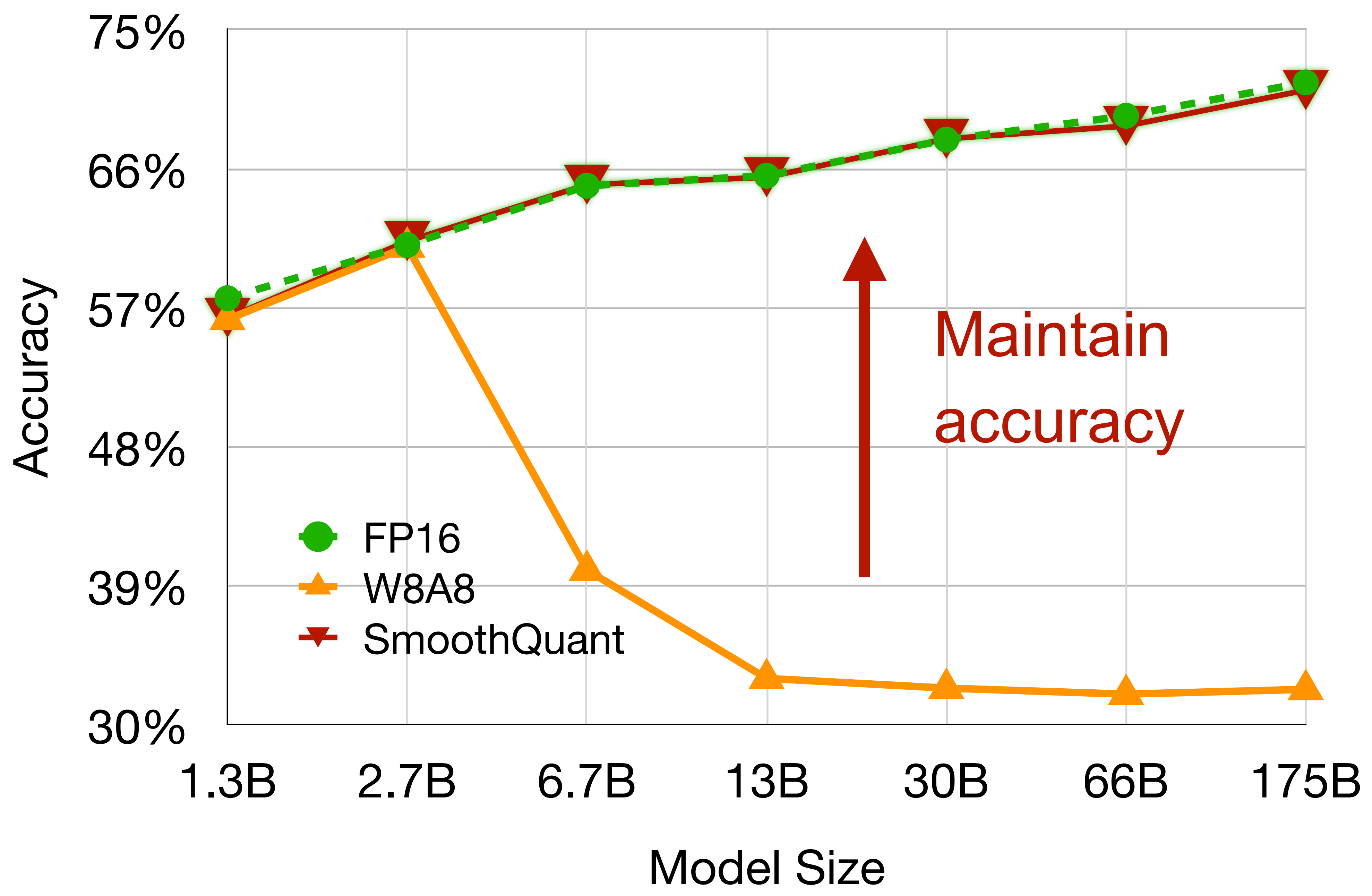
Haotian Tang

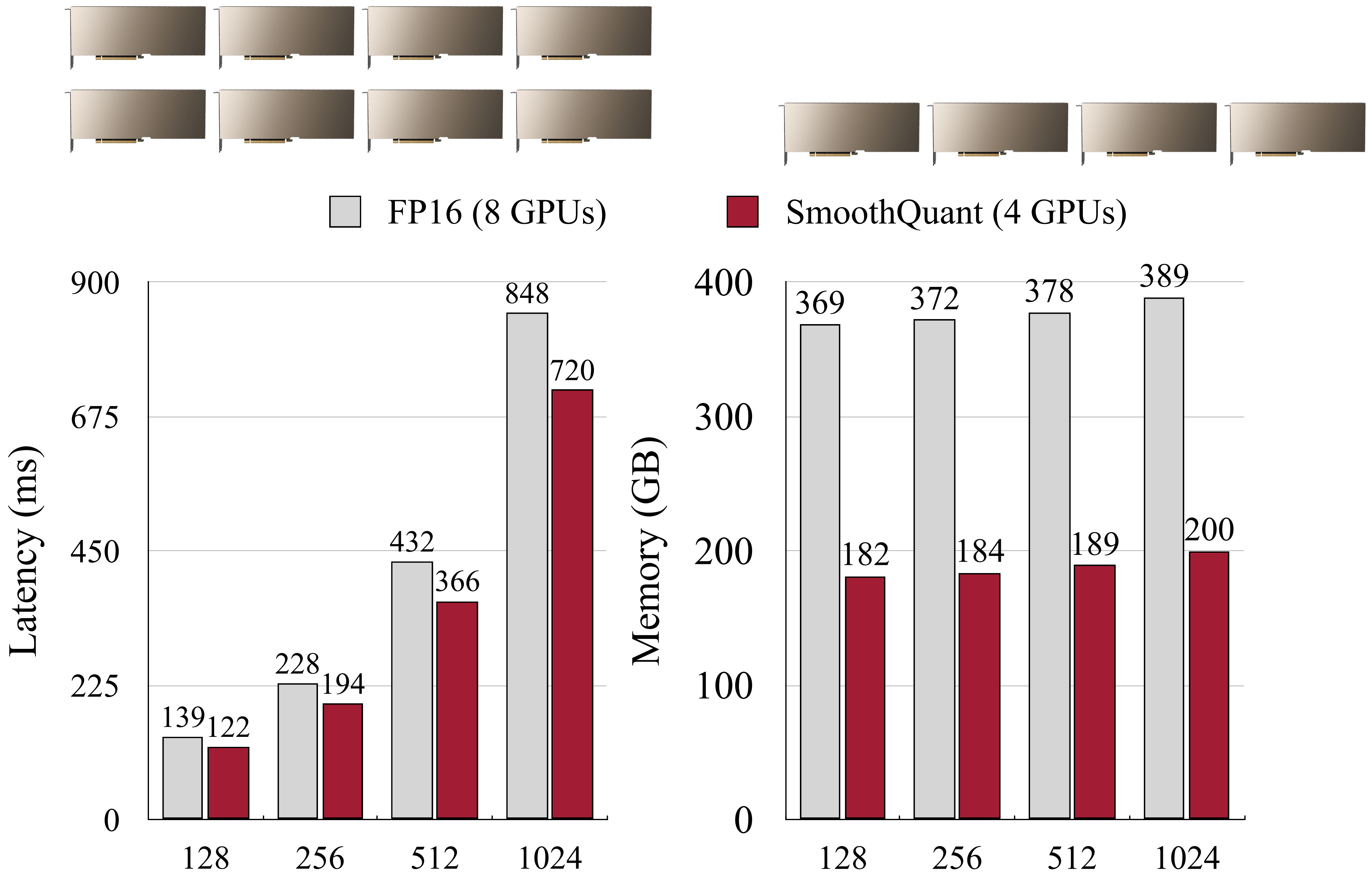
kentang@mit.edu
http://kentang.net

Revisit: SmoothQuant (W8A8)

Accurate and efficient quantization of various LLMs

- SmoothQuant well maintains the accuracy without fine-tuning.
- SmoothQuant can both accelerate inference and halve the memory footprint.





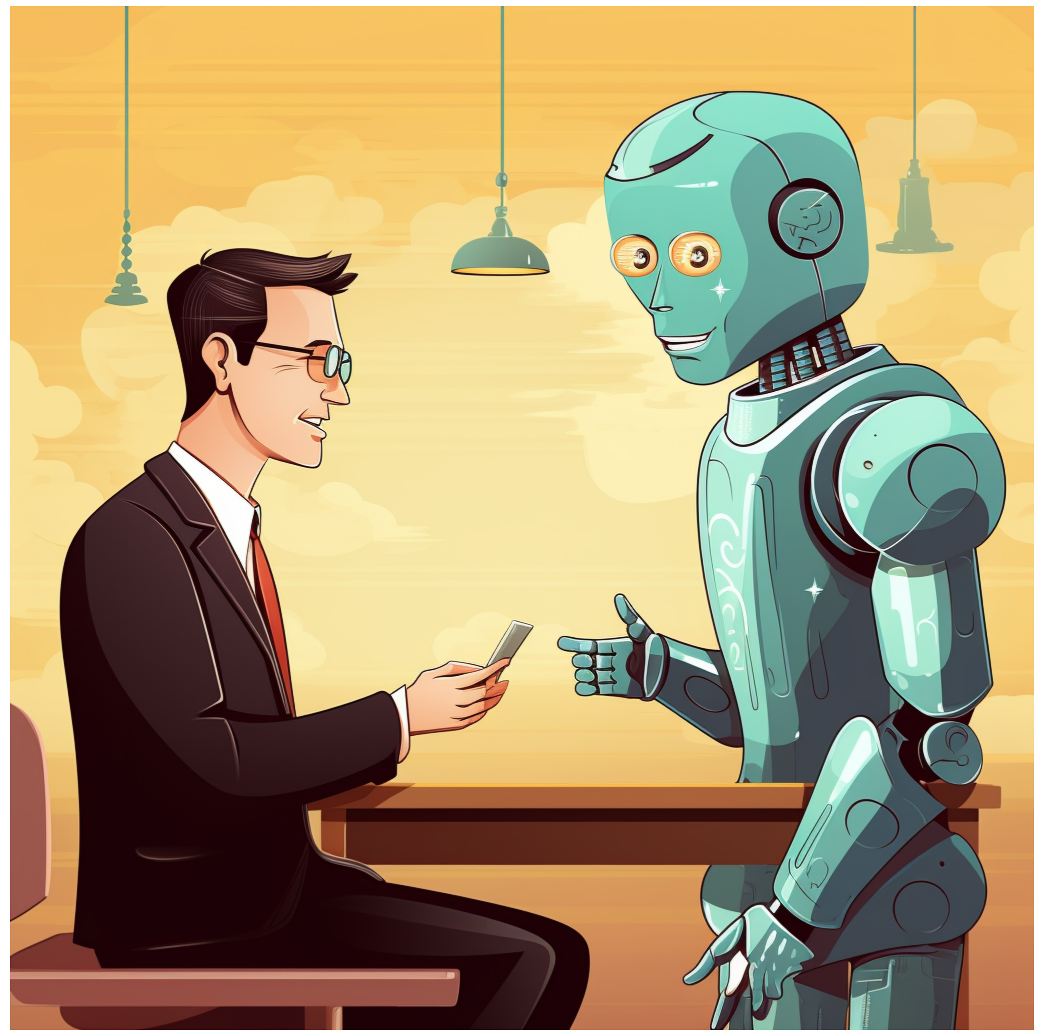
OPT-175B

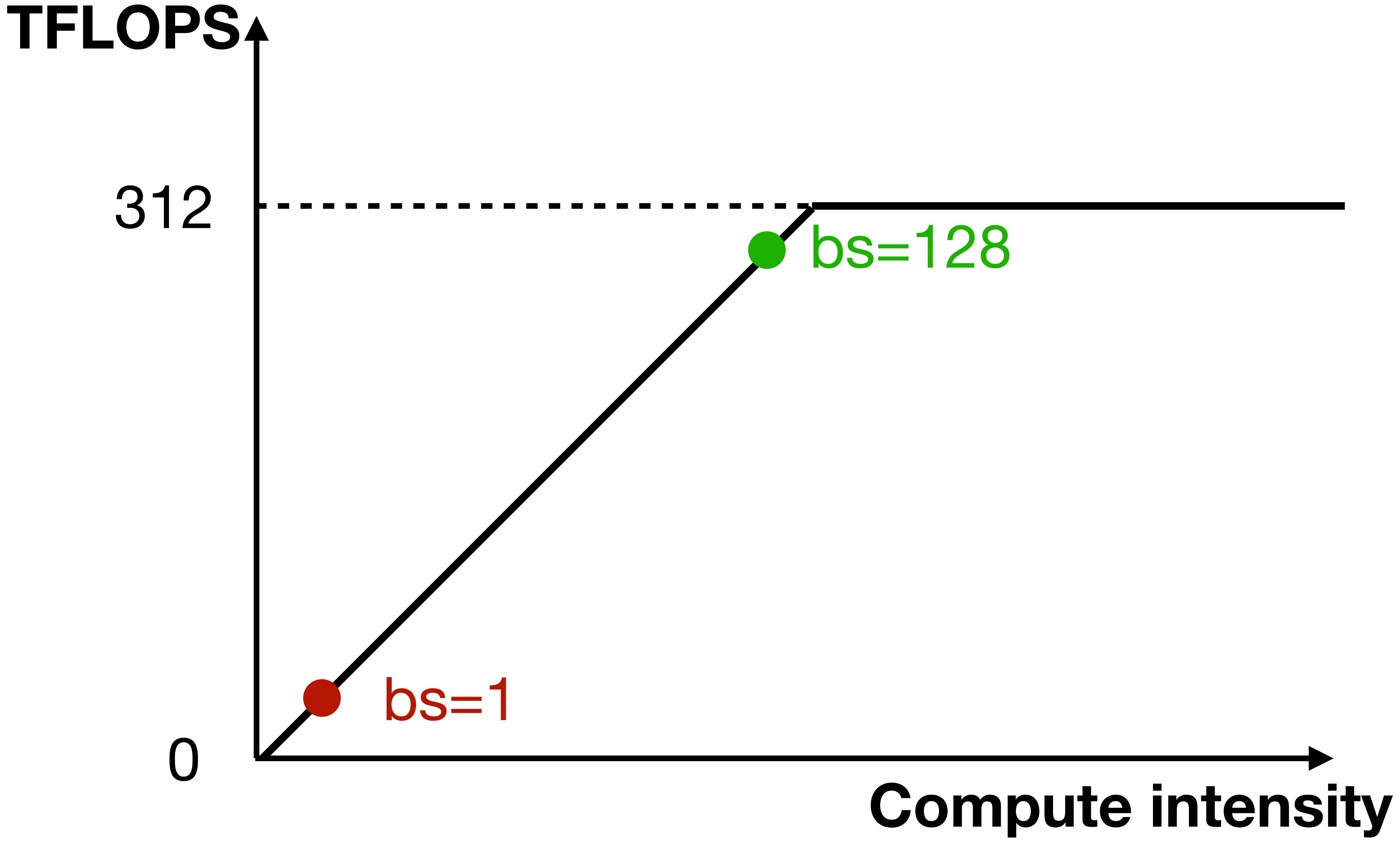
SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models [Xiao *et al.*, 2022]

W4A16 for Single Query Serving

W8A8 cannot address low computational intensity of decoding

- W8A8 quantization is good for batch serving (e.g., batch size 128)
- But single-query LLM inference (e.g., local) is still highly memory-bounded
- We need **low-bit weight-only** quantization (e.g., W4A16) for this setting



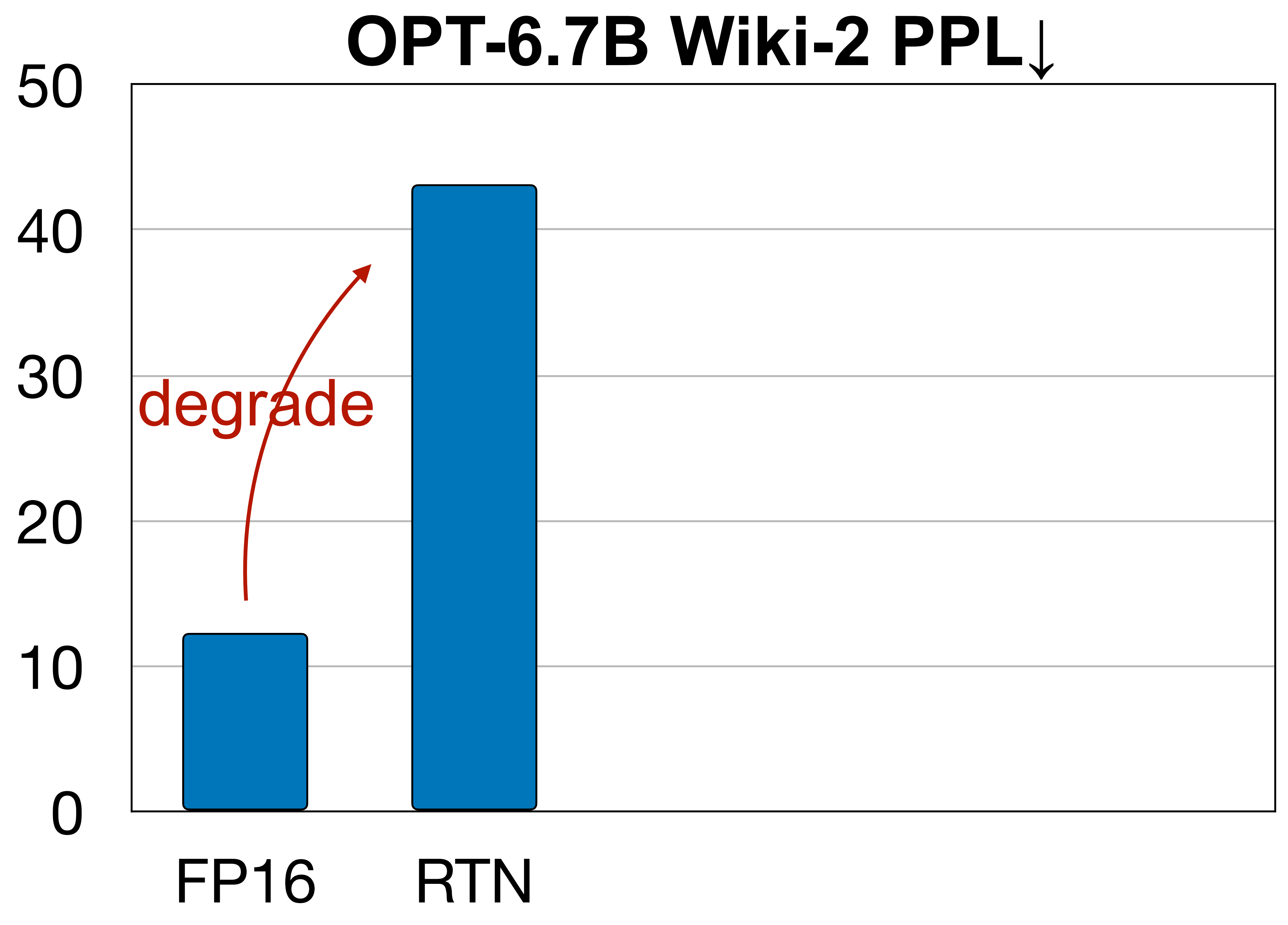


- A100 GPU
- LLaMA-65B decoding

AWQ for Low-bit Weight-only Quantization

Targeting group-wise W3/W4 quantization

$$\mathbf{W}_{\text{FP16}} = \begin{bmatrix} +1.2 & -0.2 & -2.4 & -3.4 \\ -2.5 & -3.5 & +1.9 & +1.4 \\ -0.9 & +1.6 & -2.5 & -1.9 \\ -3.5 & +1.5 & +0.5 & -0.1 \\ +1.8 & -1.6 & -3.2 & -3.4 \\ +2.4 & -3.5 & -2.8 & -3.9 \\ +0.1 & -3.8 & +2.4 & +3.4 \\ +0.9 & +3.3 & -1.9 & -2.3 \end{bmatrix} \xrightarrow{\text{RTN}} Q(\mathbf{W})_{\text{INT3}} = \begin{bmatrix} +1 & +0 & -2 & -3 \\ -3 & -4 & +2 & +1 \\ -1 & +2 & -3 & -2 \\ -4 & +2 & +1 & +0 \\ +2 & -2 & -3 & -3 \\ +2 & -4 & -3 & -4 \\ +0 & -4 & +2 & +3 \\ +1 & +3 & -2 & -2 \end{bmatrix}$$



- Weight-only quantization reduces the memory requirement, and accelerates token generation by alleviating the memory bottleneck.
- Group-wise/block-wise quantization (e.g., 64/128/256) offers a better accuracy-model size trade-off.
- But there is still a performance gap with round-to-nearest (RTN) quantization (INT3-g128)

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration (Lin et al., 2023)

AWQ for Low-bit Weight-only Quantization

Observation: Weights are not equally important; 1% salient weights

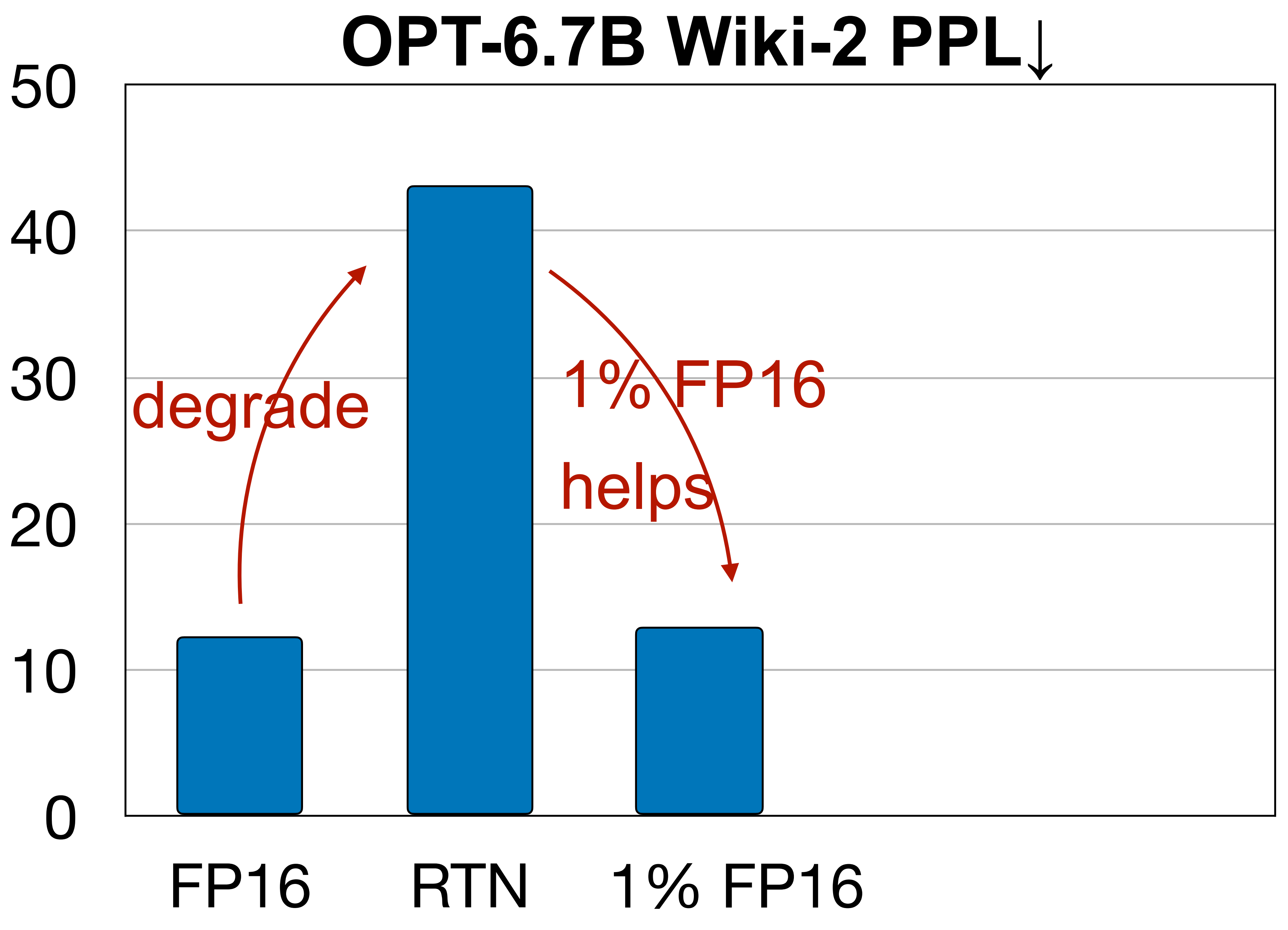
$\mathbf{W}_{\mathrm{FP16}}$

+1.2	−0.2	−2.4	−3.4
−2.5	−3.5	+1.9	+1.4
−0.9	+1.6	−2.5	−1.9
−3.5	+1.5	+0.5	−0.1
+1.8	−1.6	−3.2	−3.4
+2.4	−3.5	−2.8	−3.9
+0.1	−3.8	+2.4	+3.4
+0.9	+3.3	−1.9	−2.3

$\xrightarrow{\text{RTN}}$

$Q(\mathbf{W})_{\mathrm{MixPrec}}$

+1	+0	−2	−3	
−2.5	−3.5	+1.9	+1.4	FP16 channel
−1	+2	−3	−2	
−4	+2	+1	+0	
+2	−2	−3	−3	
+2	−4	−3	−4	
+0	−4	+2	+3	
+1	+3	−2	−2	

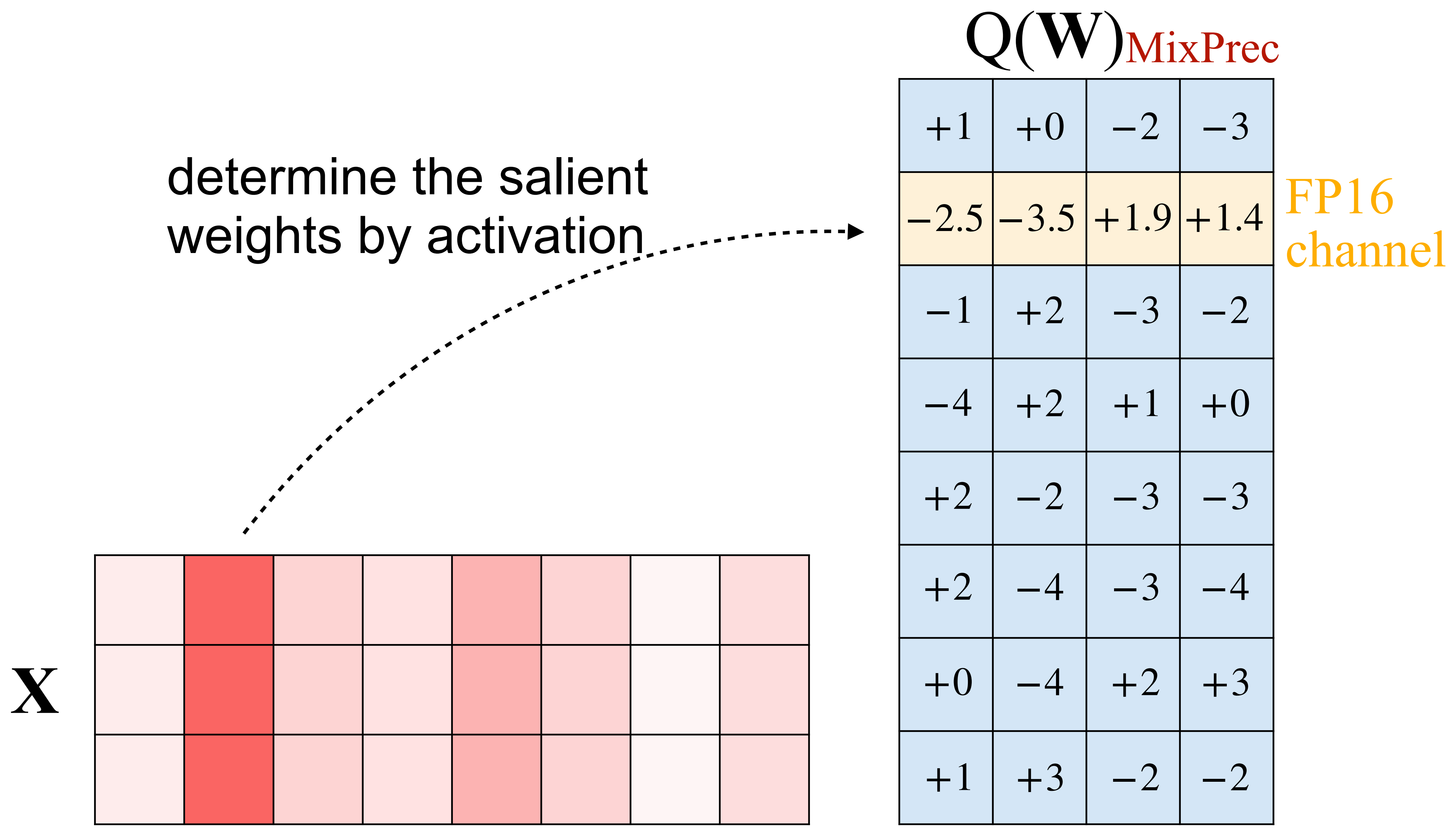


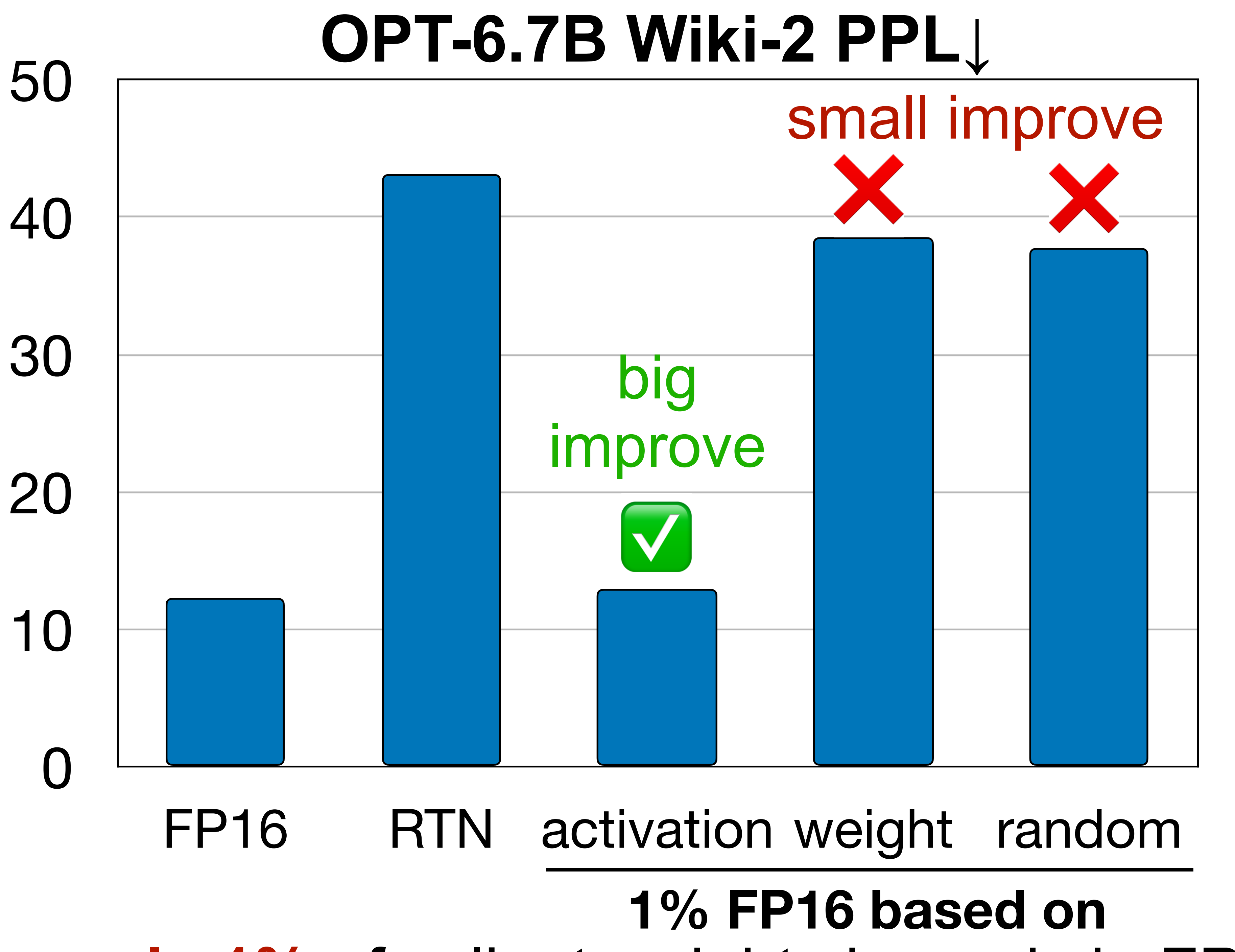
- We find that weights are not equally important, keeping **only 1%** of salient weight channels in FP16 can greatly improve perplexity
- But how do we select salient channels? Should we select based on weight magnitude?

Learning both Weights and Connections for Efficient Neural Networks (Han et al., 2015)

AWQ for Low-bit Weight-only Quantization

Salient weights are determined by activation distribution, not weight



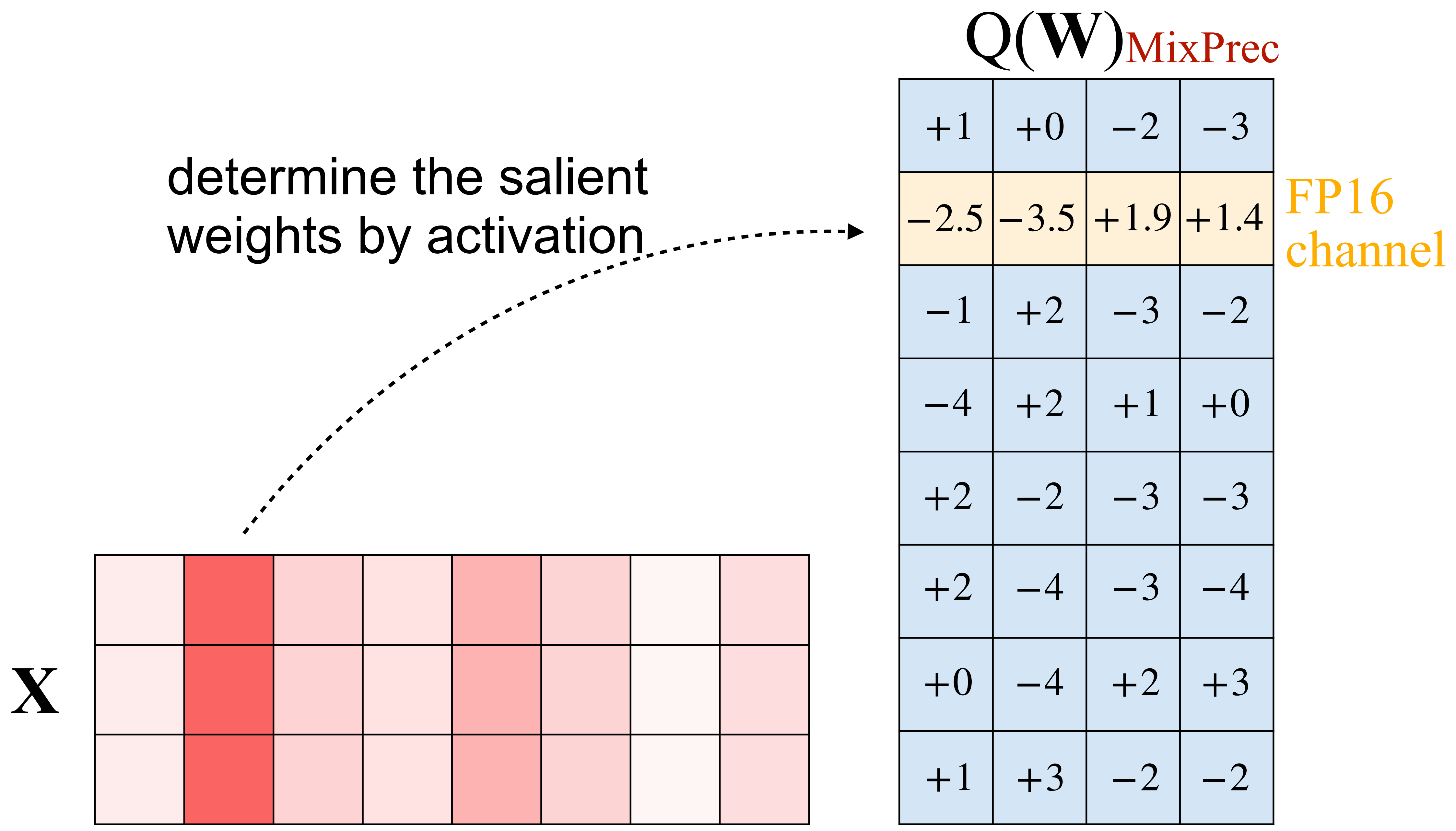


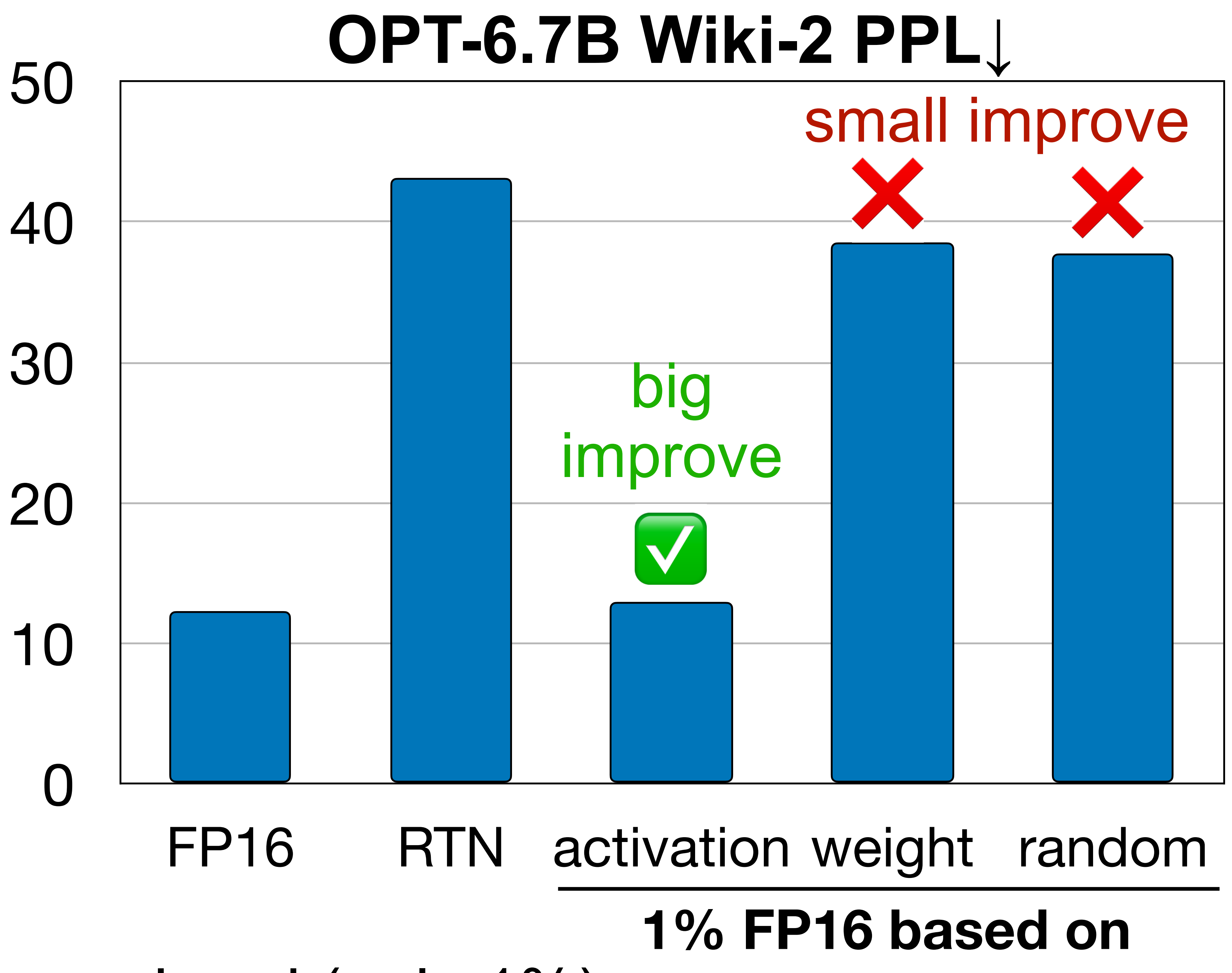
- We find that weights are not equally important, keeping **only 1%** of salient weight channels in FP16 can greatly improve perplexity
- But how do we select salient channels? Should we select based on weight magnitude?
- No! We should look for **activation distribution, but not weight!**

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration (Lin et al., 2023)

AWQ for Low-bit Weight-only Quantization

Salient weights are determined by activation distribution, not weight





- **Pro**: improve quantized performance with a negligible overhead (only 1%)
- **Con**: bad hardware efficiency due to mixed-precision weights
  - A recently work SpQR only manages to get 15% speed-up

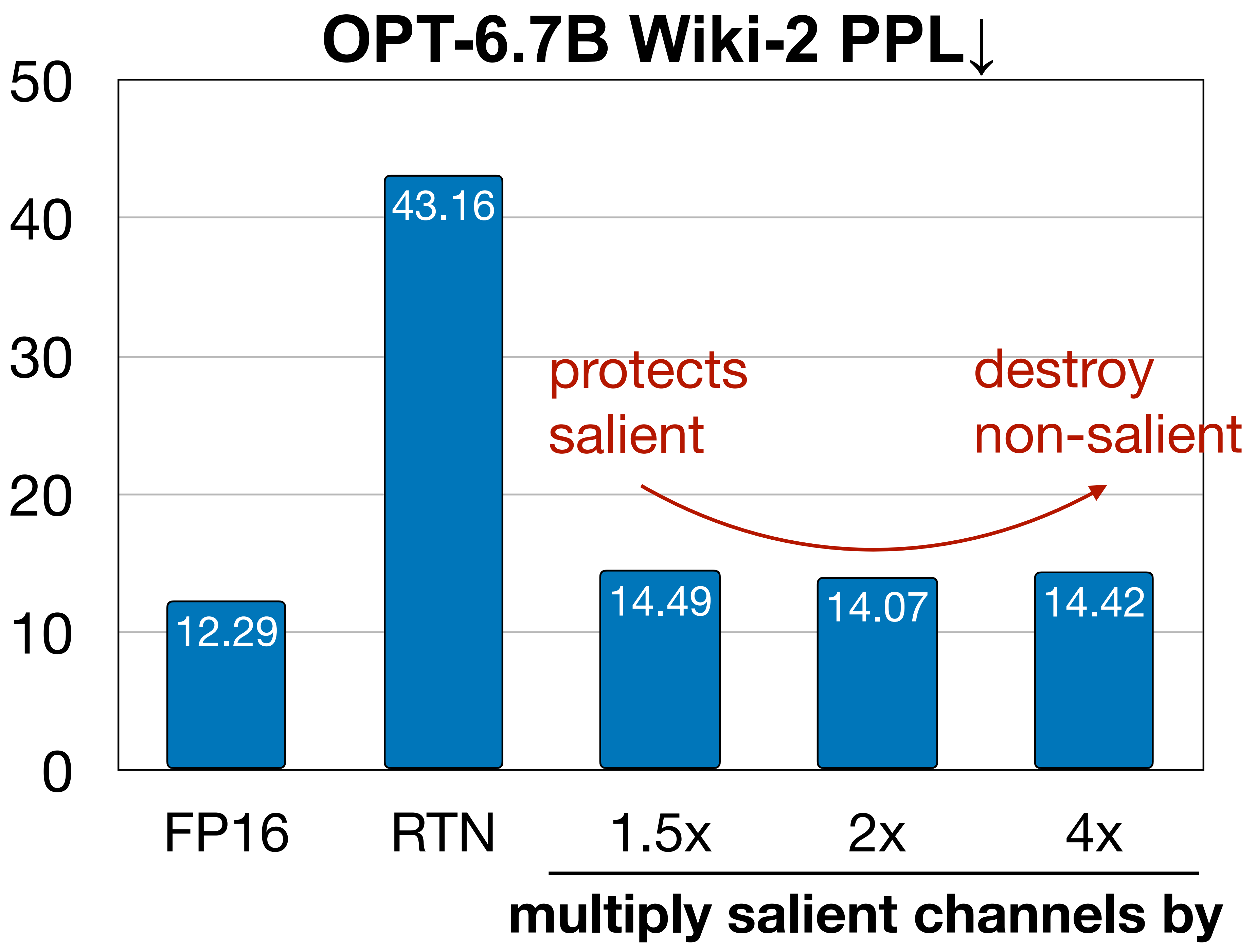
AWQ for Low-bit Weight-only Quantization

Protecting salient weights by scaling (no mixed prec.)

$$
Q\left(
\begin{array}{cccc c}
\multicolumn{4}{c}{\mathbf{W}} & \\
+1.2 & -0.2 & -2.4 & -3.4 & \times 1 \\
-2.5 & -3.5 & +1.9 & +1.4 & \times 2 \\
-0.9 & +1.6 & -2.5 & -1.9 & \times 1 \\
-3.5 & +1.5 & +0.5 & -0.1 & \times 1 \\
+1.8 & -1.6 & -3.2 & -3.4 & \times 1 \\
+2.4 & -3.5 & -2.8 & -3.9 & \times 1 \\
+0.1 & -3.8 & +2.4 & +3.4 & \times 1 \\
+0.9 & +3.3 & -1.9 & -2.3 & \times 1
\end{array}
\right)
$$

fuse to previous op

$$\mathbf{WX} \longrightarrow Q(\mathbf{W} \cdot \mathbf{s})(\mathbf{s}^{-1} \cdot \mathbf{X})$$



- Multiplying the salient channels with $s > 1$ reduces its quantization error
- Why?

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration (Lin et al., 2023)

AWQ for Low-bit Weight-only Quantization

Protecting salient weights by scaling (no mixed precision)

- Consider a linear layer channel $\mathbf{y} = \mathbf{w}x$ (from $\mathbf{Wx}$). We care about the quantization error from $Q(\mathbf{w})x$
- $Q(\mathbf{w}) = \Delta \cdot \text{Round}(\mathbf{w}/\Delta), \quad \Delta = \dfrac{\max(|\mathbf{w}|)}{2^{N-1}}$
- The scaled version is $Q(\mathbf{w} \cdot s)(x/s) = \underline{\Delta} \cdot \underline{\text{Round}(s\mathbf{w}/\Delta)} \cdot x \cdot \underline{\dfrac{1}{s}}$
- We find that the error from Round() is always ~0.25 (average from 0-0.5)
- The maximum value in a group "usually" does not change if we just scale up a channel -> Δ not changed
- With $s > 1$, the error is scaled down.

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration (Lin et al., 2023)

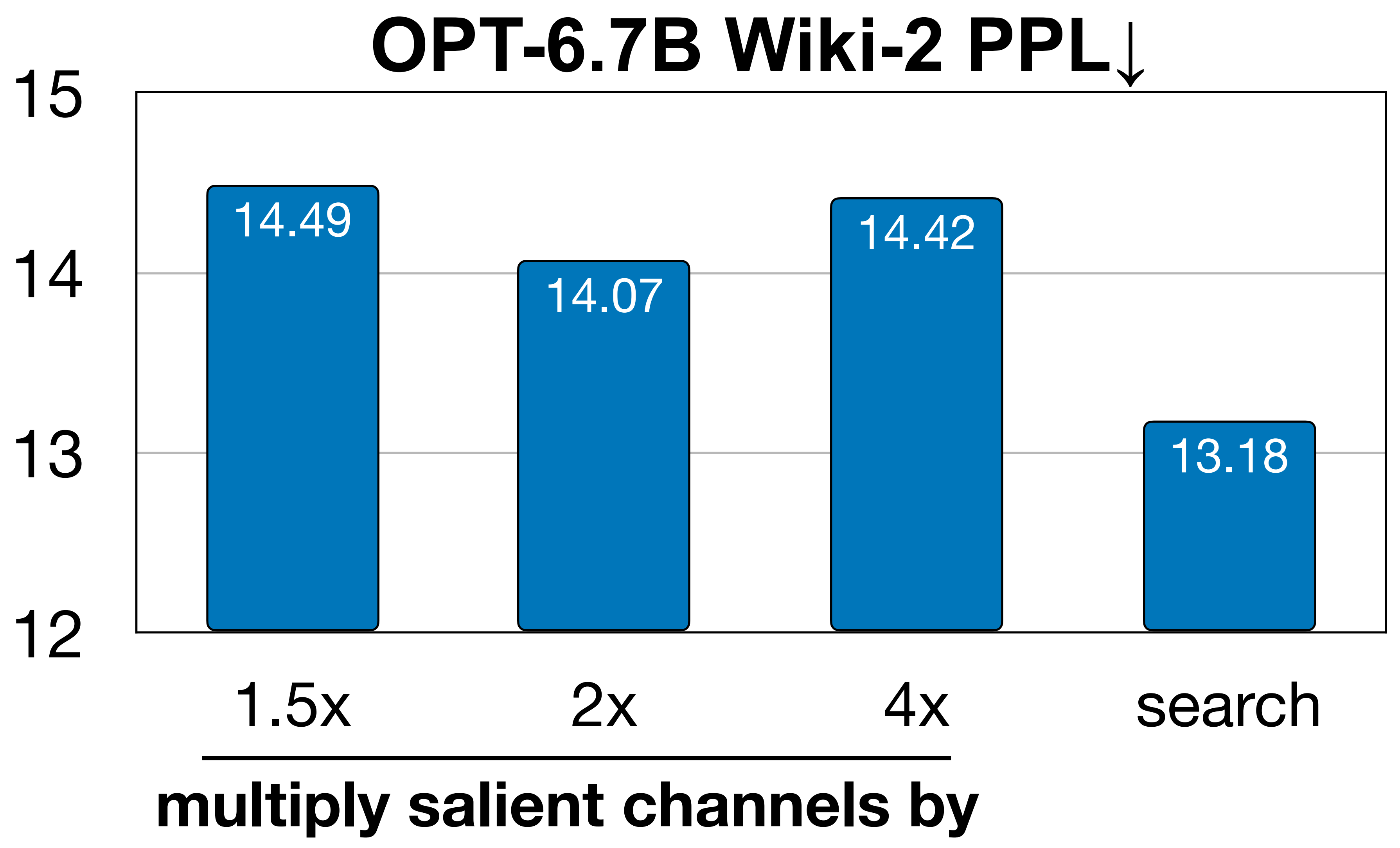
AWQ for Low-bit Weight-only Quantization

Protecting salient weights by scaling (no mixed prec.)

$$
Q\left(
\begin{array}{c}
\mathbf{W} \\
\begin{array}{|c|c|c|c|}
\hline
+1.2 & -0.2 & -2.4 & -3.4 \\ \hline
-2.5 & -3.5 & +1.9 & +1.4 \\ \hline
-0.9 & +1.6 & -2.5 & -1.9 \\ \hline
-3.5 & +1.5 & +0.5 & -0.1 \\ \hline
+1.8 & -1.6 & -3.2 & -3.4 \\ \hline
+2.4 & -3.5 & -2.8 & -3.9 \\ \hline
+0.1 & -3.8 & +2.4 & +3.4 \\ \hline
+0.9 & +3.3 & -1.9 & -2.3 \\ \hline
\end{array}
\end{array}
\begin{array}{c}
\\
\times 1 \\ \times 2 \\ \times 1 \\ \times 1 \\ \times 1 \\ \times 1 \\ \times 1 \\ \times 1
\end{array}
\right)
$$

fuse to previous op

$$\mathbf{WX} \longrightarrow Q(\mathbf{W} \cdot \mathbf{s})(\mathbf{s}^{-1} \cdot \mathbf{X})$$



$$\mathcal{L}(\mathbf{s}) = \|Q(\mathbf{W} \cdot \mathbf{s})(\mathbf{s}^{-1} \cdot \mathbf{X}) - \mathbf{WX}\|$$

$$\mathbf{s} = \mathbf{s_X}^{\alpha}$$

Activation-awareness is important, but not weight-awareness

- Multiplying the salient channels with $s > 1$ reduces its quantization error
- Take a data-driven approach with a fast **grid search**

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration (Lin et al., 2023)

AWQ for Low-bit Weight-only Quantization

Better PPL under low-bit weight-only quantization

PPL↓		Llama-2			LLaMA			
		7B	13B	70B	7B	13B	30B	65B
FP16	-	5.47	4.88	3.32	5.68	5.09	4.10	3.53
INT3 g128	RTN	6.66	5.52	3.98	7.01	5.88	4.88	4.24
	GPTQ	6.43	5.48	3.88	8.81	5.66	4.88	4.17
	GPTQ-R	6.42	5.41	3.86	6.53	5.64	4.74	4.21
	AWQ	**6.24**	**5.32**	**3.74**	**6.35**	**5.52**	**4.61**	**3.95**
INT4 g128	RTN	5.73	4.98	3.46	5.96	5.25	4.23	3.67
	GPTQ	5.69	4.98	3.42	6.22	5.23	4.24	3.66
	GPTQ-R	5.63	4.99	3.43	5.83	5.20	4.22	3.66
	AWQ	**5.60**	**4.97**	**3.41**	**5.78**	**5.19**	**4.21**	**3.62**

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration (Lin et al., 2023)

AWQ for Low-bit Weight-only Quantization

Also works for multi-modal LLMs (OpenFlamingo-9B, captioning)

COCO (CIDEr ↑)		0-shot	4-shot	8-shot	16-shot	32-shot	Δ(32-shot)
FP16	-	63.73	72.18	76.95	79.74	81.70	-
INT4 g128	RTN	60.24	68.07	72.46	74.09	77.13	-4.57
	GPTQ	59.72	67.68	72.53	74.98	74.98	-6.72
	AWQ	**62.57**	**71.02**	**74.75**	**78.23**	**80.53**	**-1.17**
INT3 g128	RTN	46.07	55.13	60.46	63.21	64.79	-16.91
	GPTQ	29.84	50.77	56.55	60.54	64.77	-16.93
	AWQ	**56.33**	**64.73**	**68.79**	**72.86**	**74.47**	**-7.23**

W4-RTN: A model airplane flying in the sky.

W4-AWQ: Two toy airplanes sit on a grass field.

W4-RTN: A man is holding a baby elephant in his arms.

W4-AWQ: A man and his daughter pose with an elephant.

W4-RTN: A man and a dog walking past some bushes.

W4-AWQ: Two dogs are walking on the street.

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration (Lin et al., 2023)

AWQ for Low-bit Weight-only Quantization

Also works for multi-modal LLMs (LLaVA, visual reasoning)

Question: Can you explain this meme in detail?

INT4-g128 RTN: The meme in the image features a plate of food, likely fried chicken, with a unique twist. There are small pictures of the Earth and other planets placed on top of the food, creating an interesting and amusing visual. Alongside the plate, a caption reads, "Sometimes I look at pictures of the Earth from space and I marvel at how beautiful it all is." This combination of food and planet images adds a touch of humor and curiosity, making the viewer wonder about the context and intent behind the meme.

INT4-g128 AWQ: The meme in the image is a lighthearted and humorous take on the concept of looking at pictures of the Earth from space. A plate of fried food, specifically chicken nuggets, is presented with the caption, "Sometimes I look at pictures of the Earth from space and I marvel at how beautiful it all is." The playful combination of a food item and an Earth-related context creates a fun and entertaining visual.

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration (Lin et al., 2023)

AWQ for Low-bit Weight-only Quantization

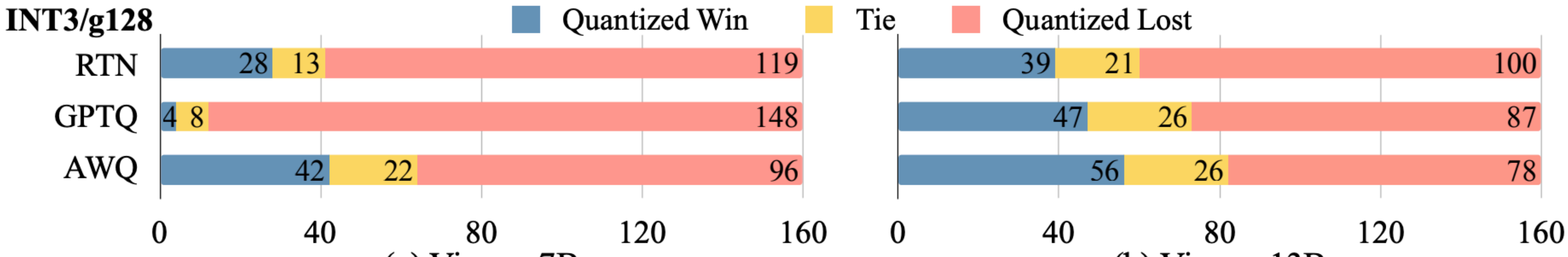
Also works for multi-modal LLMs (LLaVA, visual reasoning)

Question: Do you know who drew this painting?
INT4-g128 RTN: The painting depicts a woman, who some believe to be a representation of the famous artist, Mona Lisa. However, it is not certain that the woman in the painting is indeed Mona Lisa. The artwork is a popular subject of artistic analysis and admiration, often considered a symbol of feminine beauty and enigma. The portrait is characterized by its intricate details, captivating the viewer with the woman's expressive gaze and the subtle, enigmatic smile that has become synonymous with the Mona Lisa.
INT4-g128 AWQ: The painting depicts a woman, who is believed to be a portrait of Mona Lisa, the famous artwork by the artist Leonardo da Vinci. Although the painting is widely recognized as a representation of Mona Lisa, it is not possible to confirm the artist who drew this particular painting without additional information. Leonardo da Vinci was a renowned artist known for his exceptional skill in painting detailed, realistic portraits, and the Mona Lisa is one of his most famous works.

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration (Lin et al., 2023)

Results

Quantization of instruction-tuned models



(a) Vicuna-7B (b) Vicuna-13B

- Comparing quantized Vicuna with FP16 counterparts
- *W4 almost preserves performance*

Results

Quantization for complex generations (code & math)

MBPP (7B)	pass@1	pass@10
FP16	38.53	49.77
RTN	37.51	48.49
GPTQ	31.97	44.75
AWQ	**40.64**	**49.25**

GSM-8K	7B	13B	70B
FP16	13.87	26.16	56.41
RTN	11.07	21.23	53.98
GPTQ	12.13	24.26	56.03
AWQ	**13.57**	**25.25**	**56.40**

- **MBPP**: a Python coding dataset
- **GSM-8K**: a math reasoning dataset (requires multi-step reasoning)
- AWQ preserves the accuracy under **W4-g128** quantization

Industry & Community Impact

SmoothQuant and AWQ are widely used:

FasterTransformer
TRT-LLM

https://github.com/NVIDIA/FasterTransformer/blob/main/docs/gpt_guide.md
https://github.com/NVIDIA/TensorRT-LLM#key-features

text-generation-inference

https://github.com/huggingface/text-generation-inference/tree/main/server/text_generation_server/utils/awq/quantize

Neural Compressor
Q8-Chat

https://github.com/intel/neural-compressor/blob/master/docs/source/smooth_quant.md

lmdeploy

https://github.com/InternLM/lmdeploy/blob/main/lmdeploy/lite/quantization/awq.py

vLLM

https://github.com/vllm-project/vllm/blob/main/vllm/model_executor/quantization_utils/awq.py

oobabooga/text-generation-webui

https://github.com/oobabooga/text-generation-webui/blob/main/modules/models.py

lm-sys/FastChat

https://github.com/lm-sys/FastChat/blob/main/docs/awq.md

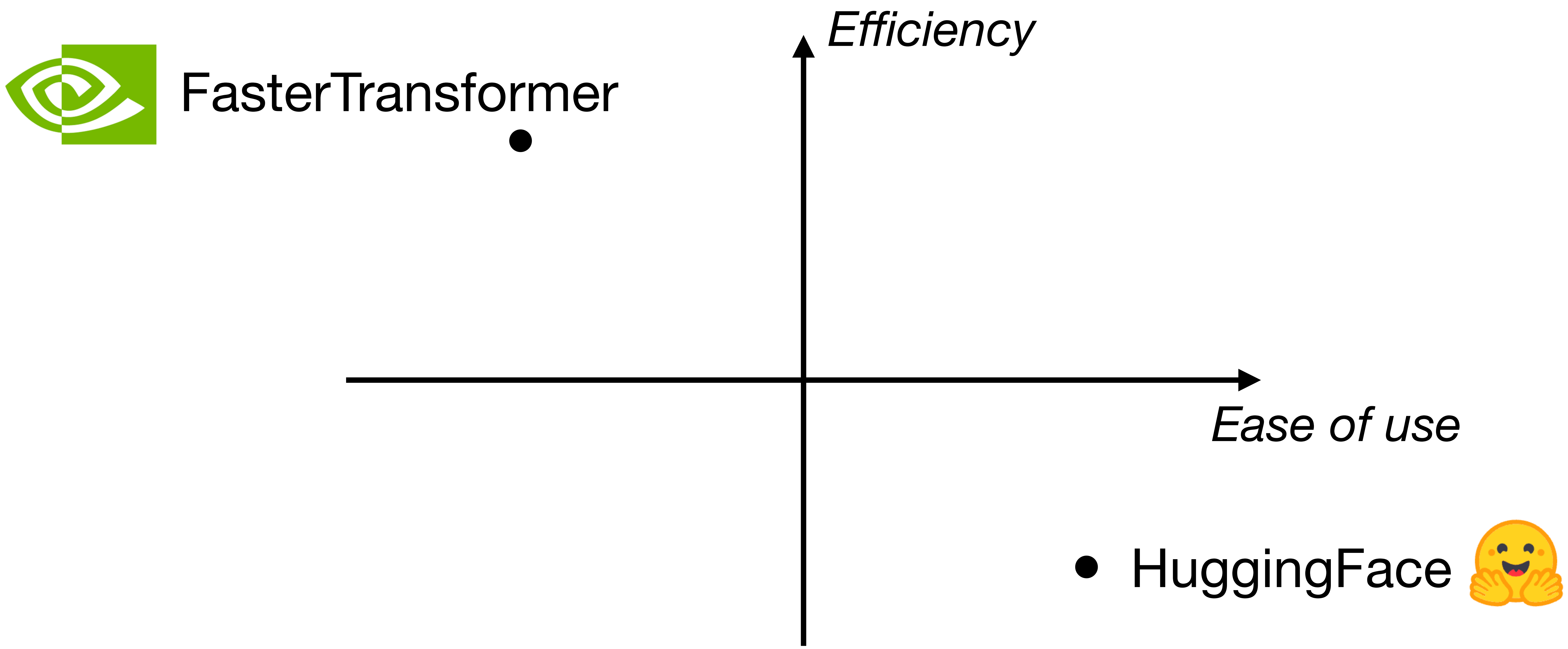
replicate

https://github.com/replicate/vllm-with-loras/blob/main/vllm/model_executor/quantization_utils/awq.py

TinyChat: A Lightweight Serving Infra

Pythonic, lightweight, efficient

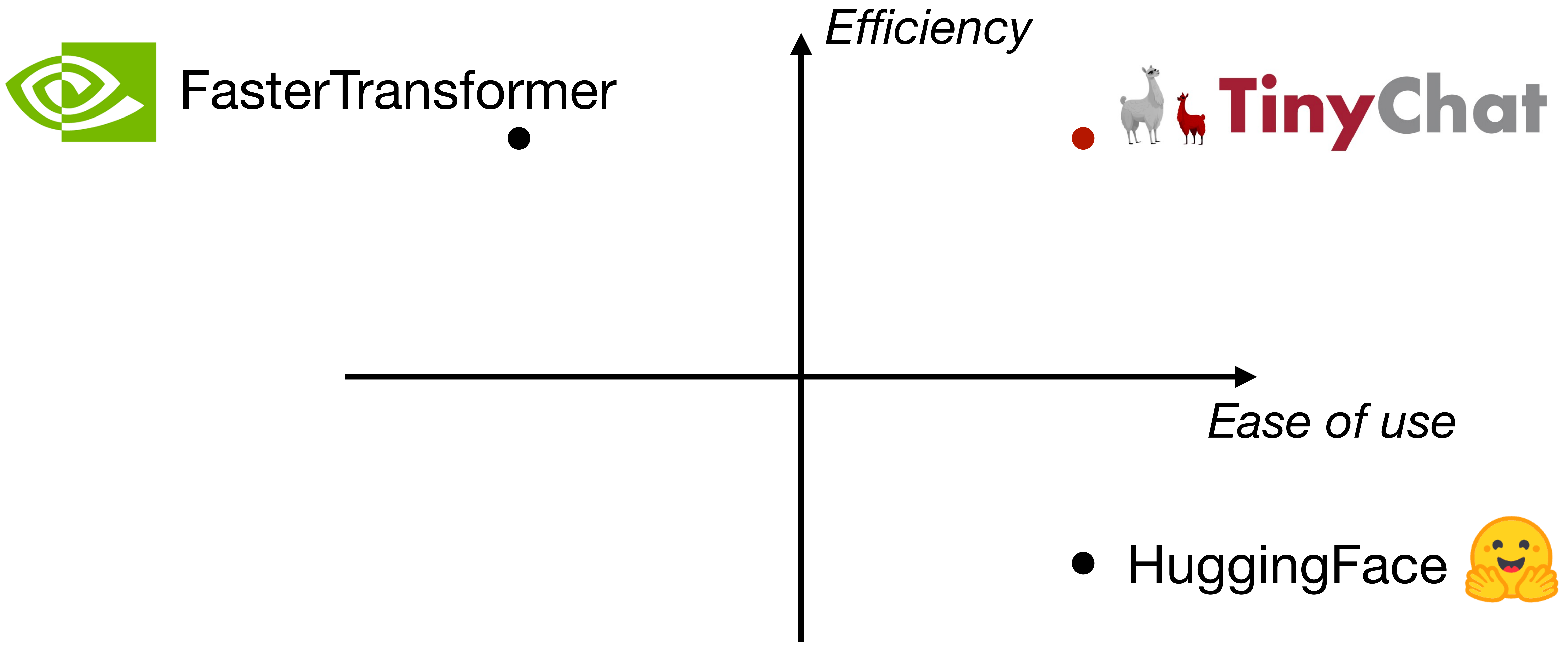
- We need a framework to serve the quantized model to achieve low latency (AWQ only for Linears)
  - HuggingFace: easy to use, but slow
  - FasterTransformer: high efficiency, but harder to use



TinyChat: A Lightweight Serving Infra

Pythonic, lightweight, efficient

- We need a framework to serve the quantized model to achieve low latency
  - HuggingFace: easy to use, but slow
  - FasterTransformer: high efficiency, but harder to use
- **TinyChat** goals: efficient, lightweight, Python-native (composable with other stacks like vLLM)



TinyChat: A Lightweight Serving Infra

Analyze the latency overhead under FP16

- Measurement based on LLaMA-7B on RTX4090

Technique	Tok/sec	Ratio
Huggingface impl.	49.0	76%
FasterTransformer impl.	64.1	100%

TinyChat: A Lightweight Serving Infra

Analyze the latency overhead under FP16

- Measurement based on LLaMA-7B on RTX4090
- Some overheads can be easily removed! 95% of FT performance in Python

Technique	Tok/sec	Ratio
Huggingface impl.	49.0	76%
Preallocate KV cache	54.1	84%
FT LayerNorm kernel	57.5	90%
FlashAttention	57.5	90%
Merge QKV projections	59.2	92%
Fuse rotary embedding	61.0	95%
FasterTransformer impl.	64.1	100%

TinyChat: A Lightweight Serving Infra

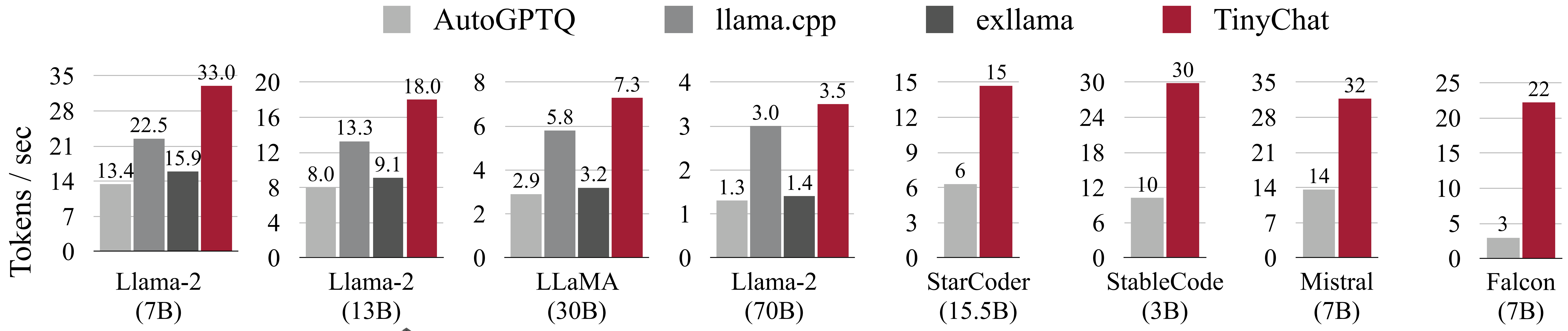
State-of-the-art W4 inference speed

- Now we plugin the AWQ to quantize the weights into 4-bit (further 3x improvement)
- We can outperform the state-of-the-art MLC-LLM (TVM compilation-based) with our **Pythonic** solution
- 50% faster on AGX Orin

LLaMA-7B	RTX 4090 Tok/sec	AGX Orin Tok/sec
llama.cpp	141	22.5
Exllama	153	15.9
MLC-LLM	191	-
TinyChat	**195**	30.2

TinyChat: A Lightweight Serving Infra

Supporting a wide range of models on NVIDIA Jetson Orin



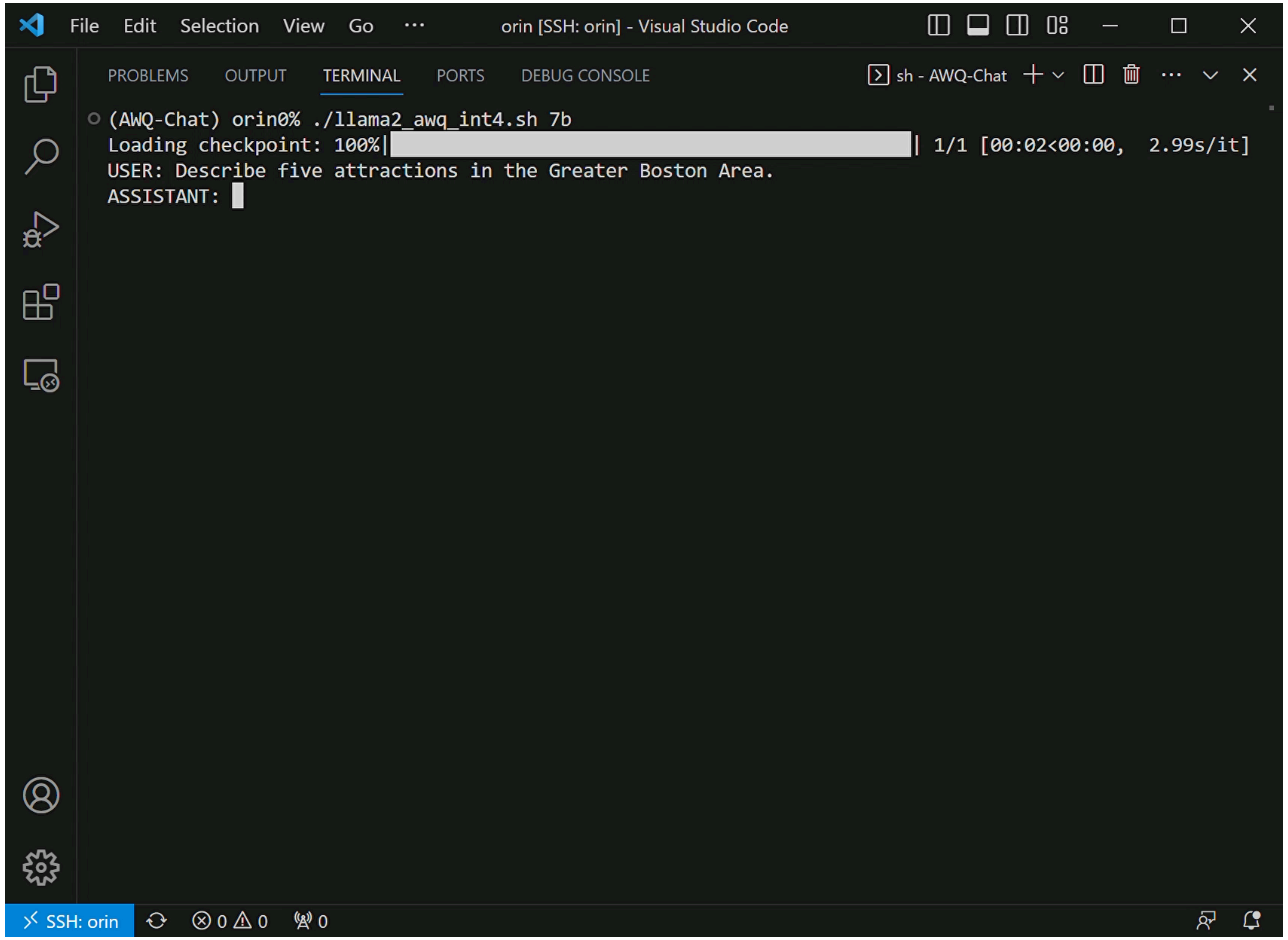
Latency comparison on Jetson Orin (64G) mobile GPU

- TinyChat achieves up to **1.5x** faster runtime for Meta's Llama models compared with systems specialized for this model.
- Compared with the only competitor that can support a diverse range of models, TinyChat is up to **7x** faster.
- Remarkably, TinyChat's front end is **fully PyTorch-based**.

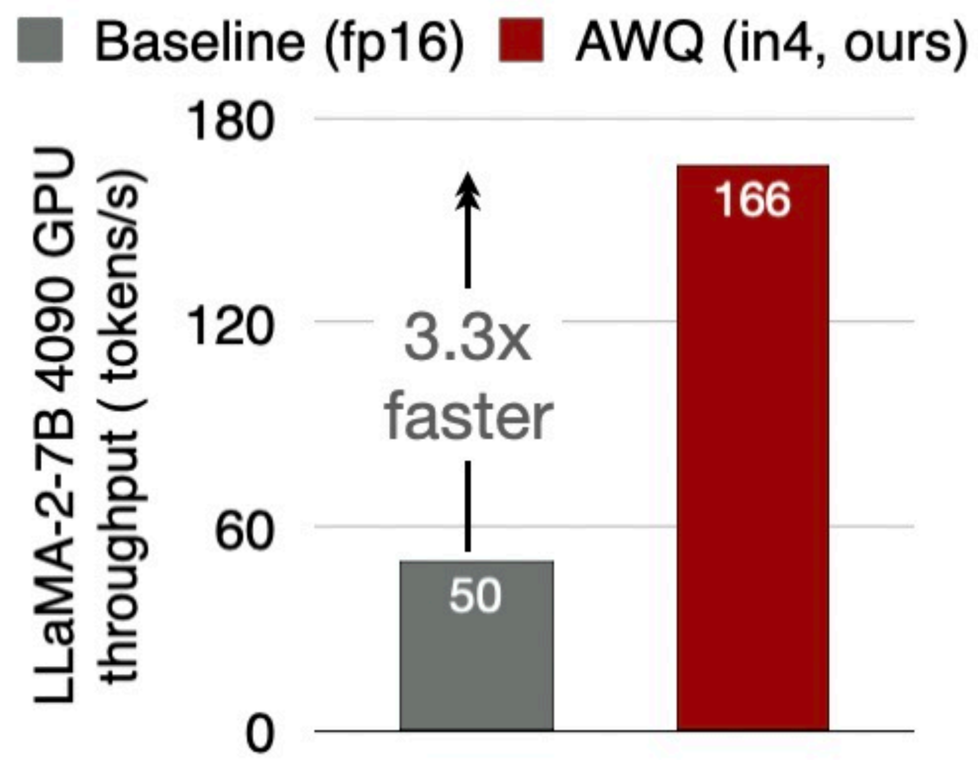
TinyChat: A Lightweight Serving Infra

Demo on AGX Orin (edge LLM inference)

- Orin Nano has 200GB/s memory bandwidth; even more memory-bounded
- Model size: 7B. ~**30 token/s** generation



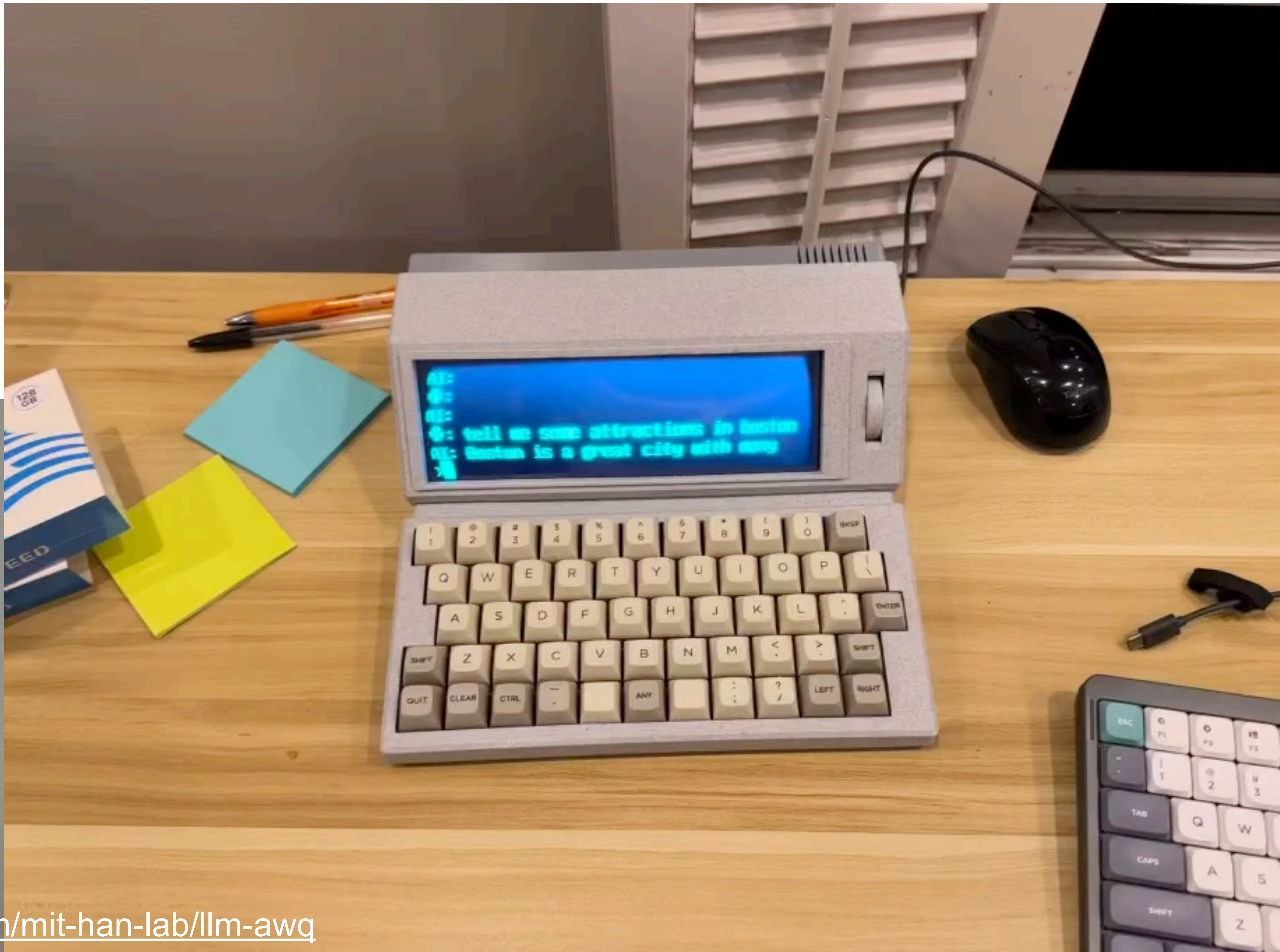
TinyChat



Speedup on 4090 GPU

- A LightWeight Chatbot for LLMs on the edge

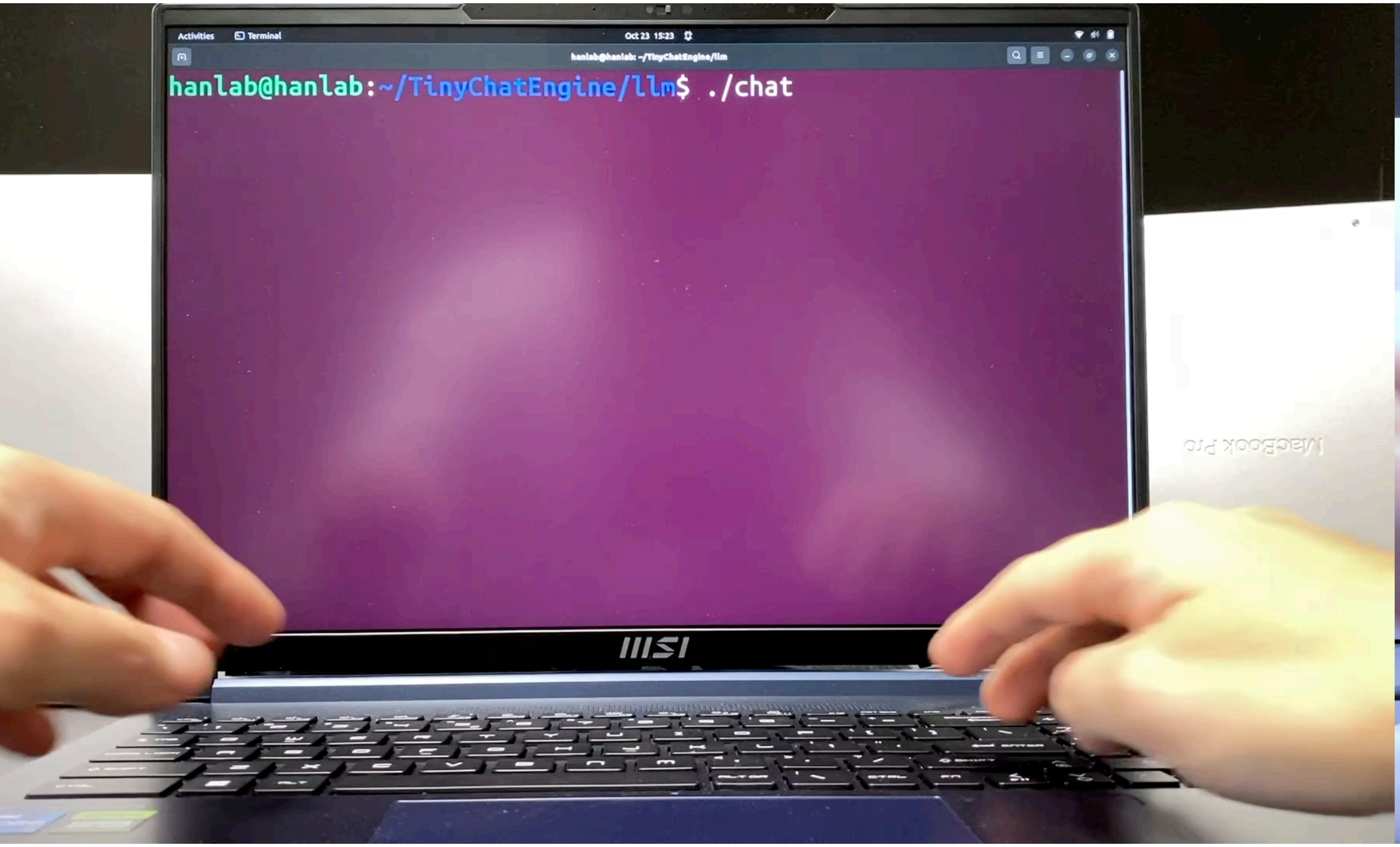
- Deploying LLM on the edge is useful: running copilot services (code completion, office, game chat) locally on laptops, cars, robots, and more. These devices are **resource-constrained**, **low-power** and sometimes **do not have access to the Internet**.
- **Data privacy** is important. Users do not want to share personal data with large companies.

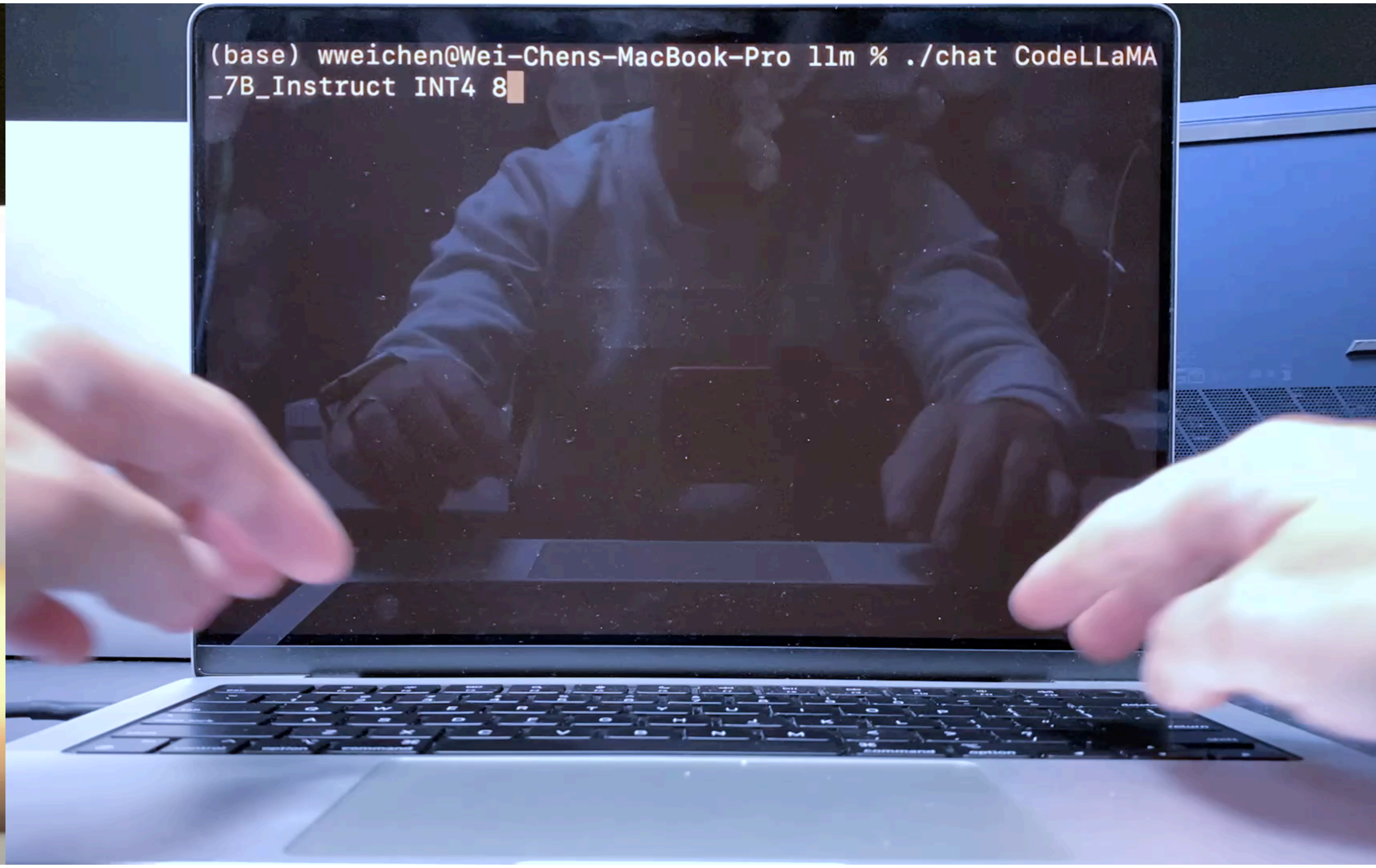


https://github.com/mit-han-lab/llm-awq

TinyChat

- TinyChatEngine implements the compressed AWQ 4bit model, built from C/C++ from scratch, easy to install and migrate to edge platforms
- Enables on-device LLM

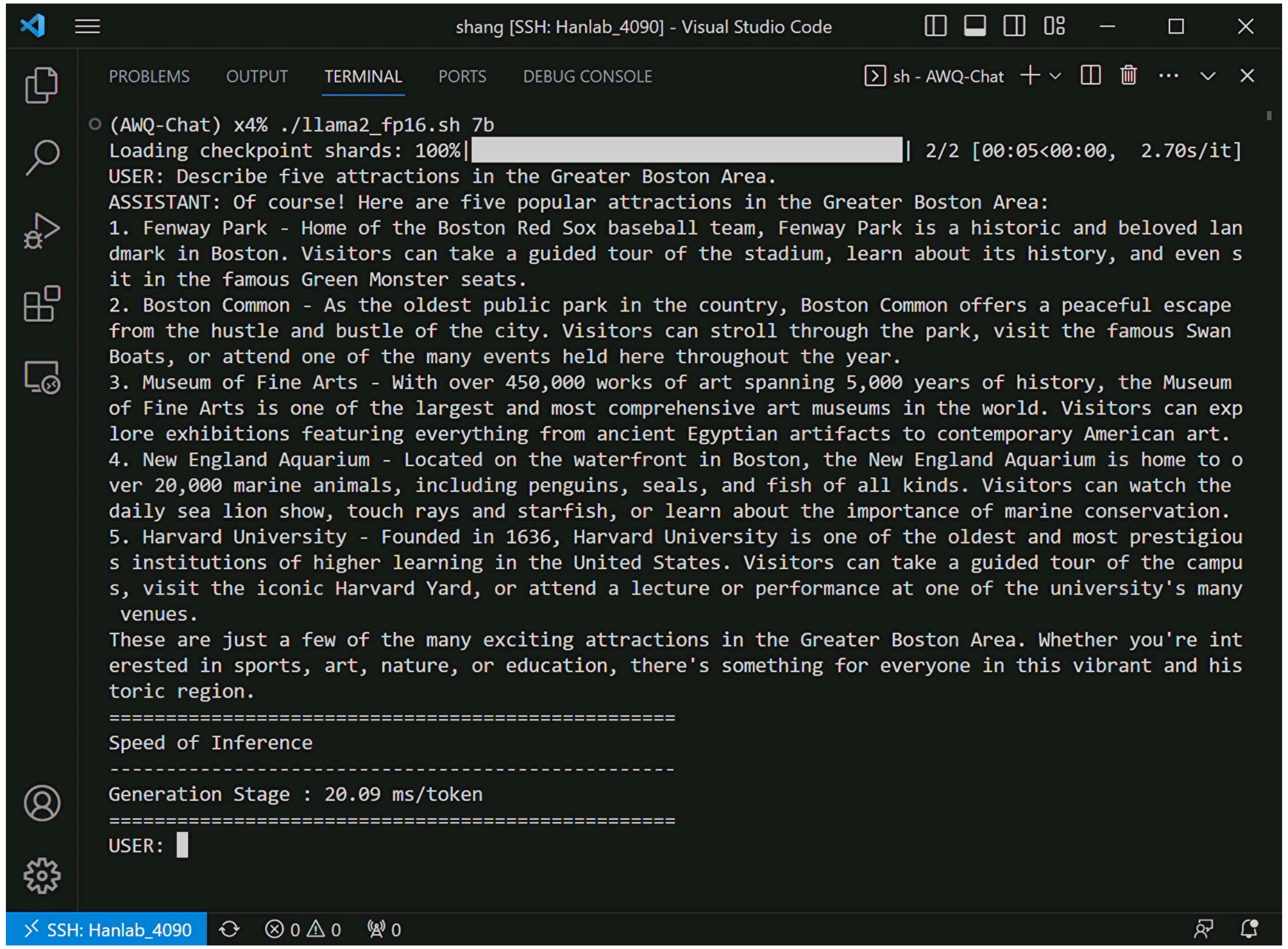




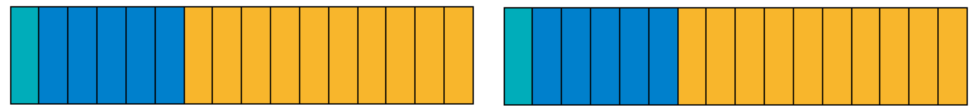
https://github.com/mit-han-lab/llm-awq

TinyChat

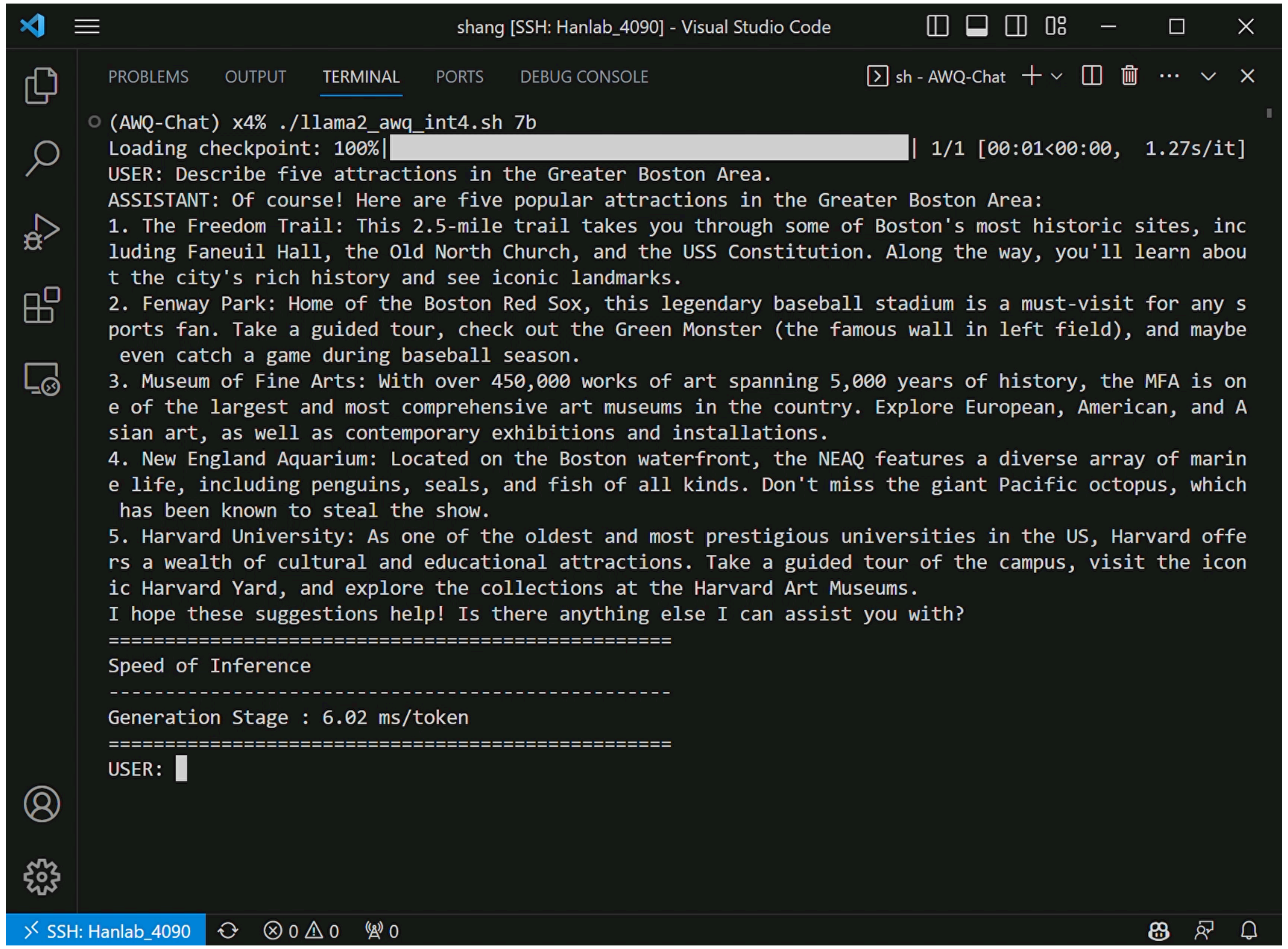
TinyChat brings about 3.3x speedup to LLaMA-2 on 4090



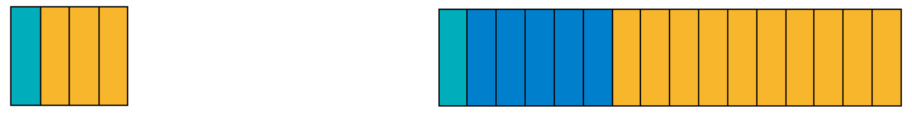
LLaMA-2-7B (FP16): 50 tokens / s



Baseline: fp16 weight, fp16 activation



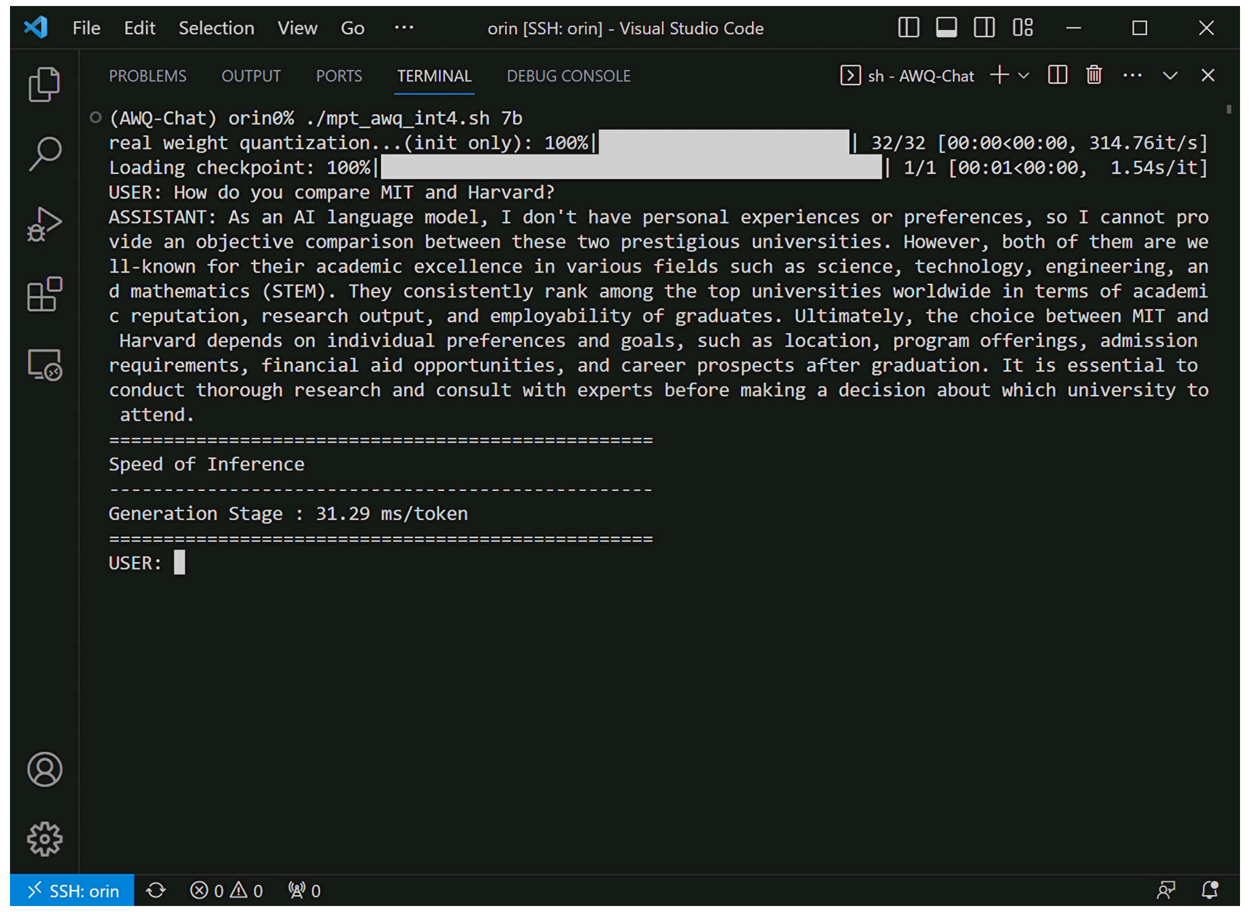
LLaMA-2-7B (W4A16, AWQ): 166 tokens / s

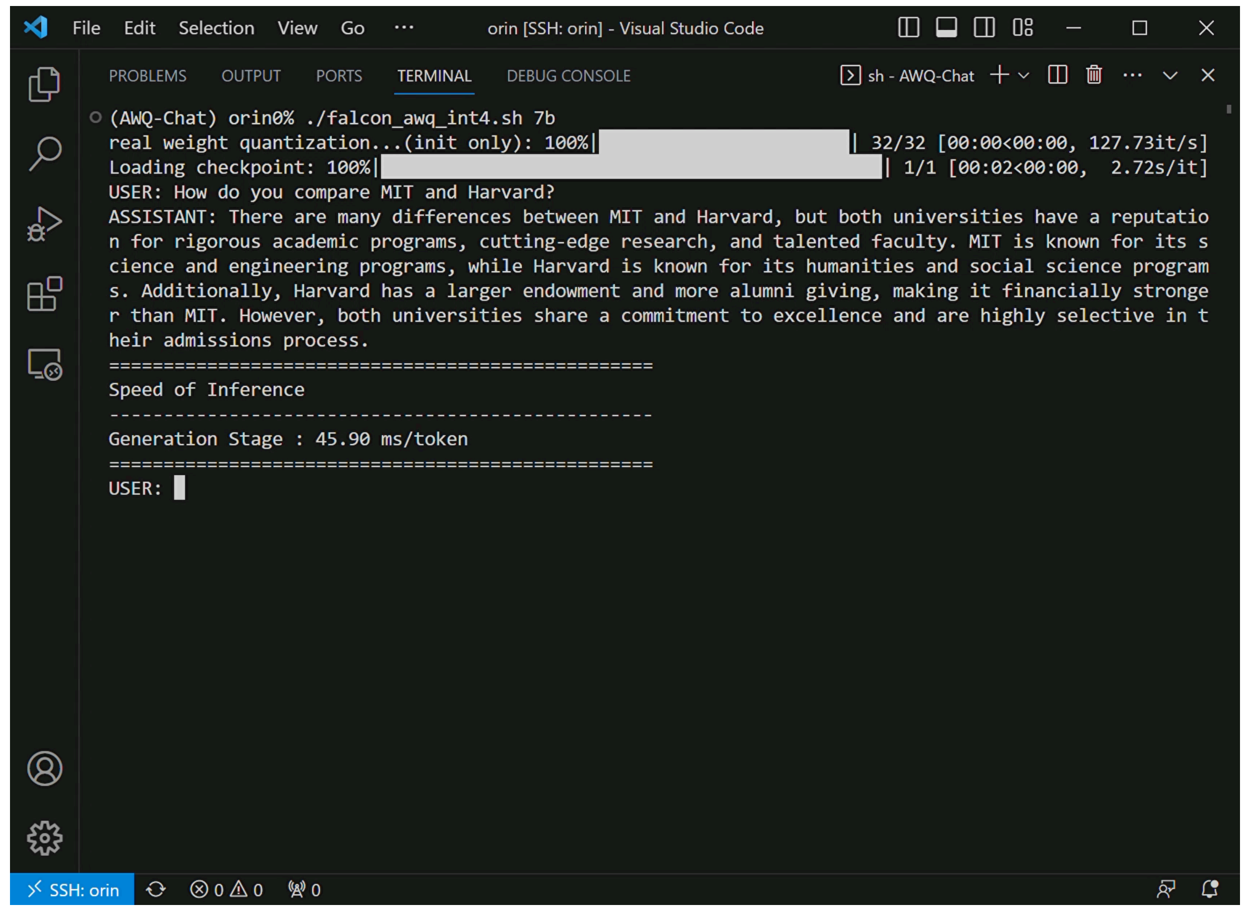


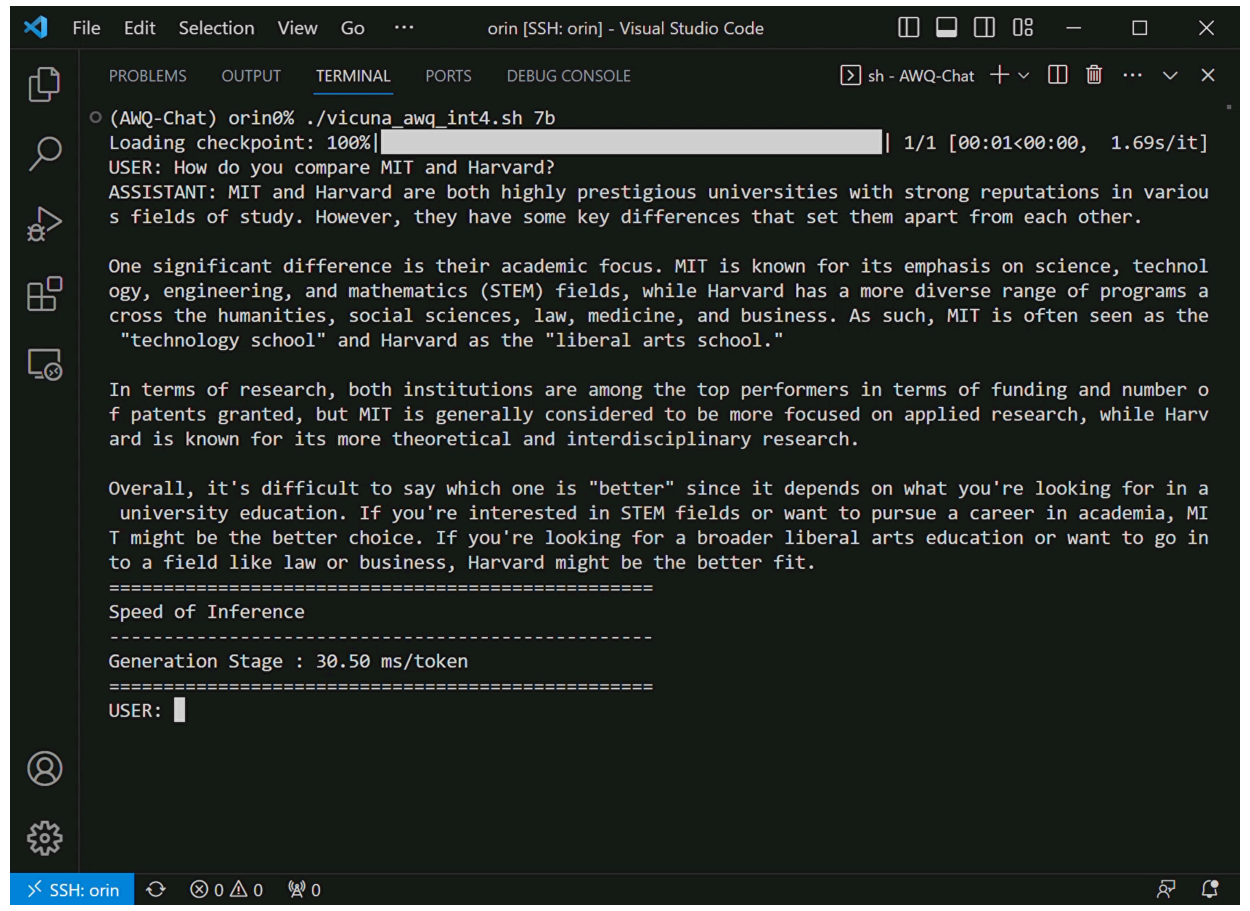
AWQ: int4 weight, fp16 activation

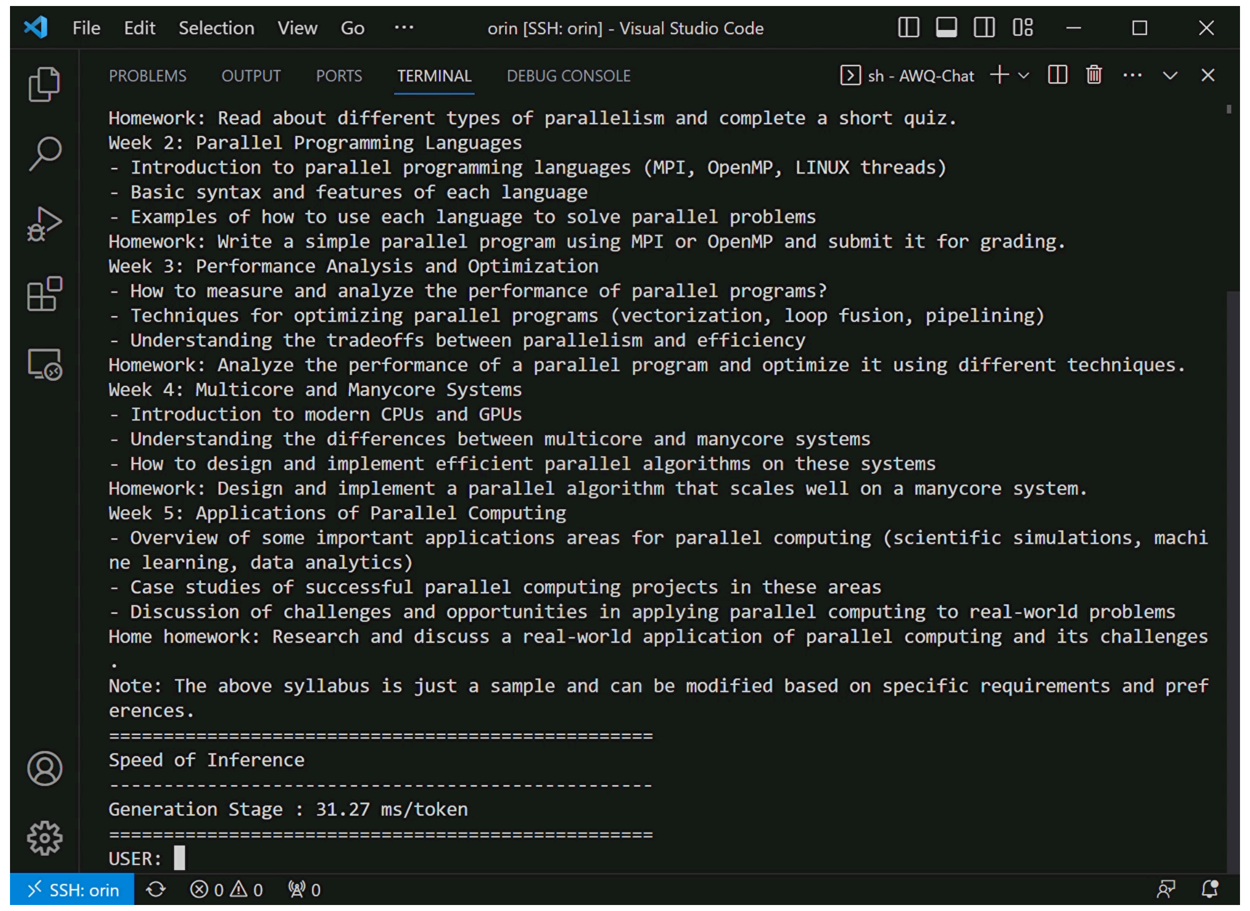
TinyChat

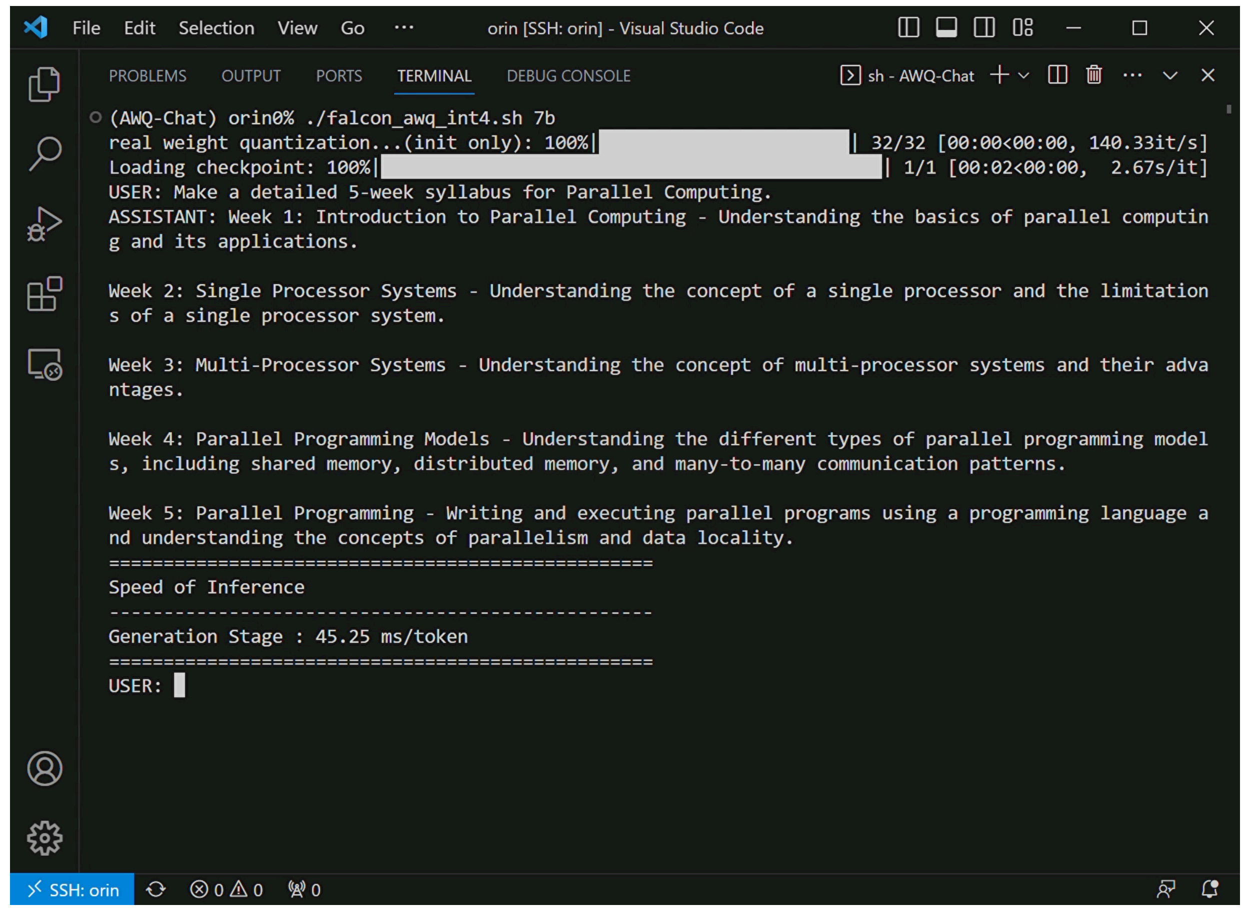
TinyChat flexibly supports different LLM architectures

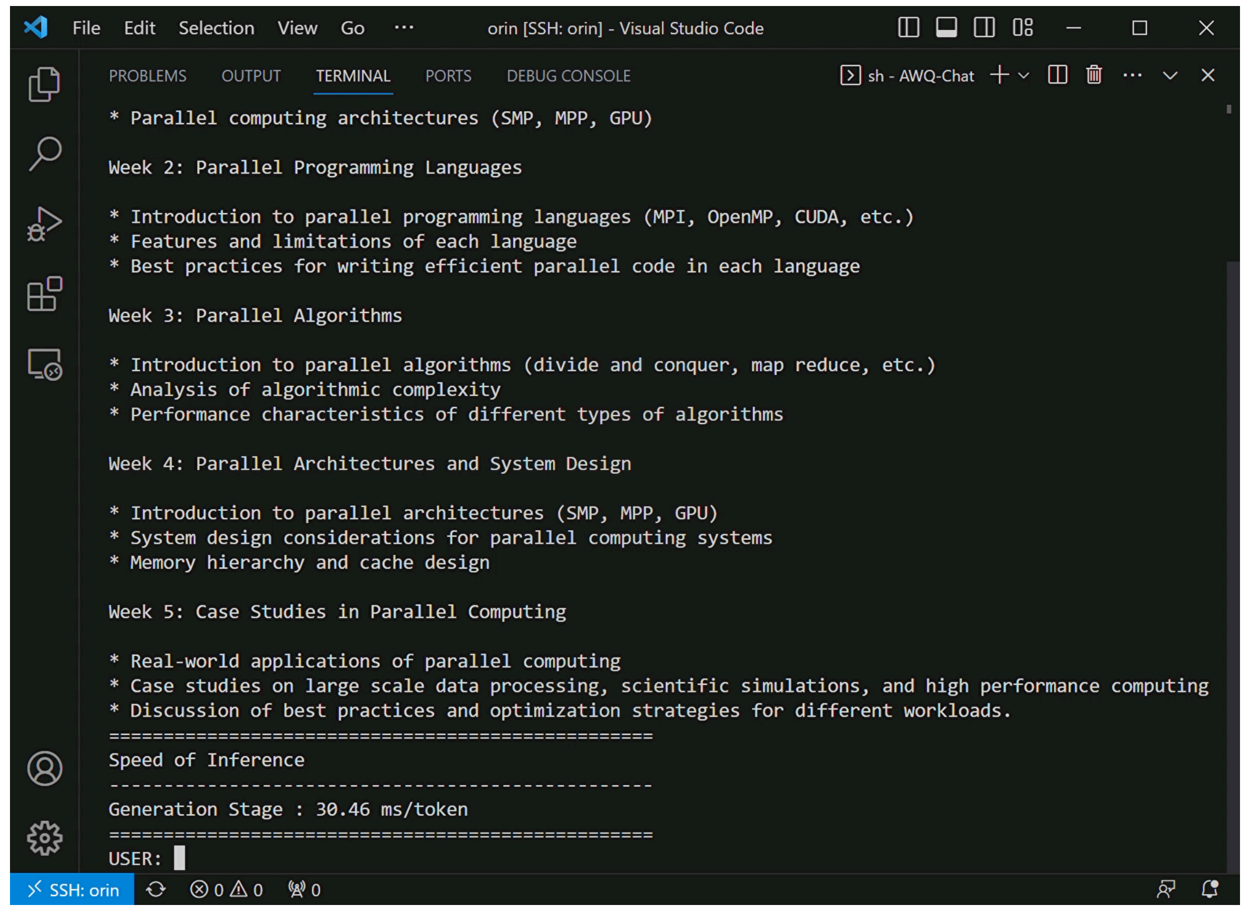












mosaic[ML]

MPT-7B: 31 tokens / s

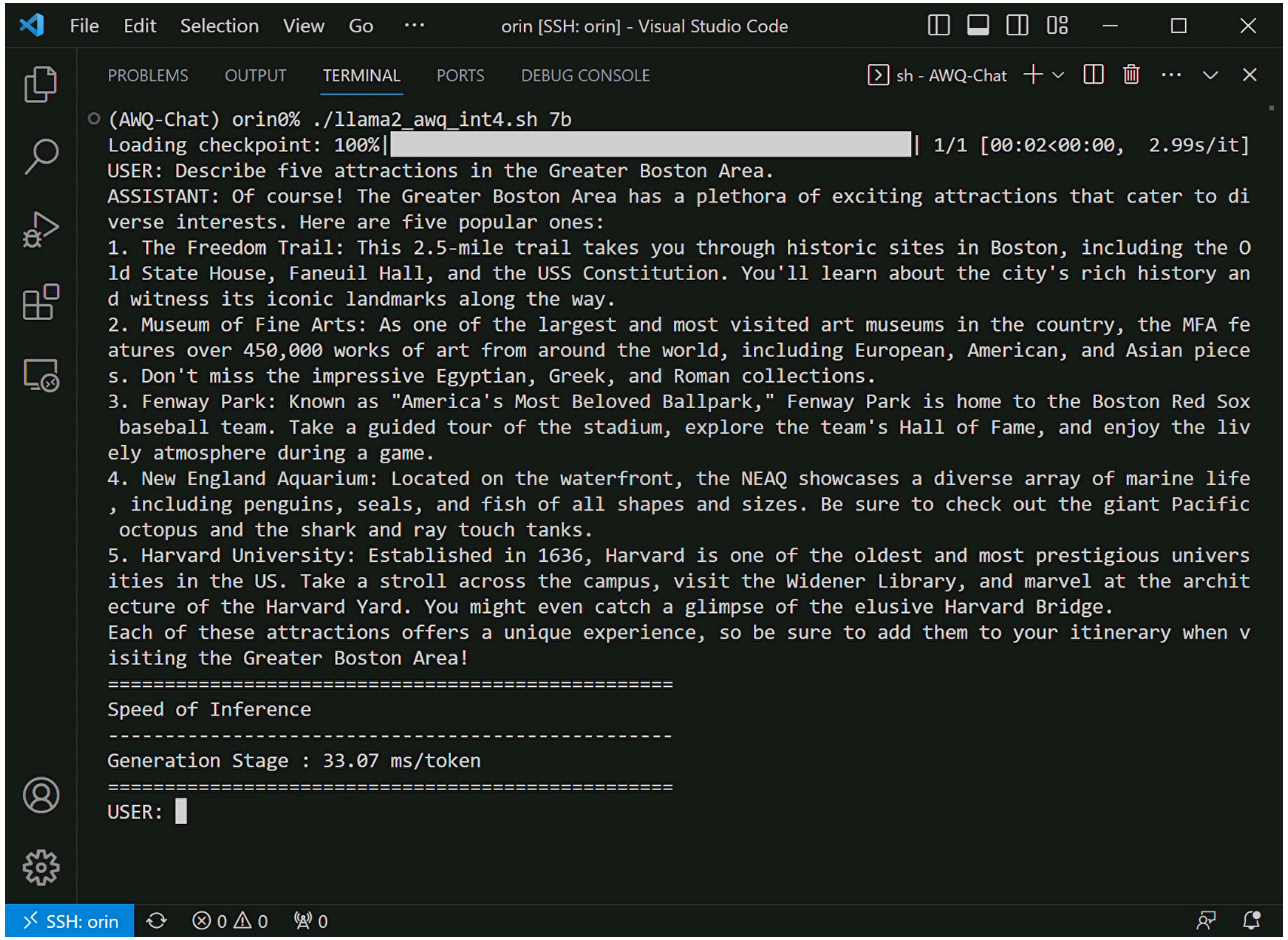
Technology Innovation Institute

Falcon-7B: 22 tokens / s

Vicuna-7B: 33 tokens / s

TinyChat

TinyChat delivers 30 tokens / second performance for LLaMA2



LLaMA-2-7B (W4A16, AWQ): 30 tokens / s

https://github.com/mit-han-lab/llm-awq

Thanks for Listening!