PowerInfer: Fast Large Language Model Serving with a Consumer-grade GPU

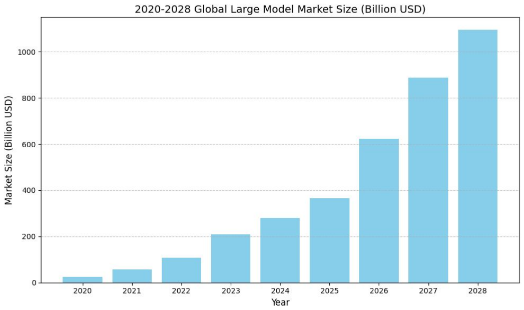
Group: **Haiquan Wang**, Jiaqi Ruan and Jia He

2024-12-10

Agenda

- ❑ Background
- ❑ Related Work
- ❑ Motivation
- ❑ PowerInfer
- ❑ Evaluations

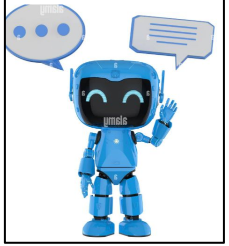
Background - The trends of LLMs



The market for large models will continue to grow in the future

Admitting the top 2,000 students who are best at using ChatGPT is a very interesting approach

Chat Robot



Text2Img

Code Assistant

...

Applications of LLMs are continuously emerging

Background - LLM inference

- The process starts with a prompt and unfolds in two phases :
  - The prompt phase outputs an initial token
  - the generation phase sequentially produces tokens until a maximum limit or an end-of-sequence (<EOS>) token is reached

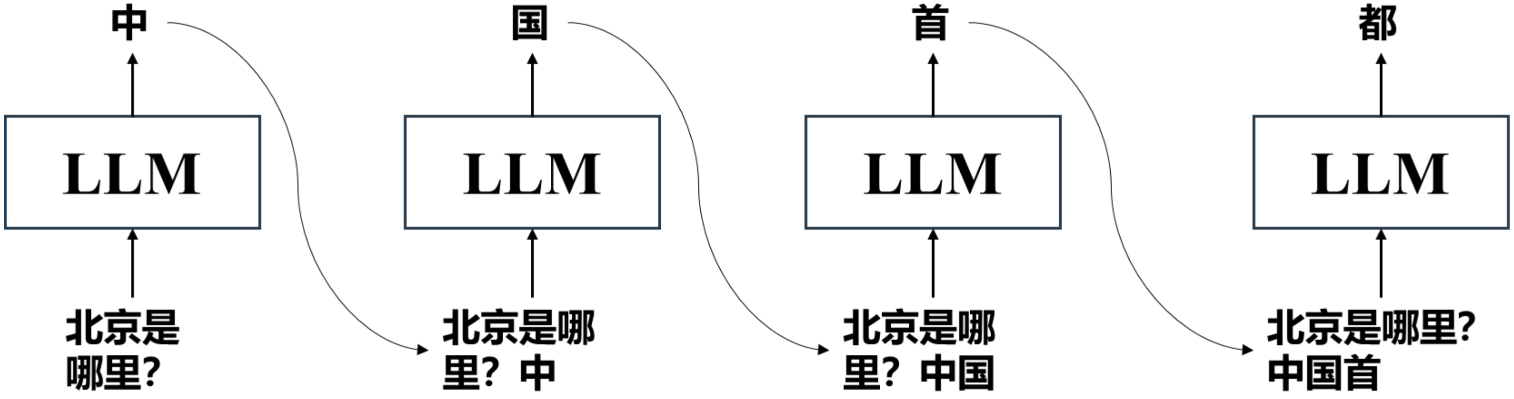
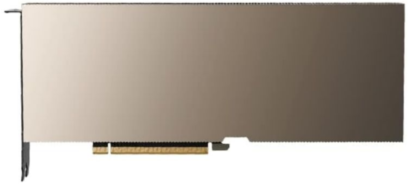


Figure cited from Gong Ping's group

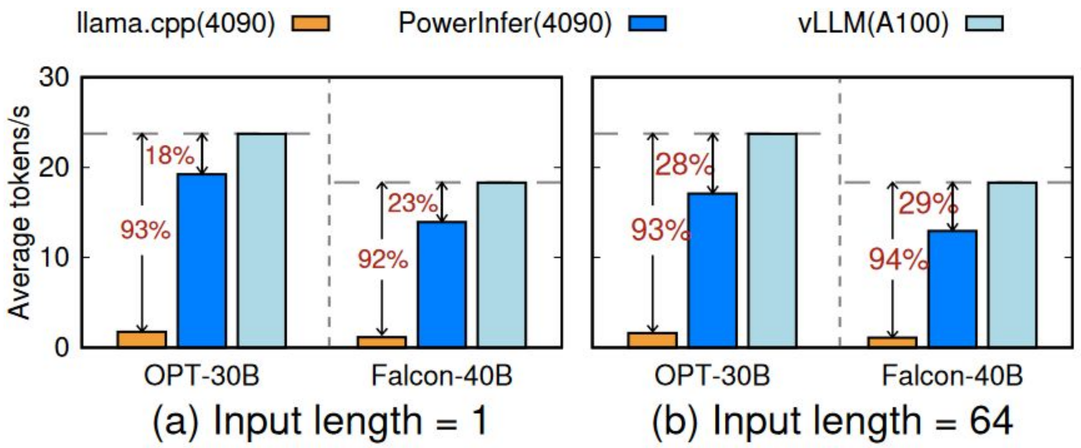
Background - LLM infer with consumer-grade GPU

		Memory	Peak TFLOPs (FP16)	Price	
4090		24 GB	330	$2300	
A100		80 GB	312	$18000	

By using PowerInfer, the **Llama 70B** model can be deployed on the 4090. Combined with quantization techniques, it can further support inference with the **OPT 175B** model.

Background - LLM infer with consumer-grade GPU

Background - LLM infer with consumer-grade GPU



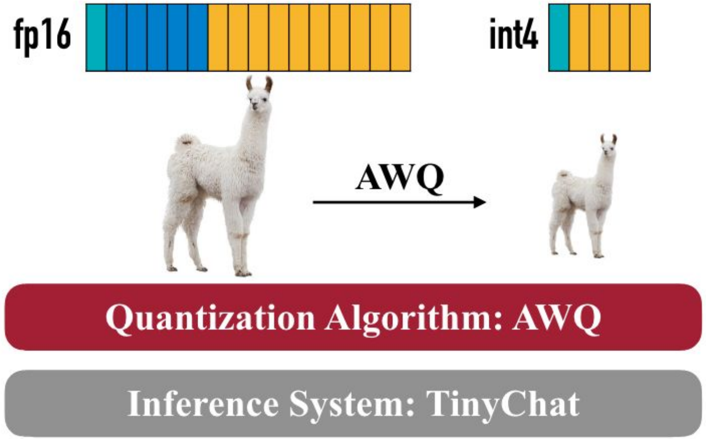
- **Spending 12% of the money can achieve 70% to 80% of the performance of an A100.**
- **Although llama.cpp can run on the 4090, its performance is poor.**

Agenda

- ❑ Background
- ❑ Related Work
- ❑ Motivation
- ❑ PowerInfer
- ❑ Evaluations

Related Work

- **Model Quantization**
  - Reduces the bit-precision of deep learning models which helps to reduce the model size and accelerate inference
  - GPTQ（ICLR 23’）、MARLIN（MLSys 24’）、AWQ（MLSys 24’）

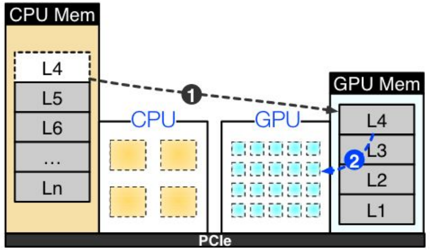


Drawback: Even deeply compressed models remain too large for consumer-grade GPUs. SOTA Method AWQ can only hold up to 30B LLM on RTX4090 while PowerInfer can hold up to 70B without additional quantization.

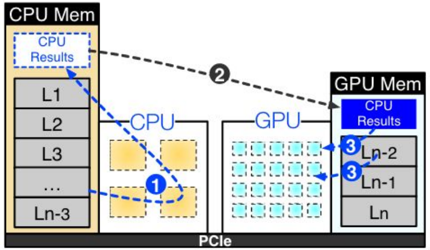
Related Work

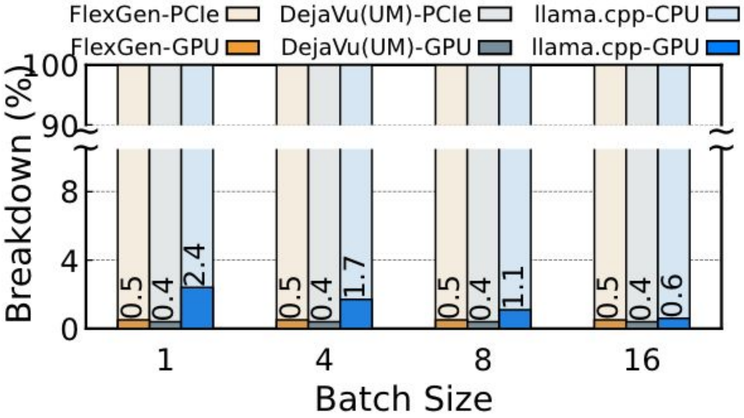
- **Offloading-based LLM Inference**
  - **GPU-Centric Offloading (FlexGen、DejaVu(UM)、SpecInfer)**
    - Use CPU memory to store model parameters, computation only happens on GPU
    - **Drawback: Huge data transfer overhead between CPU and GPU**
  - **GPU-CPU Hybrid Offloading (Llama.cpp)**
    - Use both CPU and GPU to store transformer layers and do **serial** computation
    - **Drawback: Large dense computation on CPU results suboptimal latency**

GPU-Centric Offloading



GPU-CPU Hybrid Offloading



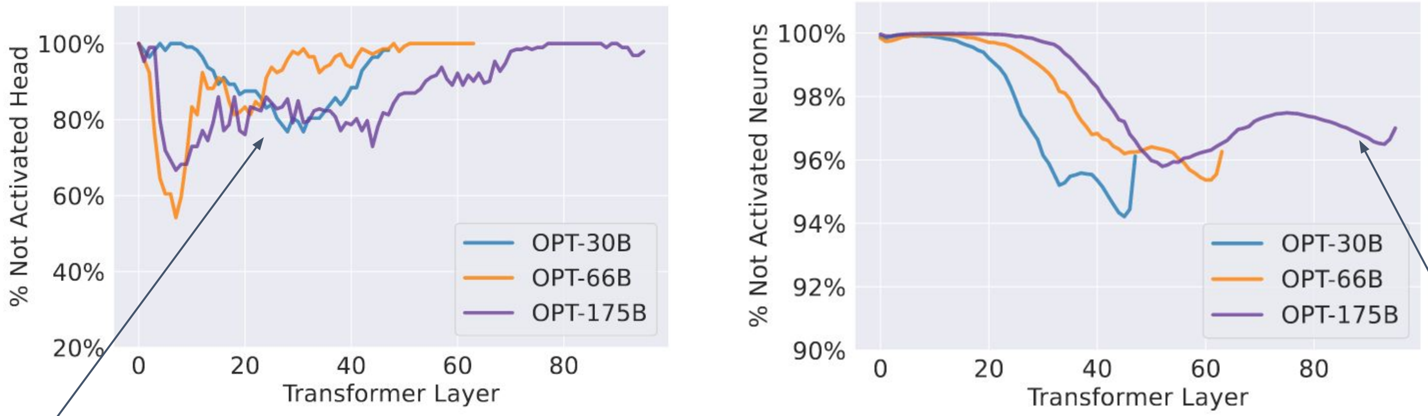


Agenda

- ❑ Background
- ❑ Related Work
- ❑ Motivation
- ❑ PowerInfer
- ❑ Evaluations

Sparsity in LLM Inference

LLM inference shows a contextual sparsity where **small, input-dependent sets of attention heads and MLP neurons[1] lead to (nearly) the same output as the full model for any input**.



(a) Contextual sparsity in Attention Head

(b) Contextual sparsity in MLP Block

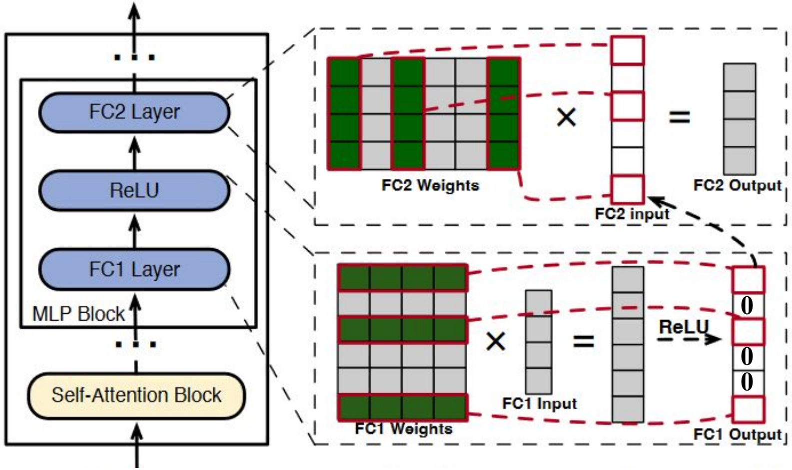
Up to 80% sparsity on attention heads on average

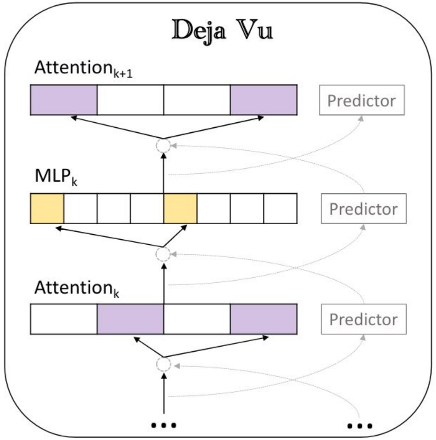
Up to 95% sparsity on MLP neurons on average

[1]: neuron can be defined as a specific row/column in a weight matrix

Sparsity Prediction

- **Dive into Sparsity**
  - **Reason (take MLP layer as an example)**
    - Activation Functions like ReLU selectively influence neuron activations in MLP
  - **Effect**
    - **FLOPs** can be drastically **reduced** by **predicting** non-important computations and avoiding them, thus speeding up the inference process. DejaVu has done such work





Predictors here are small MLPs

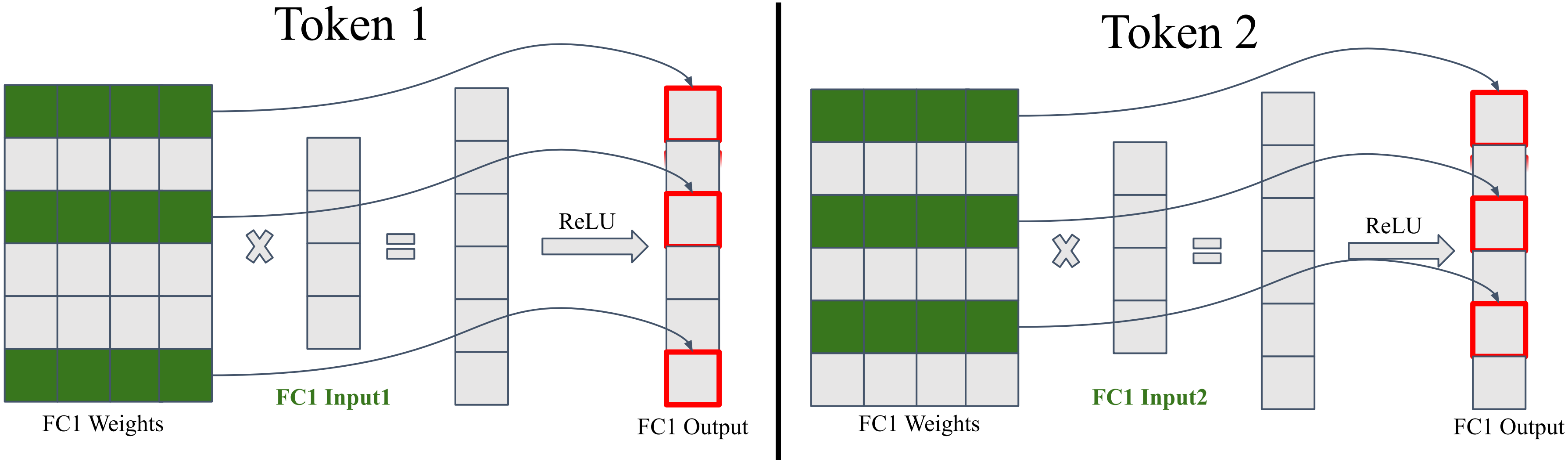
Sparsity is prevalent

LLM	Activation Function	Sparsity
OPT-30B	ReLU	97%
LLaMA2-13B	SwiGLU	43%
Yi-34B	SwiGLU	53%

- For ReLU-based models, sparsity is the proportion of neurons with zero activation.
- For SwiGLU-based models, it's the proportion of neurons that can be dynamically pruned with less than 1% impact on perplexity.

Locality in LLM Sparse Inference

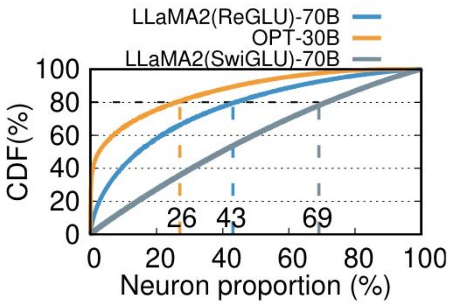
- **Locality**
  - **A consistent group of neurons is frequently activated**



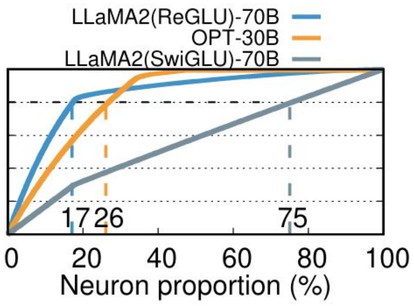
Hot neurons are always activated across all inputs, while cold neurons are not.

Locality in LLM Sparse Inference

- The Locality in LLM Sparse Inference

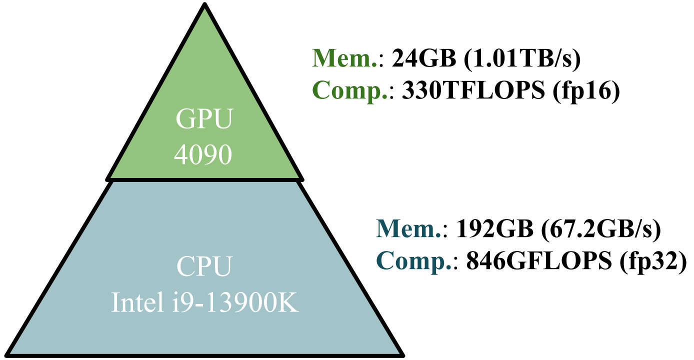


(a) MLP Block



(b) Entire Model

- GPU-CPU Hierarchical Architecture



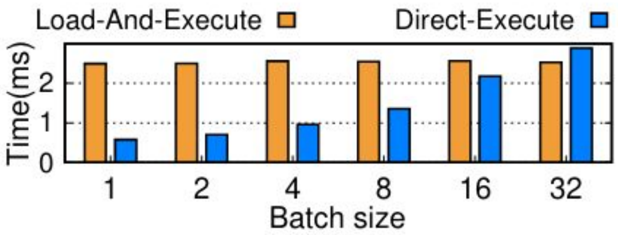
Small subset of hot neurons is **always activated** across various inputs ⟺ GPU has **less** memory but computes **fast**

Majority cold neurons are **selectively activated** based on the inputs ⟺ CPU has **more** memory but computes **slow**

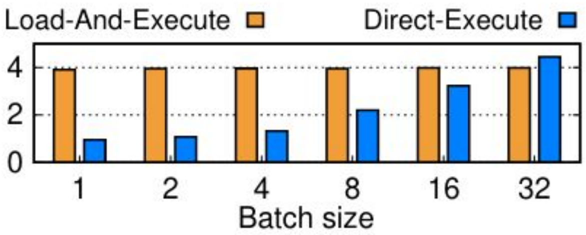
Matching them together: Hot neurons should be placed on GPU while cold neurons on CPU

Locality in LLM Computation

- Fast In-CPU Computation
  - with the **small number of activated neurons** and the **small batch sizes** typical in local deployments, **computing activated cold neurons on the CPU is faster than transferring and computing them on the GPU**



(a) MLP block

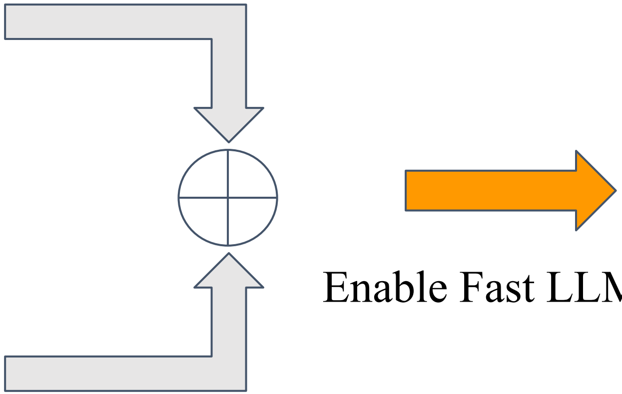


(b) Attention block

What If We Combine Them Together…

Sparsity

Reduce FLOPS



PowerInfer

Enable Fast LLM Inference on Consumer-Grade GPU

Locality

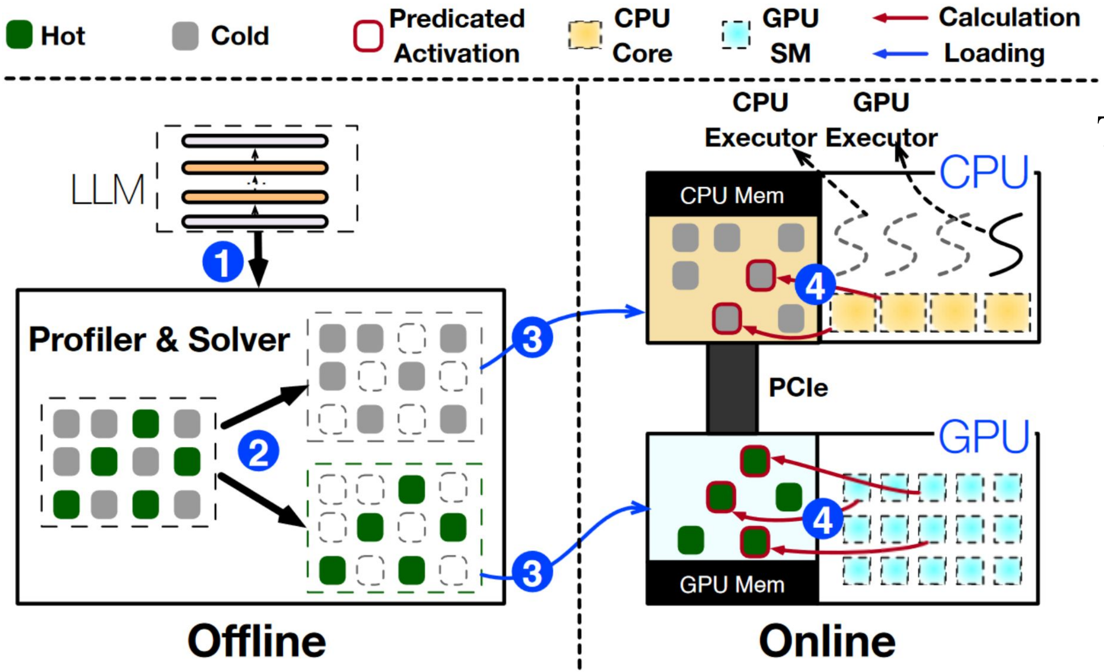
Offload Storage and Computation of Cold Neurons

Agenda

- ❑ Background
- ❑ Related Work
- ❑ Motivation
- ❑ PowerInfer
- ❑ Evaluations

Design - PowerInfer Overview

- PowerInfer
  - Exploit the high sparsity and locality inherent in LLM inference



Technique Contributions:

1. How to determine neuron placement
2. How to efficiently execute sparse operators in a GPU-CPU hybrid manner

Design - Neuron Placement Policy

- **How to decide which neuron is hot or cold?**
  - Insight: the hot neurons in general corpus are also activated frequently across different scenarios
  - Profile the activation information of neurons across multiple general datasets
  - Hot or cold neuron is defined by its activation frequency obtained during profiling

Design - Neuron Placement Policy

❑ How to decide neuron placement?

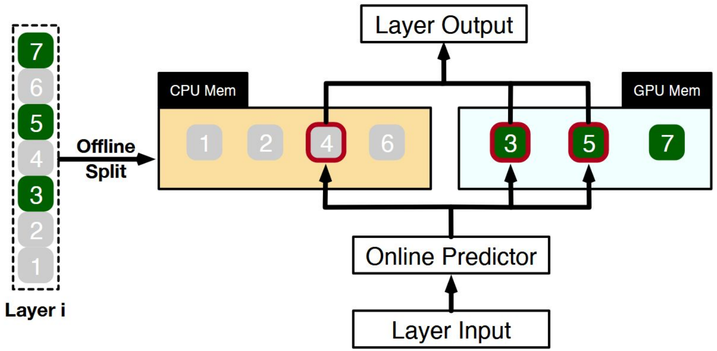
Target: Maximize the placement of high-frequency neurons[1] on the GPU

Constraint
- Comm ⟹ neurons on CPU time - neurons on GPU time[2] $\geq$ sync
- Memory ⟹ the memory of neurons $\leq$ GPU and CPU capability

[1]: To expedite the process, aggregating neurons within each layer into groups.
[2]: In LLM inference, particularly with smaller batch sizes and high sparsity, the limiting factor is memory bandwidth.

Design - Adaptive Sparsity Predictors

The online predictor reduces comp by processing only the predicted activated neurons.



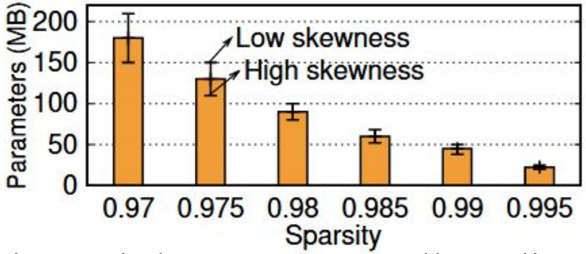
a useful online predictor
- stored in GPU（low latency）
- Large (accuracy)

occupy gpu memory → 4090 OOM!

How to add Online Predictor on a single consumer gpu?

Design - Adaptive Sparsity Predictors

Insight:



With high model sparsity and skewness, a small predictor can achieve the same accuracy.

PowerInfer predictor:

Each layer has adaptive iterative training non-fixed-size predictors[1].

Adaptive
- The sparsity of the layer decide the origin predictor hidden size
- High skewness[1] reduce predicator hidden size, vice versa

Iterative Training stops when the model's perplexity with the predictor approximates that of the baseline model.

Extra memory footprint: 6%

[1] Predictors training on the WikiText-2 dataset
[2] High skewness indicate that neuron activation values are too concentrated or sparse, limiting the neural network's expressive capacity.

Design - Neuron-aware Operator

The Neuron-aware Operator reduces comp by directly computing activated neurons

why need design Neuron-aware operator

SOTA sparse-aware[1]
- static compilation
- require dynamic conversion of sparse to dense
- not support GPU-CPU hybrid execution

PowerInfo scenario: Dynamic sparsity, GPU-CPU hybrid execution

PowerInfo sparse operator:

GPU: Focus on individual row/colomn vector computation (vector computation are advantageous in small batch)

CPU: Assign a neuron-aware operator to multiple cores, dividing neurons into smaller batches for concurrent activation checking and hardware vector extensions like AVX2[2] optimizing.

[1]: SparTA, FlashLLM, cuSPARSE, PIT
[2]: AVX2 is a CPU instruction set extension that accelerates integer and floating-point operations

Agenda

- ❑ Background
- ❑ Related Work
- ❑ Motivation
- ❑ PowerInfer
- ❑ Evaluations

Evaluations - Setup

- **Hardware:**
  - **PC-High:** Intel i9-13900K processor (eight 5.4GHz cores), 192GB host memory (bandwidth of 67.2 GB/s), an NVIDIA RTX 4090 GPU (24GB), and PCIe 4.0 interface (64GB/s bandwidth)
  - **PC-Low:** Intel i7-12700K processor (eight 4.9GHz cores), 64GB host memory (bandwidth 38.4 GB/s), an NVIDIA RTX 2080Ti GPU (11GB), and PCIe 3.0 interface (32GB/s bandwidth)
- **Models:**

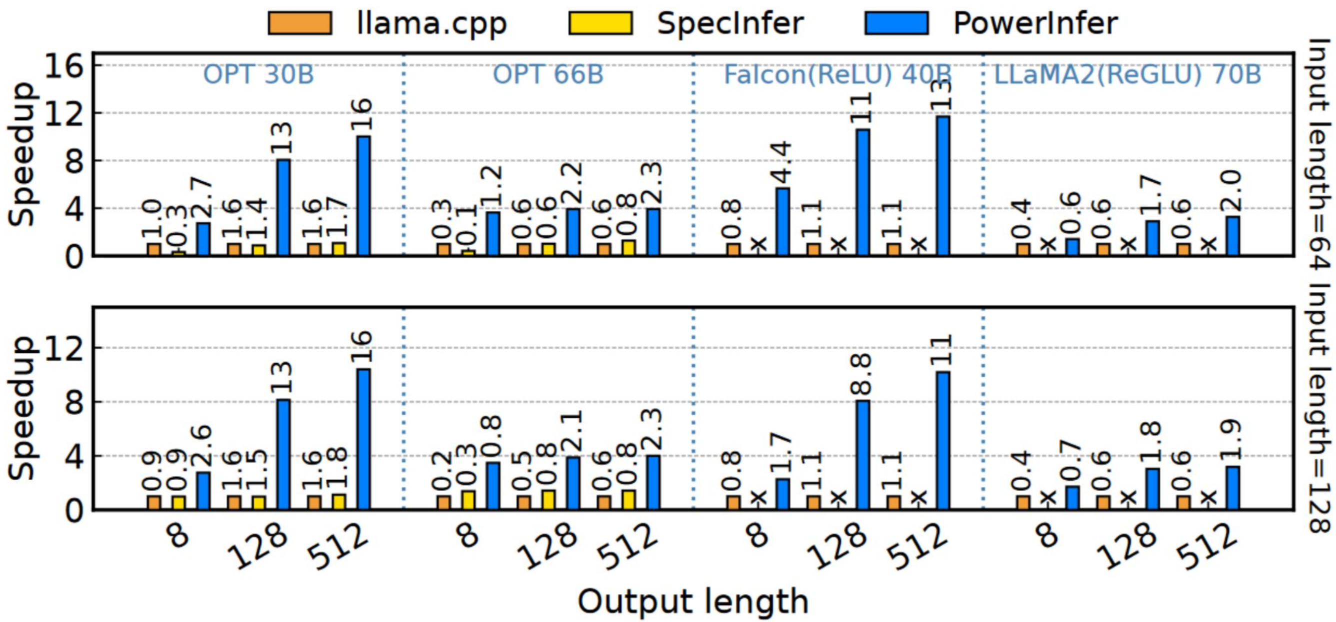
Model	Activation Function	Sparsity
Bamboo-7B	dReLU	90%
OPT-7B/13B/30B/66B/175B	ReLU	96%-98%
Falcon(ReLU)-40B	ReLU	95%
LLaMA2(ReGLU)-7B	ReGLU	70%
LLaMA2(ReGLU)-13B	ReGLU	78%
LLaMA2(ReGLU)-70B	ReGLU	82%
Qwen1.5-4B	SwiGLU	40%
LLaMA2(SwiGLU)-13B	SwiGLU	43%
Yi-34B	SwiGLU	53%

Evaluations - Baseline

- **Batch size:**
  - 1 in overall experiment
- **Baseline:**
  - llama.cpp: LLM inference framework for local scenarios with GPU-CPU hybrid offloading
  - SpecInfer: Support speculative inference and GPU-Centric offloading

Evaluations - Overall Results

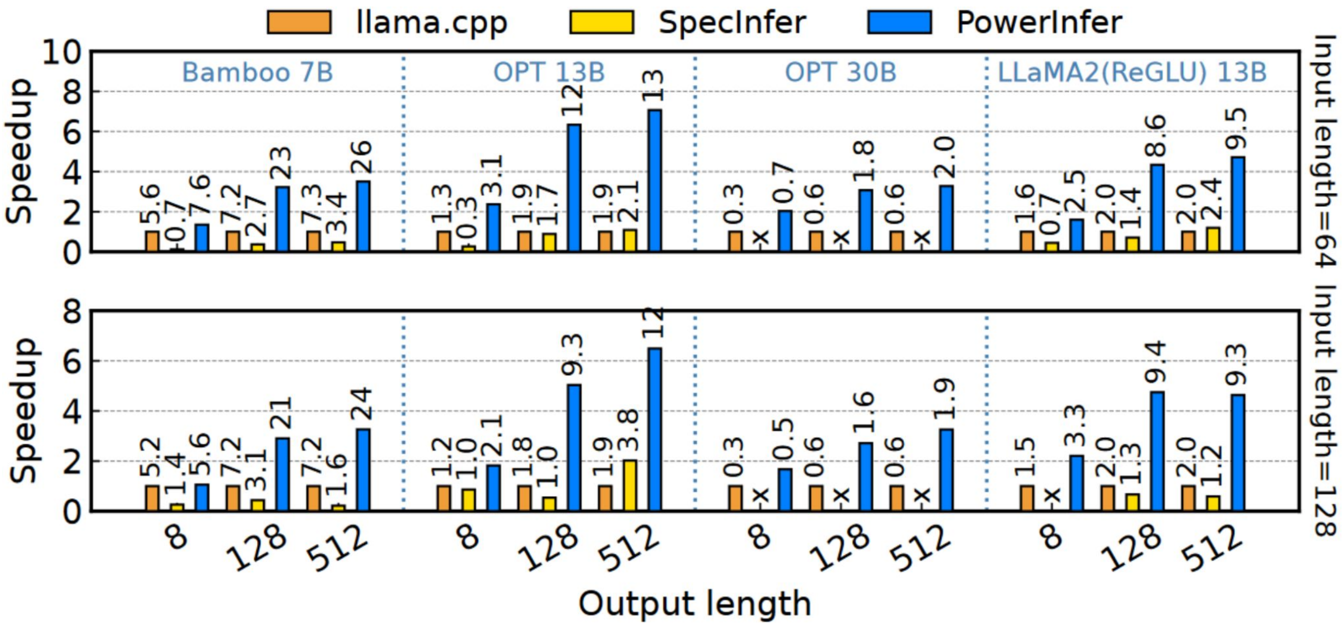
PC-High



- outperforming llama.cpp and SpecInfer with average speedups of **7.23×** and **6.19×**
- Longer output seq length, higher speedup

Evaluations - Overall Results

PC-Low



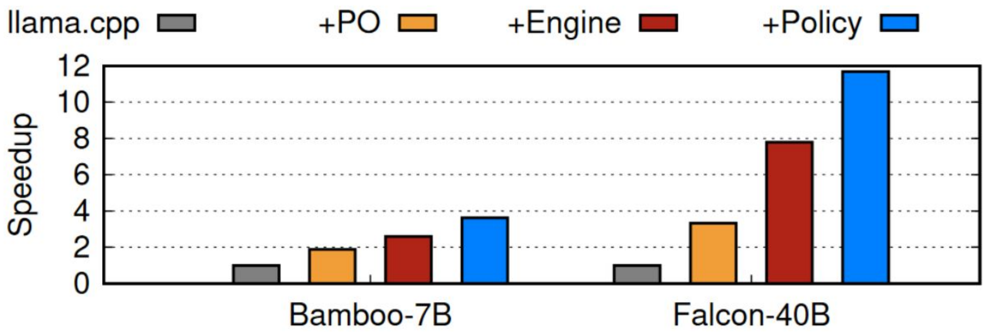
- performance enhancement over llama.cpp and SpecInfer, averaging a speedup of 4.71×, 5.97 × and peaking at 7.06× and 7.47×

Evaluations - Accuracy

	PIQA	Winogrande	Arc-Challenge	MMLU	GSM8K	Average
OPT-7B	75.78%	65.19%	30.63%	24.95%	1.90%	39.69%
OPT-7B-PowerInfer	75.67%	65.51%	30.63%	24.73%	1.82%	39.67%
OPT-13B	76.01%	64.96%	32.94%	25.02%	2.12%	40.21%
OPT-13B-PowerInfer	76.28%	65.98%	33.19%	24.76%	2.20%	40.48%
LLaMA(ReGLU)-13B	76.44%	70.09%	36.52%	50.21%	25.40%	51.73%
LLaMA(ReGLU)-13B-PowerInfer	74.06%	69.93%	36.60%	49.47%	23.90%	50.79%
Falcon-40B	81.23%	75.45%	50.68%	51.78%	21.99%	56.23%
Falcon-40B-PowerInfer	81.01%	75.92%	50.68%	51.68%	20.45%	55.95%
LLaMA(ReGLU))-70B	82.01%	75.93%	52.39%	62.30%	62.30%	66.99%
LLAMA(ReGLU)-70B-PowerInfer	82.05%	75.53%	51.45%	61.90%	61.90%	66.57%

PowerInfer causes negligible loss in inference accuracy, regardless of the model size or type of task

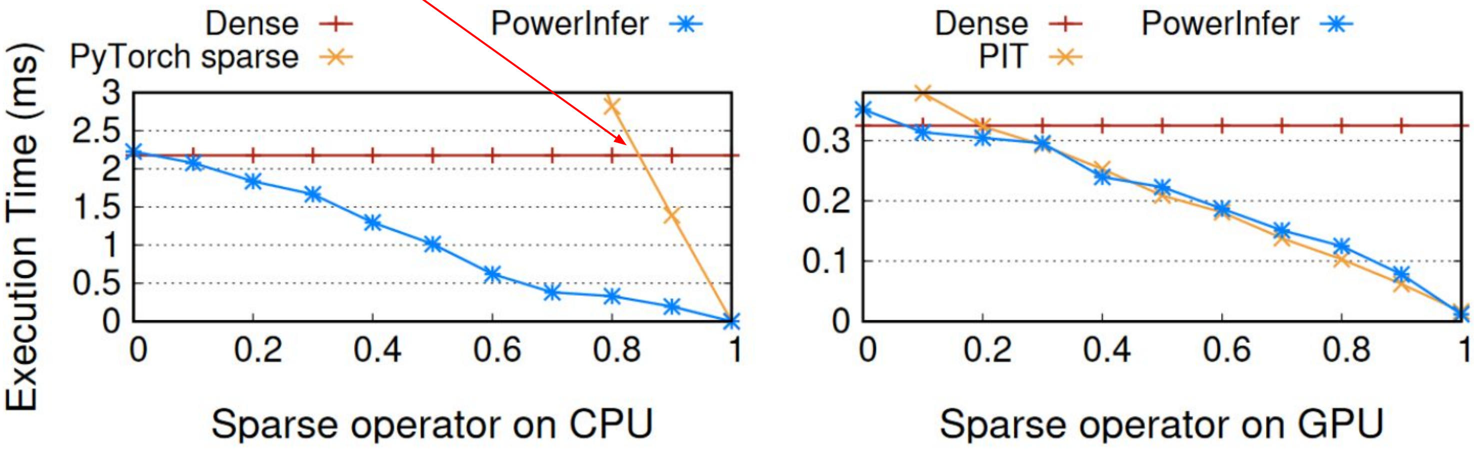
Evaluations - Performance Breakdown



- PO: add PowerInfer’s predictors and neuron-aware operators into llama.cpp
- Engine: PowerInfer’s hybrid inference engine
- Policy: integrating our optimized policy

Evaluations - Neuron-aware Operators

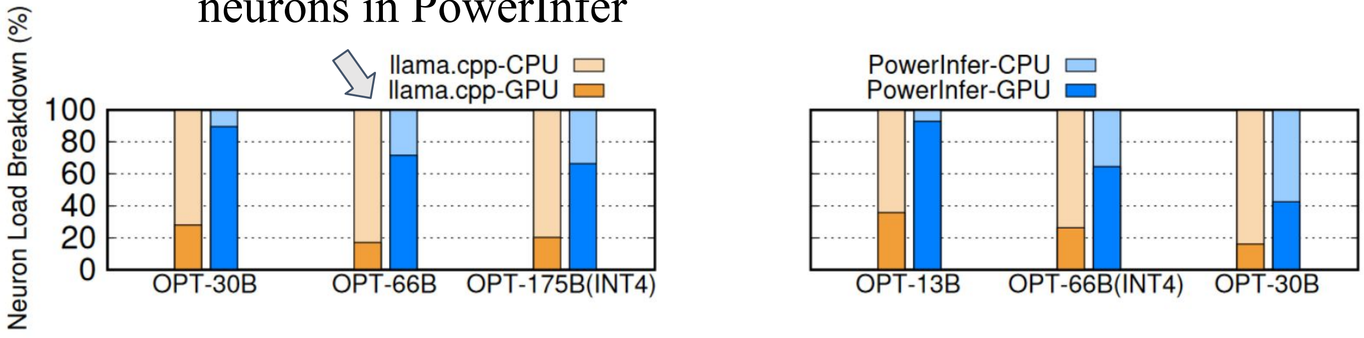
Traditional sparse operators do not outperform dense computation until sparsity surpasses 87%



Focus on sparse matrix-vector multiplication using a [4096, 4096] × [4096, 1]

Evaluation - Neuron Load Distribution

GPU processes 70% of activated neurons in PowerInfer

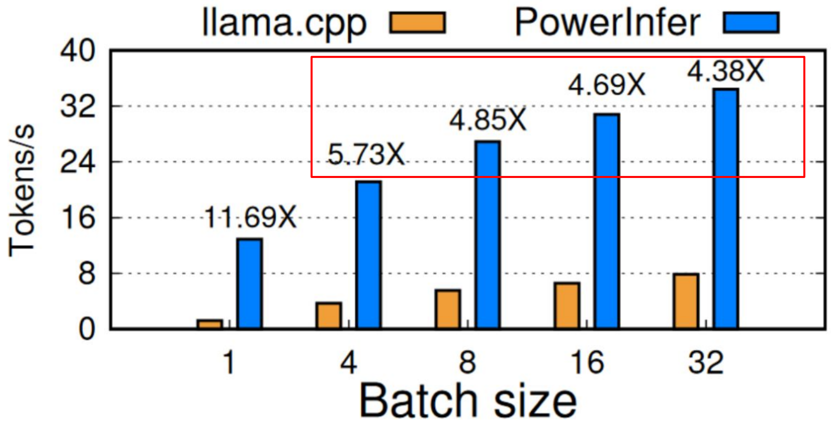


(a) PC-High

(b) PC-Low

The GPU's neuron load is reduced to 42% due to limitations in GPU memory.

Evaluation - Batch Inference



As the batch size increases, the speed-up ratio offered by PowerInfer decreases.

Conclusion

Problems:

- The necessity of the current design under a large batch size
- Longer sequences will result in greater KV cache storage overhead, and KV cache offloading also needs to be considered
- If the model's memory demand greatly exceeds GPU memory, most neurons will be placed on the CPU, diluting PowerInfer's benefits.

End

- Thank you!
- Q&A