Can Large Language Models Predict Parallel Code Performance?

Gregory Bolet[1], Giorgis Georgakoudis[2], Harshitha Menon[2], Konstantinos Parasyris[2], Niranjan Hasabnis[3], Hayden Estes[1], Kirk W. Cameron[1], Gal Oren[4]

HPDC 2025 – AI4Sys Workshop

July 20, 2025

[1]Virginia Tech

[2]Lawrence Livermore National Laboratory (LLNL)

[3]Code Metal AI

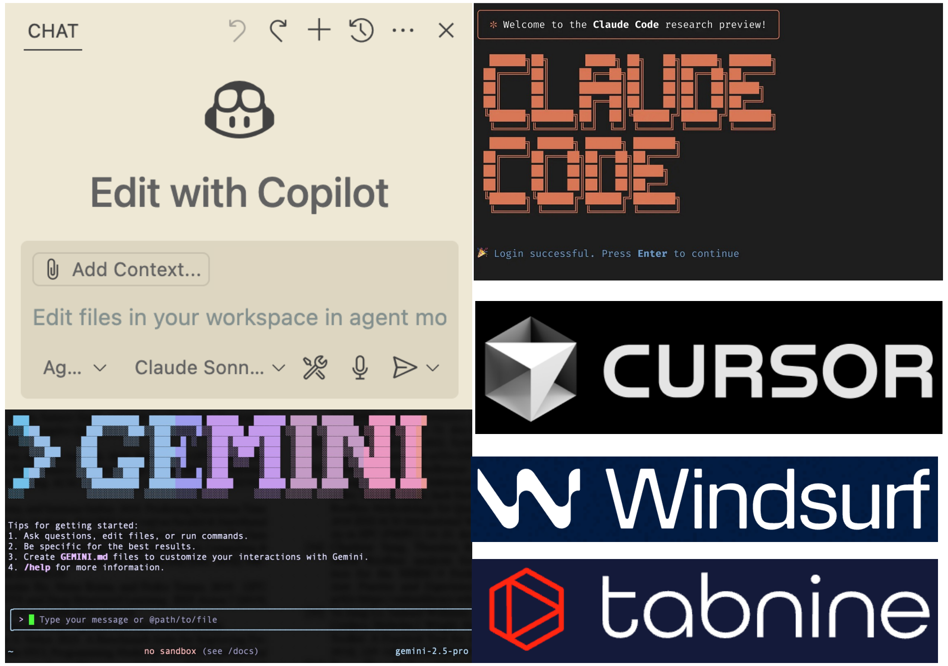
[4]Technion & Stanford

LLNL-CFPRES-2008561

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

A confluence of trends motivated this work

1) Normalization of LLM-based "assistants" in software development



2) circa Aug 2024, not many Performance Analysis (PA) sub-fields using **LLMs** for GPU performance prediction

- Compiler Optimization
- Log Parsing
- Code Transpilation (OpenMP)
- HPC Advising Tools
- Debugging Assistants
- Code Optimization

3) Existing works assumed hardware access for GPU profiling

Profile 1
Profile 2
⋮
Profile N
→ Fine-tuned LLM

OR

Unoptimized Code → Profile Performance → LLM optim. suggestions → Update Code → Profile Performance

Idea: Can LLMs predict GPU code performance ***without*** the need for hardware/profiling?

What would be a “simple” PA task we could ask of the LLMs?

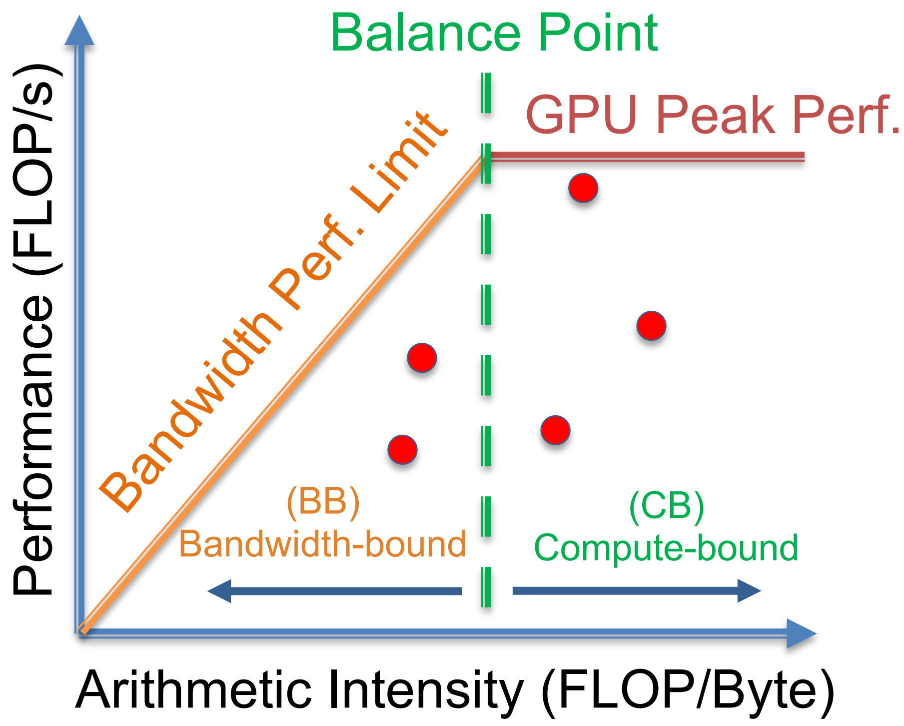
🌈 🌟 Roofline Model 🌟 🌈

Arithmetic Intensity Classification

🤔

The Roofline Model guides code optimization

$$a = \texttt{Max Bandwidth (GB/s)}$$
$$b = \texttt{Peak Performance (GFLOP/s)}$$
$$\boxed{y = \min(xa, b)}$$



Optimizations if (code == BB)
- cudaMemCpy only necessary data
- Data/cache re-use via smart access pattern
- Sparsity / strided-access reduction

Optimizations if (code == CB)
- Intrinsics (e.g: Fused-Multiply-Add -- FMA)
- Switching precisions / datatypes
- Loop unrolling
- Avoid implicit operations (e.g: division)

Idea: What Roofline metrics can we get LLMs to predict for us?

Predicting exact Roofline metrics with LLMs is hard

- **Roofline *Regression* Task:**

  Get an LLM to predict Arithmetic Intensity (AI) (FLOP/Byte) or Performance (FLOP/s) from source code?

  ❌ LLMs are not good at regression (yet…)

- **Roofline *Classification* Task:**

  Get an LLM to ***classify*** Arithmetic Intensity from source code?

  ✅ LLMs can do classification

Proven LLM Classification Tasks

sentiment analysis

spam detection

Key Question: How *well* can an LLM classify Roofline AI of GPU codes?

Arithmetic Intensity (AI) Classification Research Questions

RQ1 (Baseline Roofline Classification)

- Can LLMs classify AI well when given the hardware roofline, and arithmetic intensity values?

RQ2 (Zero-Shot Classification)

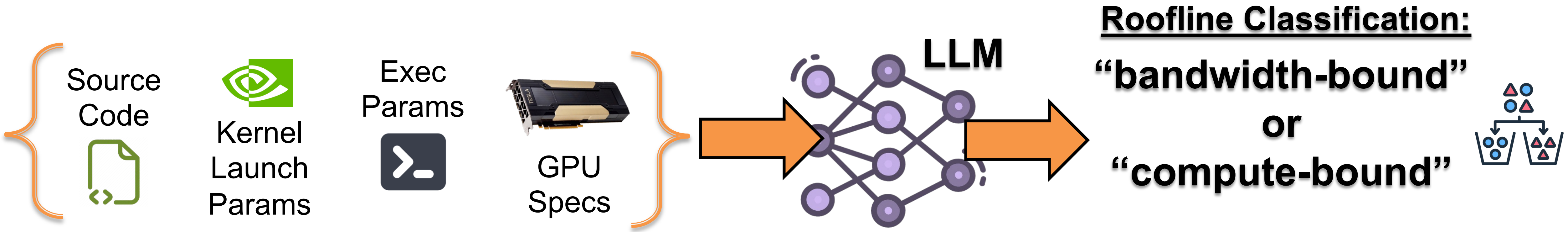
- Can LLMs classify AI well when given source code, execution specs, and minimal instructions?

RQ3 (Few-Shot Classification)

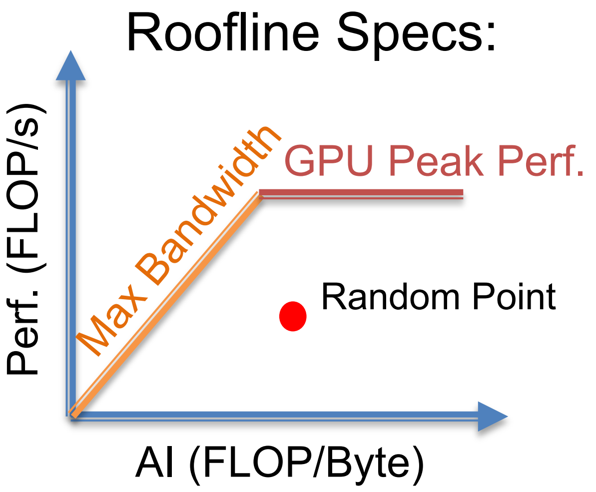
- Can LLMs classify AI well when given source code, execution specs, and a few *real* examples of codes with their expected classifications?

RQ4 (Fine-Tuned Classification)

- Can we fine-tune LLMs for roofline AI classification?



SoTA LLMs understand Arithmetic Intensity pretty well (RQ1)



120 BB questions
120 CB questions
Random Rooflines

Model Name	Reasoning	RQ1 Acc.	RQ1 CoT Acc.
o3-mini-high	✓	**100**	**100**
o1	✓	–	–
o3-mini	✓	**100**	**100**
gpt-4.5-preview		–	–
o1-mini-2024-09-12	✓	**100**	**100**
gemini-2.0-flash-001		91.25	92.50
gpt-4o-2024-11-20		91.25	96.25
gpt-4o-mini		90.00	**100**
gpt-4o-mini-2024-07-18		90.00	**100**

RQ1 CoT Prompting Template

2, 4, 8-shot examples
w/ optional (chain-of-thought) CoT
(redacted)

Question: Given a GPU having a global memory with a max bandwidth of 99.9 GB/s and a peak performance of 73.45 GFLOP/s, if a program executed with an Arithmetic Intensity of 1.55 FLOP/Byte and a performance of 32.8 GFLOP/s, does the roofline model consider the program as compute-bound or bandwidth-bound?

Findings:
- All models have a reasonably-good understanding of AI
- Reasoning models have good prediction accuracy w/ and w/out CoT
- 2 prompt examples is sufficient

GPU Program Source Code Dataset Creation

https://github.com/zjin-lcf/HeCBench

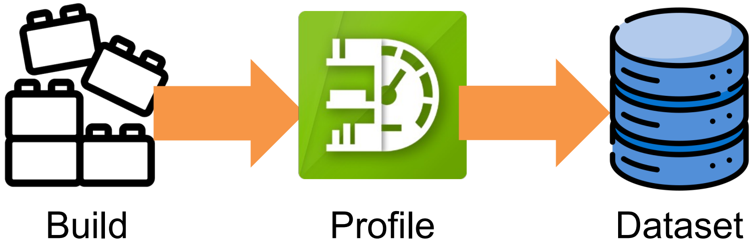
zjin-lcf / **HeCBench**

248 stars 89 forks 4 watching

3 Branches 0 Tags Activity

Public repository

170 CUDA Programs
170 OpenMP Programs



Collected Attributes:
- Program Name
- Target Kernel Name
- Source Code
- Hardware Specs Info
- Executable Args
- Launch Grid / Block Size
- Arithmetic Intensity (AI) Values
- AI Classification (CB/BB)

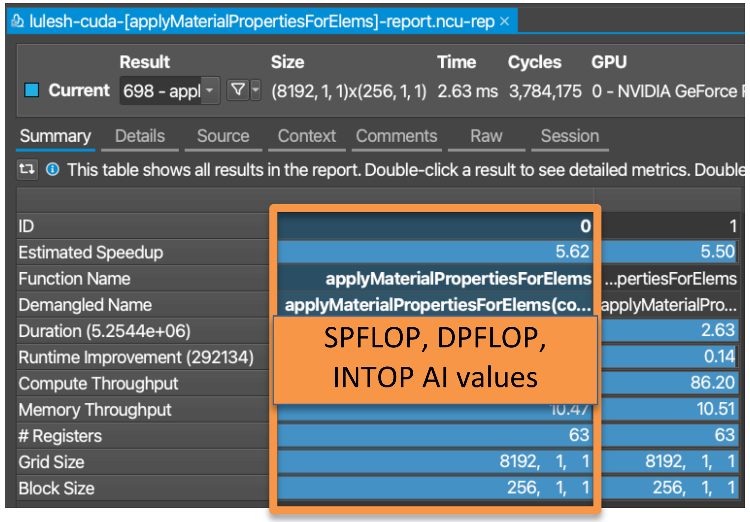
Data Collection Design Decisions:

1) Sampled metrics on NVIDIA RTX 3080 GPU

2) Concatenate all source files for prompting

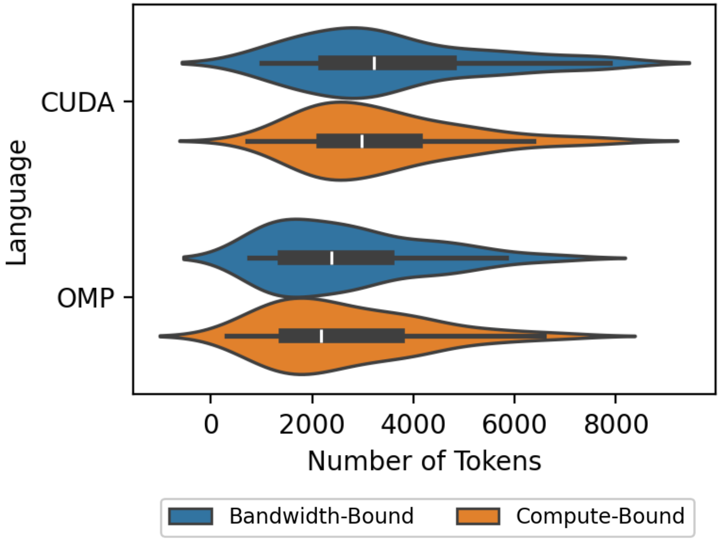
```
-----------------
my_header.h
-----------------
/* source code here */

-----------------
my_cuda_kernels.cu
-----------------
/* source code here */

-----------------
main.cu
-----------------
/* source code here */
```

3) Profiled only 1st execution of 1 kernel per program



ID	0	1
Estimated Speedup	5.62	5.50
Function Name	applyMaterialPropertiesForElems	...pertiesForElems
Demangled Name	applyMaterialPropertiesForElems(co...	applyMaterialPro...
Duration (5.2544e+06)		2.63
Runtime Improvement (292134)		0.14
Compute Throughput		86.20
Memory Throughput	10.47	10.51
# Registers	63	63
Grid Size	8192, 1, 1	8192, 1, 1
Block Size	256, 1, 1	256, 1, 1

4) Balanced dataset w.r.t: token counts, language, AI class



Reasoning-based LLMs are the best at predicting AI (RQ2)

Hardware Roofline Specs

Execution Specs

RQ2 Prompting Template (see paper for full prompt)

[omitted context-setting beginning of prompt]

Classify the [language] kernel called [kernel name] as **Bandwidth** or **Compute** bound. The system it will execute on is a [GPU model] with:

- peak single-precision performance of [X] GFLOP/s
- peak double-precision performance of [X] GFLOP/s
- peak integer performance of [X] GINTOP/s
- max bandwidth of [X] GB/s

The block and grid sizes of the invoked kernel are (X,Y,Z) and (X,Y,Z), respectively. The executable running this kernel is launched with the following command-line arguments: `[arg1 arg2 arg3]`.
Below is the source code of the requested [language] kernel:

[concatenated source code files]

Model Name	Reasoning	Input/Output Cost (1M tokens)	RQ2 Acc.
`o3-mini-high`	✓	$1.1 / $4.4	**64.12**
`o1`	✓	$15 / $60	64.12
`o3-mini`	✓	$1.1 / $4.4	62.06
`gpt-4.5-preview`		$75 / $150	59.71
`o1-mini-2024-09-12`	✓	$1.1 / $4.4	59.64
`gemini-2.0-flash-001`		$0.1 / $0.4	55.59
`gpt-4o-2024-11-20`		$2.5 / $10	52.06
`gpt-4o-mini`		$0.15 / $0.6	50.59
`gpt-4o-mini-2024-07-18`		$0.15 / $0.6	50.29

Findings:

- Non-reasoning (i.e.: cheaper) models are *marginally* better than a coinflip at predicting the correct AI class
- Similar accuracy for both CUDA/OMP codes
- Still room for improvement with o3-mini-high achieving highest accuracy of 64%

Real code examples don't improve accuracy by much (RQ3)

Instructions + 2 real code examples

RQ3 Prompting Template (see paper for full prompt)

[omitted context-setting beginning of prompt]

Provide **only one word** as your response, chosen from the set: `['Compute', 'Bandwidth']`.

Examples:

Example 1:

hand-tuned CB CUDA/OMP example program

Response: **Compute**

Example 2:

hand-tuned BB CUDA/OMP example program

Response: **Bandwidth**

Hardware Roofline Specs

Now, analyze the following source codes for the requested kernel of the specified hardware.

Classify the [language] kernel called [kernel name] as **Bandwidth** or **Compute** bound. The system it will execute on is a [GPU model] with:

- peak single-precision performance of [X] GFLOP/s
- peak double-precision performance of [X] GFLOP/s
- peak integer performance of [X] GINTOP/s
- max bandwidth of [X] GB/s

Execution Specs

The block and grid sizes of the invoked kernel are (X,Y,Z) and (X,Y,Z), respectively. The executable running this kernel is launched with the following command-line arguments: `[arg1 arg2 arg3]`.

Below is the source code of the requested [language] kernel:

[concatenated source code files]

Model Name	Reasoning	RQ2 Acc.	RQ3 Acc.
o3-mini-high	✓	**64.12**	**63.53** ↓
o1	✓	64.12	61.47 ↓
o3-mini	✓	62.06	62.94 ↑
gpt-4.5-preview		59.71	60.88 ↑
o1-mini-2024-09-12	✓	59.64	56.47 ↓
gemini-2.0-flash-001		55.59	53.82 ↓
gpt-4o-2024-11-20		52.06	53.24 ↑
gpt-4o-mini		50.59	52.35 ↑
gpt-4o-mini-2024-07-18		50.29	52.06 ↑

Findings:

- Similar results to RQ2, suffers from higher query costs due to increased prompt size
- Non-reasoning models slightly improve accuracy (by 1-2%) when given real code examples
- Accuracy was similar for both CUDA/OMP codes

We need more data to fine-tune LLMs to predict AI (RQ4)

Approach:

- Fine-tuned gpt-4o-mini using an 80/20 train/test split of our 340-sample dataset (272/68 split)
- Used prompt template from RQ3 for training/testing
- Trained for 2 epochs (~$400 USD to train)
- Queried trained model 3x on each test sample

Fine-tuned LLM (1 epoch)

		Predicted Class	
		CB	BB
True Class	CB	24.51 %	25.49 %
	BB	20.10 %	29.90 %

54% total accuracy

Fine-tuned LLM (2 epoch)

		Predicted Class	
		CB	BB
True Class	CB	50 %	0.00 %
	BB	50 %	0.00 %

50% total accuracy

Findings:

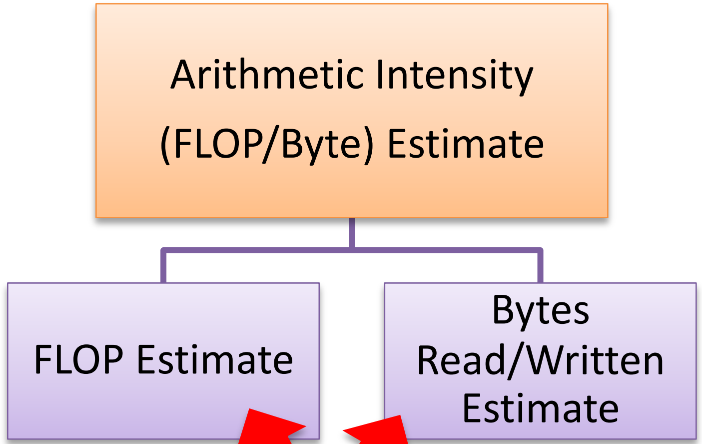
- Fine-tuning causes model responses to be constant
- No response variation across the 3 repeated queries
- Not enough data to thoroughly train model

Main Conclusions + Takeaways

- SoTA LLMs *do* understand the Roofline Model for GPU performance analysis
- SoTA LLMs *can* predict parallel code performance – when limited to classifying Arithmetic Intensity (AI) of CUDA and OpenMP programs
- Reasoning-equipped LLMs (e.g.: o3-mini-high) offer significantly better classification accuracy when compared to non-reasoning LLMs
- Reasoning-equipped LLMs don’t need real code examples in their prompts to help them provide better classifications (can save money on input tokens)
- Fine-tuning an LLM for better AI classification accuracy is going to need more data and money

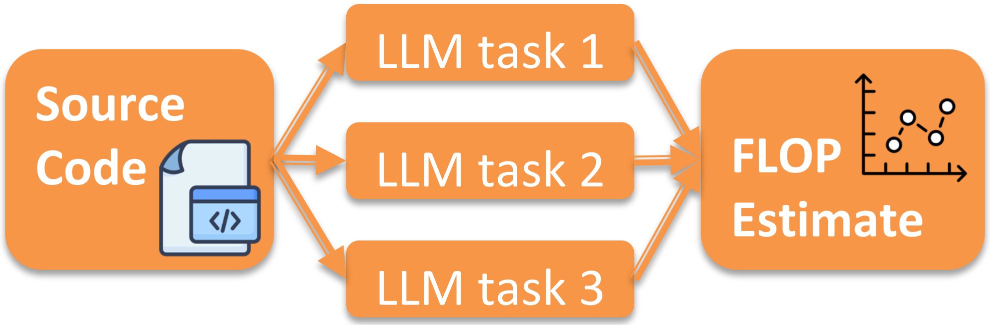
Major Shortcomings + Future Work

- Small dataset size
- Scraped source codes include all files
- Linear/single-query approach



What if the LLMs could estimate these values for us?

We currently have *some* success in applying Question Decomposition to estimate FLOPs



Target Name	Empirical FLOP Count	LLM-Estimated FLOP Count	% Diff
resize-cuda	16779307	16777216	0.012 %
zerocopy-cuda	1050389	1048576	0.17 %
iso2dfd-cuda	54419825	53196468	2.24 %
nlll-cuda	6006	6273	4.44 %
backprop-cuda	3080240	3080192	0.001 %

🙋‍♀️ Questions 🙋

Slides + Paper
+ Poster
Available Here

Thank You 😊