Scope is all you need:

Transforming LLMs for HPC Code

Contributors and Sponsors

Tal Kadosh
Ben-Gurion University, IAEC, Israel

Niranjan Hasabnis
Intel Labs, United States

Vy A. Vo
Intel Labs, United States

Nadav Schneider
Ben-Gurion University, IAEC, Israel

Neva Krien
Independent Researcher, Israel

Abdul Wasay
Intel Labs, United States

Nesreen Ahmed
Intel Labs, United States

Ted Willke
Intel Labs, United States

Guy Tamir
Intel, United States

Yuval Pinter
Ben-Gurion University, Israel

Timothy Mattson
Intel Labs, United States

Gal Oren
Technion, NRCN, Israel
galoren@cs.technion.ac.il

intel labs

Ben-Gurion University of the Negev

oneAPI

Centers of Excellence

Hewlett Packard Enterprise

CRAY

Our 2023 papers in this topic (so far)

- Kadosh, Tal, Niranjan Hasabnis, Vy A. Vo, Nadav Schneider, Neva Krien, Abdul Wasay, Nesreen Ahmed, Ted Willke, Guy Tamir, Yuval Pinter, Timothy Mattson, and Gal Oren . "**Scope is all you need: Transforming LLMs for HPC Code.**" *arXiv preprint arXiv:2308.09440* (2023).
- Kadosh, Tal, Niranjan Hasabnis, Timothy Mattson, Yuval Pinter, and Gal Oren. "**Quantifying OpenMP: Statistical Insights into Usage and Adoption [HPCorpus].**" *arXiv preprint arXiv:2308.08002* (2023). HPEC’23
- Schneider, Nadav, Tal Kadosh, Niranjan Hasabnis, Timothy Mattson, Yuval Pinter, and Gal Oren. "**MPI-rical: Data-Driven MPI Distributed Parallelism Assistance with Transformers.**" *arXiv preprint arXiv:2305.09438* (2023). AI4DEV @ SC’23
- Kadosh, Tal, Nadav Schneider, Niranjan Hasabnis, Timothy Mattson, Yuval Pinter, and Gal Oren. "**Advising OpenMP Parallelization via a Graph-Based Approach with Transformers.**" *arXiv preprint arXiv:2305.11999* (2023). IWOMP’23
- Harel, Re'em, Yuval Pinter, and Gal Oren. "**Learning to parallelize in a shared-memory environment with transformers.**" *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*. 2023. PPoPP’23
  - Re'em Harel, Tal Kadosh, Niranjan Hasabnis, Timothy Mattson, Yuval Pinter, and Gal Oren. “**PragFormer: Data-driven Parallel Source Code Classification with Transformers**”, 29 August 2023, *preprint* (V1) Research Square [https://doi.org/10.21203/rs.3.rs-3254961/v1]

Scope is all you need:

Intro and objectives - LLMs for HPC:	Gal Oren
Novel HPC code database - HPCorpus:	Niranjan Hasabnis
Compiler-oriented tokenizer - Tokompiler:	Vy A. Vo
Downstream Task #1 - OpenMP:	Tal Kadosh
Downstream Task #2 - MPI:	Nadav Schneider

Scope is all you need:

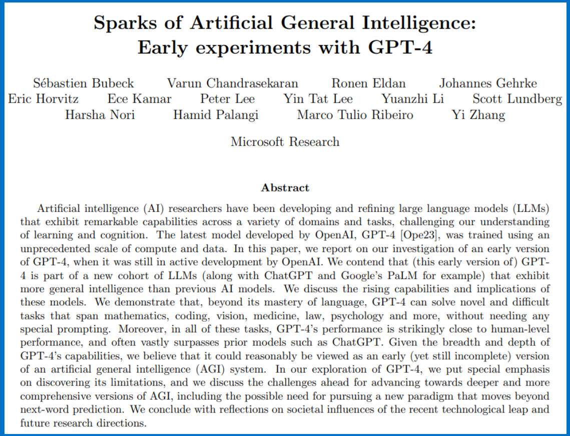
Intro and objectives - LLMs for HPC:	Gal Oren
Novel HPC code database - HPCorpus:	Niranjan Hasabnis
Compiler-oriented tokenizer - Tokompiler:	Vy A. Vo
Downstream Task #1 - OpenMP:	Tal Kadosh
Downstream Task #2 - MPI:	Nadav Schneider

Ever-increasing scope

- Infinite data
- Extend data representations
- (Really) large models
- Huge compute power
- Increasing tasks
- Fusion of knowledge
- …

Source: https://www.softwebsolutions.com/large-language-model-development.html

General Intelligence



Sparks of Artificial General Intelligence: Early experiments with GPT-4

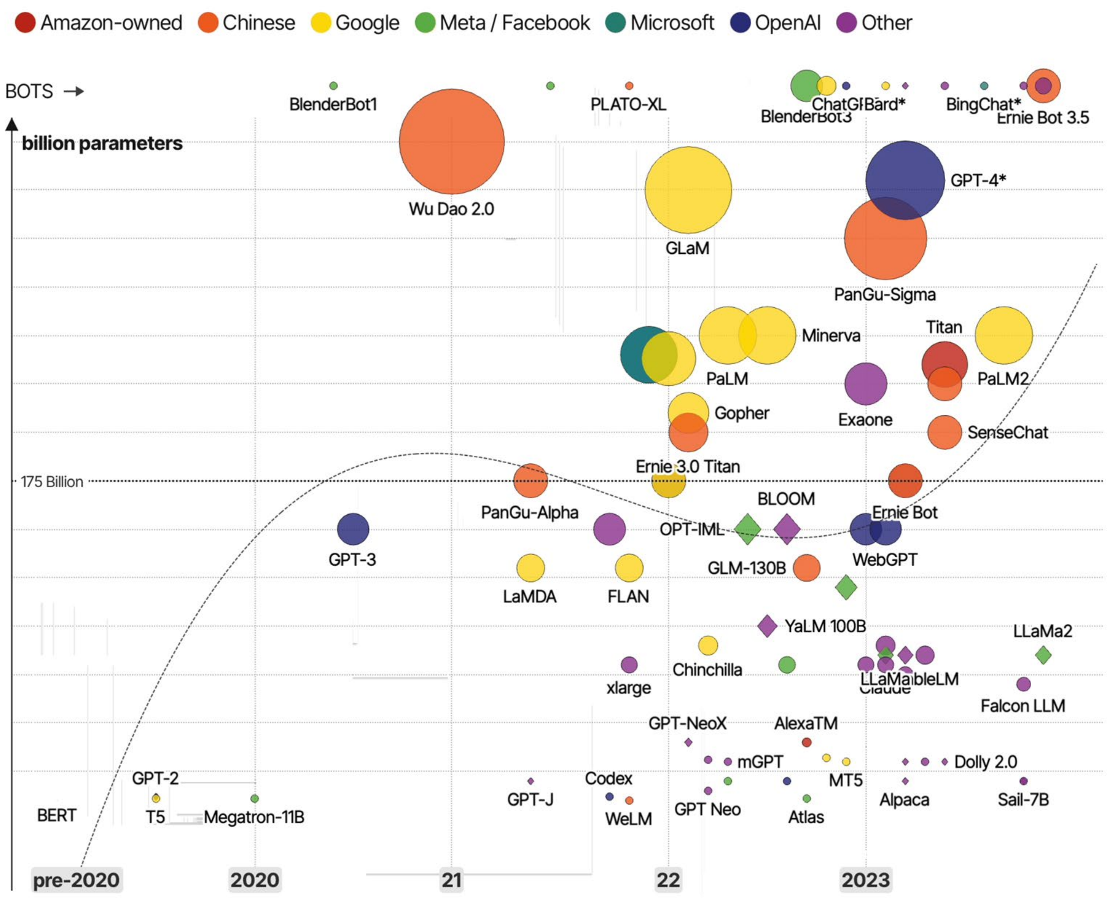
Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, Yi Zhang

Microsoft Research

Abstract

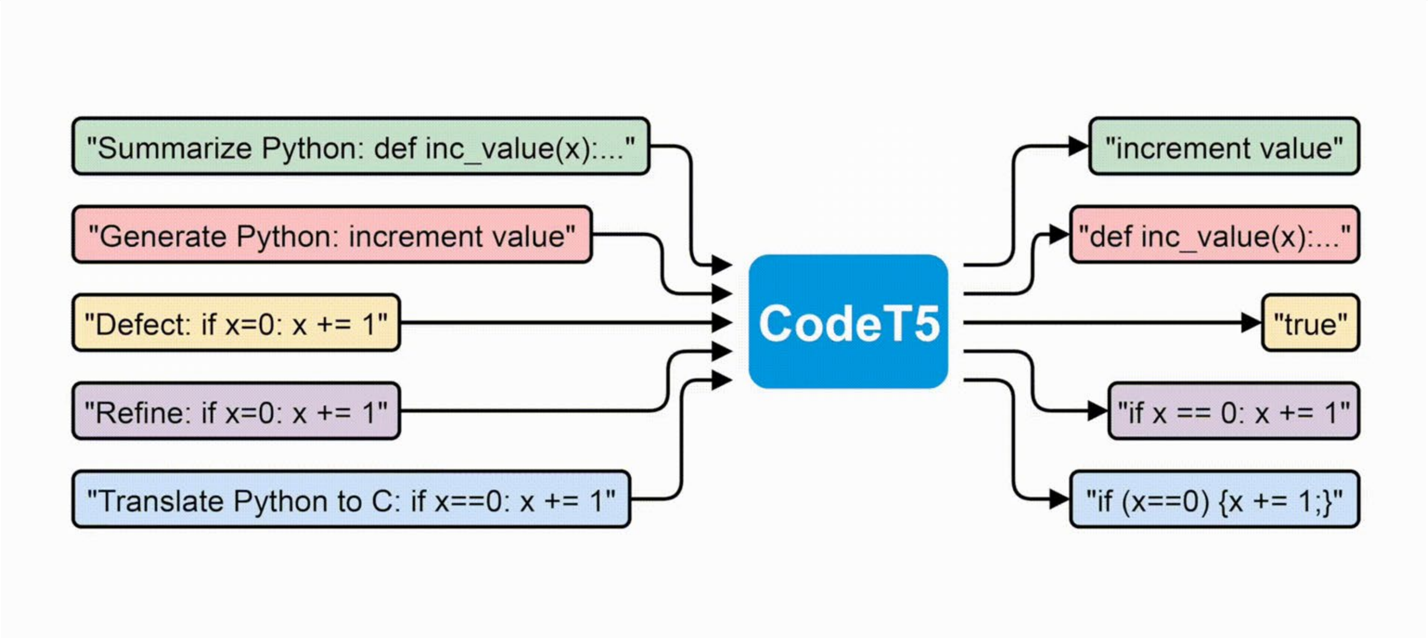
Artificial intelligence (AI) researchers have been developing and refining large language models (LLMs) that exhibit remarkable capabilities across a variety of domains and tasks, challenging our understanding of learning and cognition. The latest model developed by OpenAI, GPT-4 [Ope23], was trained using an unprecedented scale of compute and data. In this paper, we report on our investigation of an early version of GPT-4, when it was still in active development by OpenAI. We contend that (this early version of) GPT-4 is part of a new cohort of LLMs (along with ChatGPT and Google's PaLM for example) that exhibit more general intelligence than previous AI models. We discuss the rising capabilities and implications of these models. We demonstrate that, beyond its mastery of language, GPT-4 can solve novel and difficult tasks that span mathematics, coding, vision, medicine, law, psychology and more, without needing any special prompting. Moreover, in all of these tasks, GPT-4's performance is strikingly close to human-level performance, and often vastly surpasses prior models such as ChatGPT. Given the breadth and depth of GPT-4's capabilities, we believe that it could reasonably be viewed as an early (yet still incomplete) version of an artificial general intelligence (AGI) system. In our exploration of GPT-4, we put special emphasis on discovering its limitations, and we discuss the challenges ahead for advancing towards deeper and more comprehensive versions of AGI, including the possible need for pursuing a new paradigm that moves beyond next-word prediction. We conclude with reflections on societal influences of the recent technological leap and future research directions.

Source: https://arxiv.org/abs/2303.12712



Source: https://informationisbeautiful.net/visualizations/the-rise-of-generative-ai-large-language-models-llms-like-chatgpt/

Code Intelligence

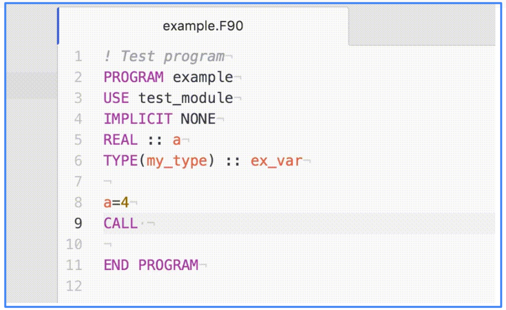


- Code Generation
- Code Debugging
- Code Optimization
- Code Explanation
- Algorithm Design
- Data Manipulation
- Web Development
- DB Interaction
- Code Review
- API Integration
- Code Documentation
- Code Translation
- Code Testing
- Security Analysis

Source: https://github.com/salesforce/CodeT5

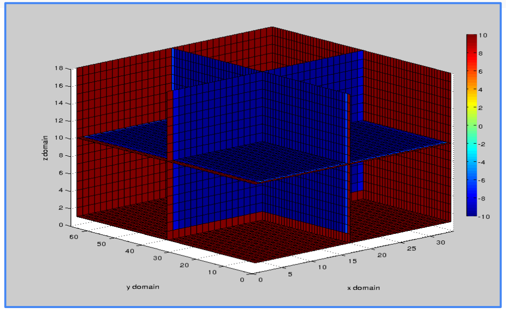
HPC Code - Same objectives?

HPC Langs



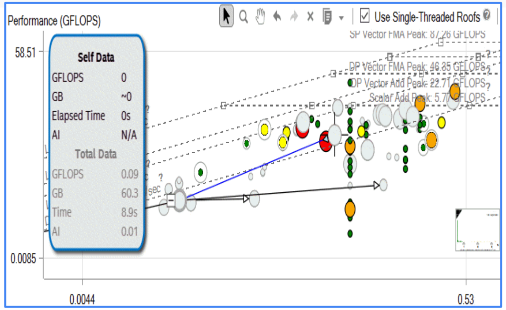
Source: https://github.com/hansec/autocomplete-fortran

Scientific Comp



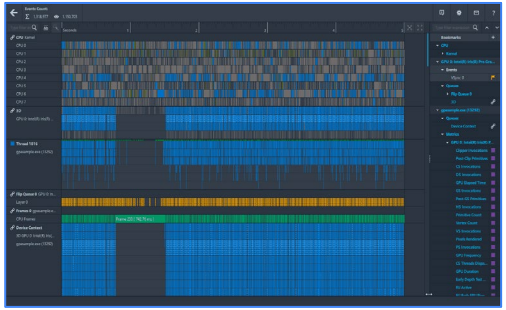
Source: https://dournac.org/info/parallel_heat3d

Performance



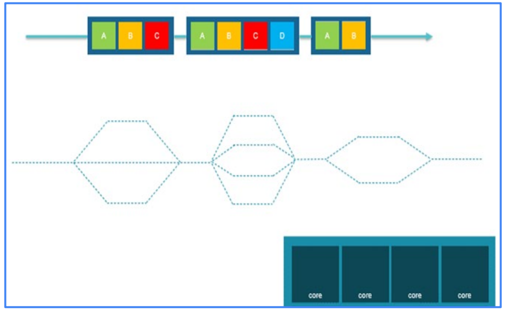
Source:.intel.com/content/www/../roofline-with-callstacksl

Dependencies



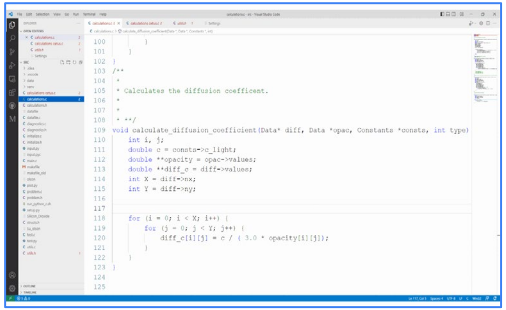
Source:intel.com/content/www../graphics-performance...html

Hardware Arch



Source: http://ferestrepoca.github.io/paradigmas-de-programacion

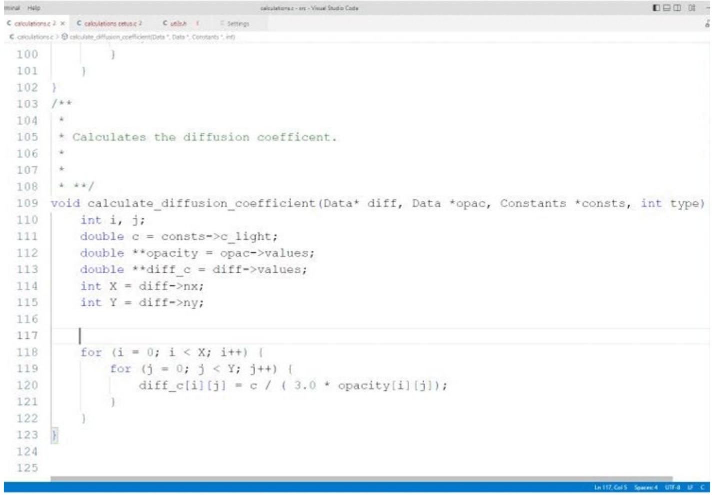
Parallelization



Source: https://www.cs.technion.ac.il/view-event.php?evid=10334

HPC Code Intelligence

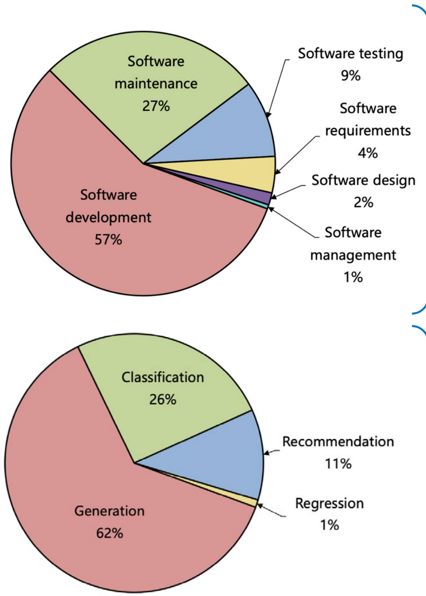
- Speed up code development
- Generate optimized parallel code from high-level specifications
- Improve code readability and maintainability as legacy code
- Facilitate cross-platform compatibility through code translation
- Optimize code for parallel execution and resource utilization
- Identify performance bottlenecks and parallelization issues
- Enable code search for existing optimized implementations
- Facilitate correct API usage
- Analyze code dependencies
- Enforce performance-oriented coding standards



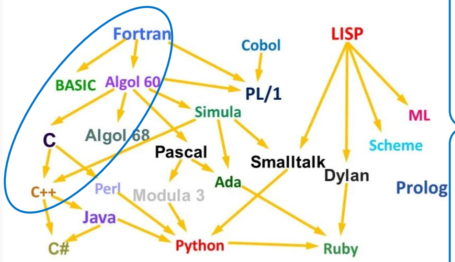
Suggest shared-memory parallelization on-the-fly based on LLM

What is our baseline? (1/3)

- Can non-HPC languages be the baseline for HPC tasks?
- Are there any relevant large corpora of high-performance code?
- What are the exact downstream tasks for HPC, how to correlate those with common models?
- Is mixing of languages and tasks is even good for HPC LLMs?



Hou, Xinyi, et al. "Large Language Models for Software Engineering: A Systematic Literature Review." arXiv preprint arXiv:2308.10620 (2023).



Source: https://medium.com/@anaharris/human-languages-vs-programming-languages-c89410f13252

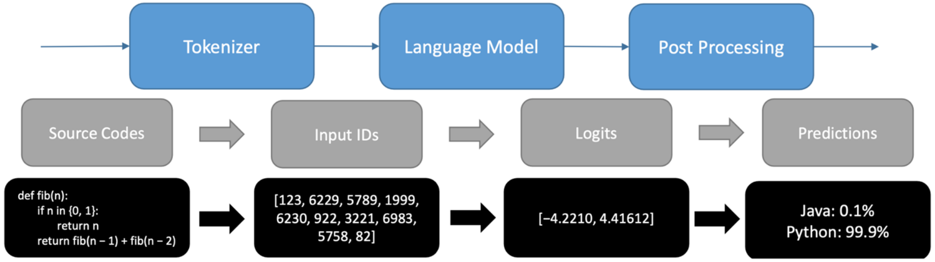
Framework	Year	Task(s)	Baseline(s)	Supported Language(s)	Open Sourced
Refactory [137]	2019	Defect Detection	BLEU	Java	✗
CuBERT [138]	2020	Code Refinement, Defect Detection	BERT	Python	✓
CugLM [139]	2020	Code Completion	BERT	Java, TypeScript	✓
Intellicode [140]	2020	Code Generation, Code Completion	GPT-2	Python, C#, JavaScript, and TypeScrip	✗
Great [141]	2020	Defect Detection	Vanilla Transformers	Python	✓
TreeGEN [51]	2020	Code Generation	Vanilla Transformers	Python	✓
C-BERT [127]	2020	Defect Detection	BERT	C	✗
TransCoder [142]	2020	Code Translation	Vanilla Transformers	C++, Java, and Python	✗
GraphCodeBERT [143]	2020	Code Summarization, Code Refinement	BERT	Java	✗
Codex [35]	2021	Code Generation, Code Completion, Code Summarization, Benchmark	GPT-3	JavaScript, Go, Perl, and 6 more	✗
Copilot [144]	2021	Code Generation, Code Completion	Codex	Java, PHP, Python, and 5 more	✗
CodeT5 [145]	2021	Code Summarization, Code Generation, Code Translation, Code Refinement, Defect Detection, Clone Detection	T5	Python, Java	✓
Tfix [146]	2021	Code Refinement, Defect Detection	T5	JavaScript	✓
CodeRL [147]	2021	Code Summarization, Code Generation, Code Translation, Code Refinement, Defect Detection, Clone Detection	T5	Java	✓
TreeBERT [148]	2021	Code Summarization	Vanilla Transformers	Python, Java	✓
BUGLAB [149]	2021	Code Refinement, Defect Detection	GREAT	Python	✓
TBCC [150]	2021	Clone Detection	Vanilla Transformers	C, Java	✓
APPS [36]	2021	Benchmark	N/A	Python	✓
CodeXGLUE [34]	2021	Benchmark	N/A	Python	✓
CoTexT [151]	2021	Code Summarization, Code Generation, Code Refinement, Defect detection	T5	Python, Java, Javascript, PHP, Ruby, Go	✓
SynCoBERT [152]	2021	Code Translation, Defect Detection, Clone Detection	BERT	Ruby, Javascript, Go, Python, Java, PHP	✗
TravTrans [153]	2021	Code Completion	Vanilla Transformers	Python	✗
CCAG [154]	2021	Code Completion	Vanilla Transformers	JavaScript, Python	✗
DeepDebug [155]	2021	Defect Detection	Reformer	Java	✓
Recoder [93]	2021	Defect Detection	TreeGen	Java	✓
PLBART [156]	2021	Code Summarization, Code Generation, Code Translation, Code Refinement, Clone Detection, Detect Detection	BART	Java, Python	✗
CODEGEN [157]	2022	Code Generation	GPT-NEO & GPT-J	Python	✓
GPT-2 for APR [158]	2022	Code Refinement	GPT-2	JavaScript	✓
CERT [39]	2022	Code Generation	CODEGEN	Python	✓
PyCoder [87]	2022	Code Generation	GPT-2	Python	✓
AlphaCode [38]	2022	Code Generation	GPT	Java	✗
InCoder [40]	2022	Code Generation, Code Completion, Code Summarization	GPT-3	Java, JavaScript, Python	✓
RewardRepair [159]	2022	Code Refinement, Defect Detection	T5	Java	✓
CodeParrot [37]	2022	Code Generation	GPT-2	Python	✓
AlphaRepair [160]	2022	Code Refinement, Defect Detection	CodeBERT	Java	✓
CodeReviewer [128]	2022	Code Summarization, Code Refinement, Defect Detection	CodeT5	Java	✓
TransRepair [161]	2022	Code Refinement, Defect Detection	BLEU	Java	✗
NatGen [162]	2022	Code Generation, Code Translation, Code Refinement	CodeT5	Java, Python, Go, JavaScript, Ruby, PHP	✓
DualSC [163]	2022	Code Generation, Code Summarization	T5	Shellcode	✓
VulRepair [164]	2022	Code Refinement, Defect Detection	T5	C, C++	✓
CoditT5 [165]	2022	Code Summarization, Defect Detection	CodeT5	Java, Python, Ruby, PHP, Go, JavaScript	✓
C4 [166]	2022	Clone Detection	CodeBERT	C++, C#, Java, Python	✓
SPT-Code [167]	2022	Code Summarization, Code Completion, Code Refinement, Code Translation	CodeBERT & GraphCode-BERT	Python, Java, JavaScript, PHP, Go	✓
ExploitGen [168]	2023	Code Generation	CodeBERT	Python, Assembly	✓
Santacoder [169]	2023	Code Summarization, Code Generation	GPT-2	Python, Java, and Javascript	✓
xCodeEval [42]	2023	Benchmark	N/A	Python, Java, C++, PHP, and 8 more	✓
StarCoder [170]	2023	Code Generation, Code Completion, Code Summarization	BERT & SantaCoder	HTML, Python, Java, and 83 more	✓

Wong, Man-Fai, et al. "Natural Language Generation and Understanding of Big Code for AI-Assisted Programming: A Review." Entropy 25.6 (2023): 888.

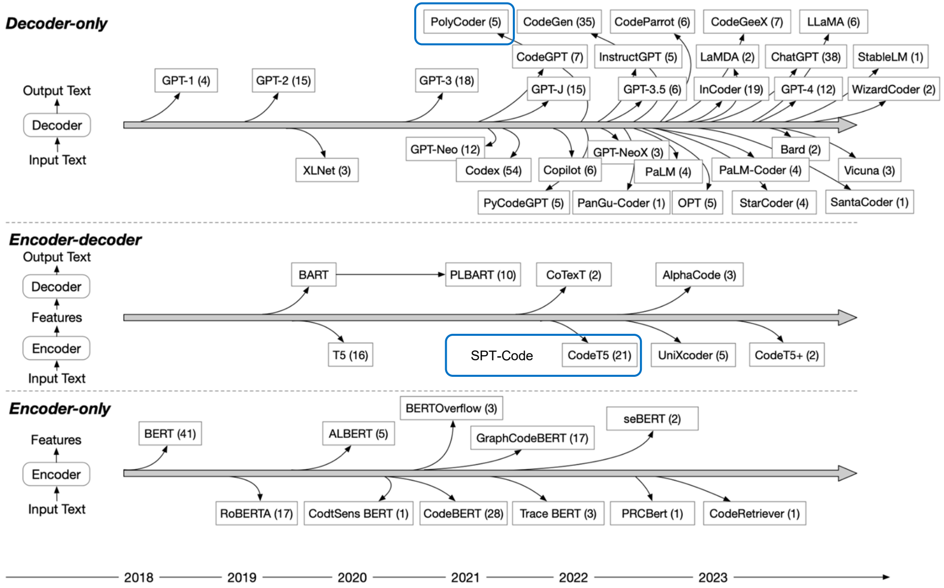
What is our baseline? (2/3)

- Are common tokenizers even relevant for compilation oriented code?
- What is the relevant model architecture for HPC tasks learning?
    - Encoder: Classification
    - Decoder: Generation
    - Encoder-Decoder: Both

      (understanding input sequences and generating output sequences, often with different lengths and structures)



Wong, Man-Fai, et al. "Natural Language Generation and Understanding of Big Code for AI-Assisted Programming: A Review." Entropy 25.6 (2023): 888.



Hou, Xinyi, et al. "Large Language Models for Software Engineering: A Systematic Literature Review." arXiv preprint arXiv:2308.10620 (2023).

What is our baseline? (3/3)

- Should we consider other representations rather just plain code or AST?
- What about hardware specs? Telemetry? Should it be part of the input?
- Can we reduce the size of the input?
    - Of the model?
        - Of the training?

Code sample:

```c
int main() {
    int r[2800 + 1];
    int i, k;
    int b, d;
    int c = 0;

    for (i = 0; i < 2800; i++) {
        r[i] = 2000;
    }

    for (k = 2800; k > 0; k -= 14) {
        d = 0;

        i = k;
      for (;;) {
        d += r[i] * 10000;
        b = 2 * i - 1;

        r[i] = d % b;
        d /= b;
        i--;
        if (i == 0) break;
        d *= i;
      }
      printf("%.4d", c + d / 10000);
      c = d % 10000;
    }

    return 0;
}
```

Tokenized code:

['int', 'main', '()', '{', 'int', 'r', '[', '28', '00', '+', '1', '];', 'int', 'i', ',', 'k', ';', 'int', 'b', ',', 'd', ';', 'int', 'c', '=', '0', ';', 'for', '(', 'i', '=', '0', ';', 'i', '<', '28', '00', ';', 'i', '++)', '{', 'r', '[', 'i', ']', '=', '2000', ';', '}', 'for', '(', 'k', '=', '28', '00', ';', 'k', '>', '0', ';', 'k', '-=', '14', ')', '{', 'd', '=', '0', ';', 'i', '=', 'k', ';', 'for', '(', ';;', ')', '{', 'd', '+=', 'r', '[', 'i', ']', '*', '10000', ';', 'b', '=', '2', '*', 'i', '-', '1', ';', 'r', '[', 'i', ']', '=', 'd', '%', 'b', ';', 'd', '/', '=', 'b', ';', 'i', '--', ';', 'if', '(', 'i', '==', '0', ')', 'break', ';', 'd', '*', '=', 'i', ';', '}', 'printf', '("', '%.', '4', 'd', '",', 'c', '+', 'd', '/', '10000', ');', 'c', '=', 'd', '%', '10000', ';', '}', 'return', '0', ';', '}']

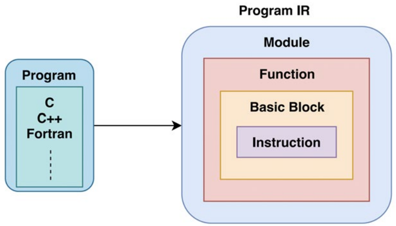
Tokenized AST:

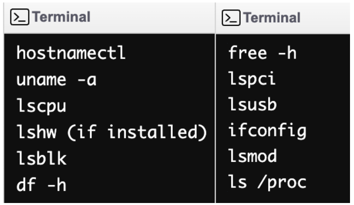
['translation_unit', '(', 'function_definition', '(', 'int', 'function_declarator', '(', 'main', 'parameter_list', '(', '(', ')', ')', ')', 'compound_statement', '(', '{', 'declaration', '(', 'int', 'array_declarator', '(', 'r', '[', 'binary_expression', '(', '2800', '+', '1', ')', ']', ')', ';', ')', 'declaration', '(', 'int', 'i', ',', 'k', ';', ')', 'declaration', '(', 'int', 'b', ',', 'd', ';', ')', 'declaration', '(', 'int', 'init_declarator', 'expression_statement', '(', 'assignment_expression', '(', 'subscript_expression', '(', 'r', '[', 'i', ']',
...

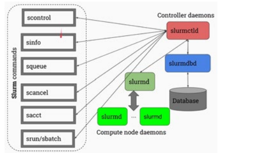
Tokenized DFG:

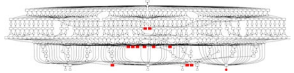
[('c', **24**, 'comesFrom', [], []), ('i', **30**, 'computedFrom', ['0'], [**32**]), ('0', **32**, 'comesFrom', [], []), ('i', **34**, 'comesFrom', ['i'], [**30**]), ('i', **38**, 'comesFrom', ['i'], [**30**]), ('r', **42**, 'computedFrom', ['2000'], [**47**]), ('i', **44**, 'computedFrom', ['2000'], [**47**]), ('2000', **47**, 'comesFrom', [], []), ('k', **52**, 'computedFrom', ['2800'], [**54**]), ('2800', **54**, 'comesFrom', [], []), ('k', **56**, 'comesFrom', ['k'], [**52**]), ('k', **60**, 'computedFrom', ['14'], [**62**]), ('14', **62**, 'comesFrom', [], []), ('d', **65**, 'computedFrom', ['0'], [**67**]), ('0', **67**, 'comesFrom', [], []),
...

IR ? loop-carried dependencies? hardware specs? Telemetry? …









SRIKANT, YN. "IR2Vec: A Flow Analysis based Scalable Infrastructure for Program Encodings." arXiv preprint arXiv:1909.06228 (2019).

Source: https://medium.com/pangeo/dask-jobqueue-d7754e42ca53
Source: educative.io/courses/learn-to-use-hpc-systems-and-..

“Scope is all you need”

(if you want transforming LLMs for HPC)

- The authors

Scope is all you need:

Intro and objectives - LLMs for HPC:	Gal Oren
Novel HPC code database - HPCorpus:	Niranjan Hasabnis
Compiler-oriented tokenizer - Tokompiler:	Vy A. Vo
Downstream Task #1 - OpenMP:	Tal Kadosh
Downstream Task #2 - MPI:	Nadav Schneider

HPC languages

- Frontier and Aurora Supercomputers mostly rely on C, C++, Fortran…
- Top applications for HPC rely on MPI for distribution and OpenMP for shared-memory parallelization and GPU offloading
- So why not collect large corpus of those for LLM training, fine-tuning and special downstream tasks?

Domain	Application	Programming Languages, Models, Portability Layers	Algorithmic / Motifs
Astrophysics	Cholla	C/C++, HIP	Finite volume, multi-D FFTs
Molecular Dynamics	NAMD	C++, HIP, OpenMP, Charm++	Particle mesh Ewald, MD
Materials Science	LSMS	Fortran, C/C++, HIP, OpenMP	Dense linear algebra
Biology	CoMet	C++, HIP, OpenMP	Dense linear algebra
Fluid Dynamics	GESTS	Fortran, OpenMP	FFTs, Fourier pseudo-spectral
Nuclear Physics	NUCCOR	Fortran, C, OpenMP, HIP	Sparse and dense linear algebra, Hartree-Fock
Plasma Physics	PIConGPU	C++, HIP, OpenMP, CUPLA & ALPAKA	Particle in cells
Subsurface Flow	LBPM	C++, HIP	Lattice Boltzmann

Source: W. Joubert, "Ready for the Frontier" presentation, ISC'23

Peak Performance
≧ 2 Exaflops DP

Intel GPU
Ponte Vecchio (PVC)

Intel Xeon Processor
Sapphire Rapids with High Bandwidth Memory

Platform
HPE Cray-Ex

Compute Node
2 Xeon SPR+HBM processors
6 Ponte Vecchio GPUs
Node Unified Memory Architecture
8 fabric endpoints

GPU Architecture
Intel XeHPC architecture
High Bandwidth Memory Stacks

Node Performance
>130 TF

System Size
>9,000 nodes

Aggregate System Memory
>10 PB aggregate System Memory

System Interconnect
HPE Slingshot 11
Dragonfly topology with adaptive routing

Network Switch
25.6 Tb/s per switch (64 200 Gb/s ports)
Links with 25 GB/s per direction

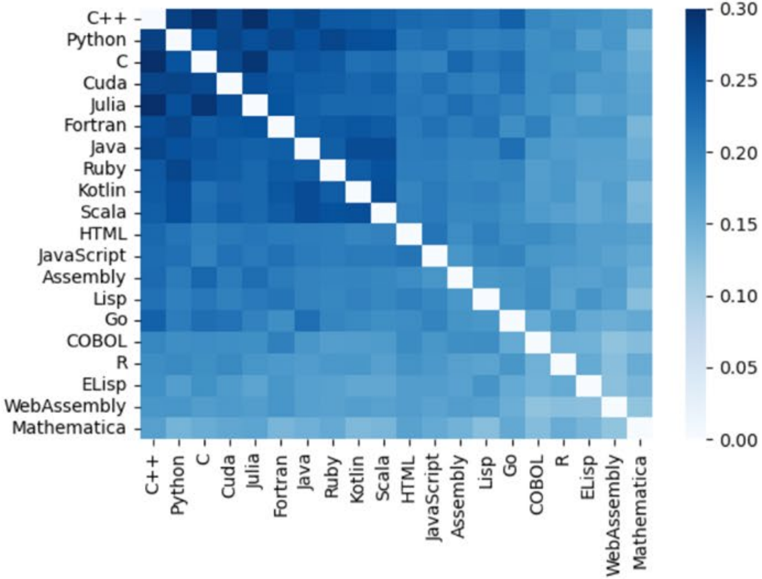
High-Performance Storage
220 PB
≧25 TB/s DAOS bandwidth

Software Environment
- C/C++
- Fortran
- SYCL/DPC++
- OpenMP offload
- Kokkos
- RAJA
- Intel Performance Tools

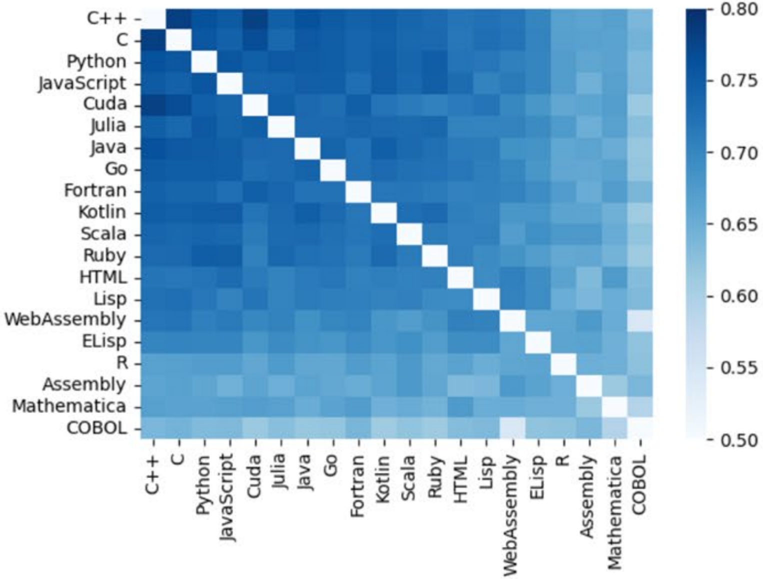
Software Environment
- C/C++
- Fortran
- SYCL/DPC++
- OpenMP offload
- Kokkos
- RAJA
- Intel Performance Tools

Source: ALCF SDL Workshop October 6, 2022, Colleen Bertoni and Scott Parker

Pairwise similarity between the representation of tokens of all code languages in CodeBERT



Non-pretrained setting



Pretrained setting

Source: Katzy, Jonathan, Maliheh Izadi, and Arie van Deursen. "On the Impact of Language Selection for Training and Evaluating Programming Language Models." arXiv preprint arXiv:2308.13354 (2023).

HPCorpus

- All of C, C++ and Fortran codes from 2012-mid 2023
- High Level:
  - ~300K repos
  - ~70 GB
  - ~9 M files
  - ~155 M functions


```sql
WITH selected_repos as (
SELECT f.id, f.repo_name as repo_name, f.ref as ref, f.path as path
FROM `bigquery-public-data.github_repos .files` as f
JOIN `bigquery-public-data.github_repos .licenses` as l on l.repo_name = f.repo_name
),
deduped_files as (
SELECT f.id, MIN(f.repo_name) as repo_name, MIN(f.ref) as ref, MIN(f.path) as path
FROM selected_repos as f
GROUP BY f.id
)
SELECT
f.repo_name, f.ref, f.path, c.copies, c.content,
FROM deduped_files as f
JOIN `bigquery-public-data.github_repos .contents` as c on f.id = c.id
WHERE
 NOT c.binary
 AND (f.path like '%.c' OR f.path like '%.cpp' OR f.path like '%.f' OR f.path like '%.f90' OR f.path like '%.f95')
```

	Repos	Size(GB)	Files (#)	Functions (#)
C	144,522	46.23	4,552,736	87,817,591
C++	150,481	26.16	4,735,196	68,233,984
Fortran	3,683	0.68	138,552	359,272

HPCorpus: statistical analysis

- Insights into parallel programming APIs and OpenMP
- Presented in our "Quantifying OpenMP" paper
- To be presented at IEEE High Performance Extreme Computing (HPEC) 2023

Quantifying OpenMP: Statistical Insights into Usage and Adoption

Tal Kadosh[1,2], Niranjan Hasabnis[3], Timothy Mattson[3], Yuval Pinter[1] and Gal Oren[4,5]

[1]Department of Computer Science, Ben-Gurion University, Israel
[2]Israel Atomic Energy Commission
[3]Intel Labs, United States
[4]Scientific Computing Center, Nuclear Research Center – Negev, Israel
[5]Department of Computer Science, Technion – Israel Institute of Technology, Israel
talkad@post.bgu.ac.il, niranjan.hasabnis@intel.com,
timothy.g.mattson@intel.com, pintery@bgu.ac.il, galoren@cs.technion.ac.il

***Abstract*—In high-performance computing (HPC), the demand for efficient parallel programming models has grown dramatically since the end of Dennard Scaling and the subsequent move to multi-core CPUs. OpenMP stands out as a popular choice due to its simplicity and portability, offering a directive-driven approach for shared-memory parallel programming. Despite its wide adoption, however, there is a lack of comprehensive data on the actual usage of OpenMP constructs, hindering unbiased insights into its popularity and evolution.**

This paper presents a statistical analysis of OpenMP usage and adoption trends based on a novel and extensive database, HPCorpus, compiled from GitHub repositories containing C, C++, and Fortran code. The results reveal that OpenMP is the dominant parallel programming model, accounting for 45% of all analyzed parallel APIs. Furthermore, it has demonstrated steady and continuous growth in popularity over the past decade. Analyzing specific OpenMP constructs, the study provides in-depth insights into their usage patterns and preferences across the three languages. Notably, we found that while OpenMP has a strong "common core" of constructs in common usage (while the rest of the API is less used), there are new adoption trends as well, such as `simd` and `target` directives for accelerated computing and `task` for irregular parallelism.

Overall, this study sheds light on OpenMP's significance in HPC applications and provides valuable data for researchers and practitioners. It showcases OpenMP's versatility, evolving adoption, and relevance in contemporary parallel programming, underlining its continued role in HPC applications and beyond. These statistical insights are essential for making informed decisions about parallelization strategies and provide a foundation for further advancements in parallel programming models and techniques.

HPCorpus, as well as the analysis scripts and raw results, are available at: https://github.com/Scientific-Computing-Lab-NRCN/HPCorpus

***Index Terms*—HPCorpus, BigQuery, GitHub, C, C++, Fortran, OpenMP, MPI, OpenCL, CUDA, TBB, Cilk, OpenACC, SYCL**

I. Introduction

With the end of Dennard Scaling [1], multicore CPUs sharing a cache-coherent address space are ubiquitous. To exploit the parallelism available from multicore systems, programmers use multithreaded programming models. Programming models that support multithreaded execution include pThreads [2] for low-level and OS-level operations, TBB [3] or Cilk [4] for task-level parallelism in C++, and OpenMP [5] for directive-driven parallelism.

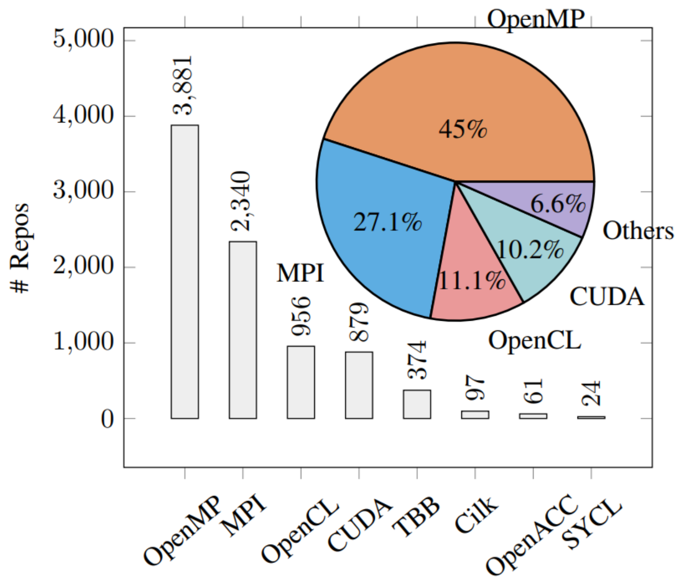
Despite the popularity of multithreaded models, little empirical data is available to assess the actual usage of the various programming constructs from these models. While anecdotal data and feedback from user-support teams at supercomputing centers exist [6], a large-scale analysis has yet to be conducted. In this paper, we perform a statistical analysis of repositories from GitHub to study the usage of parallel programming models. Our analysis reveals that OpenMP is the dominant programming model for writing multithreaded applications. We also go inside applications to gather usage data on specific OpenMP constructs. Finally, we consider the evolution of OpenMP and the adoption of newly included constructs as OpenMP Specifications are released.

II. OpenMP Fundamentals vs. Other Parallel Programming APIs

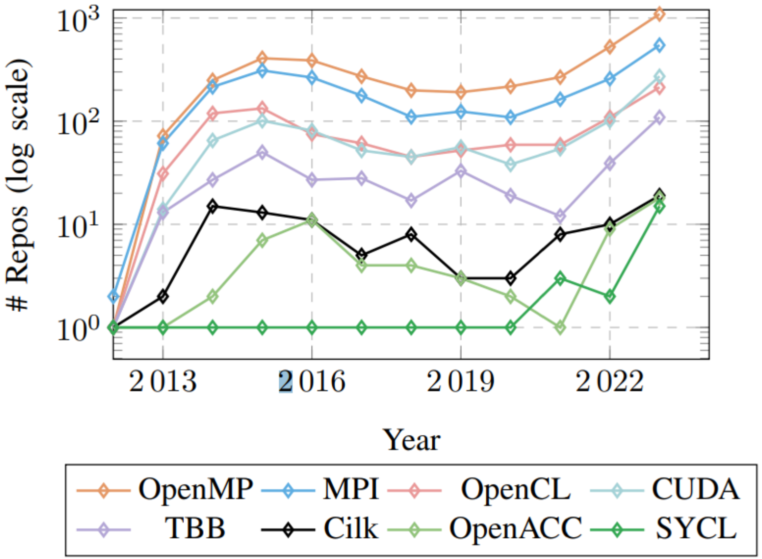
In this section, we provide a brief overview of OpenMP and other parallel programming APIs. OpenMP [7] defines a simple and portable approach to shared-memory parallel programming. OpenMP makes parallel programming more accessible by offering a directive-based approach, where directives inserted into the code guide the compiler as it generates parallel code. These directives provide high-level abstractions to specify parts of a code to execute in parallel. OpenMP uses the fork-join model of parallelism, where a single thread (the PRIMARY thread) on encountering a parallel directive forks a team of threads, each of which executes the code in a parallel region independently. Synchronization constructs, such as barriers, coordinate multithreaded execution, while shared variables facilitate data sharing among threads. OpenMP emphasizes portability with a standardized API that can be used across different platforms, hardware, and programming languages. These features make it easier to write parallel code that can be compiled and executed on systems supporting OpenMP.

https://arxiv.org/abs/2308.08002

HPCorpus: Parallel APIs

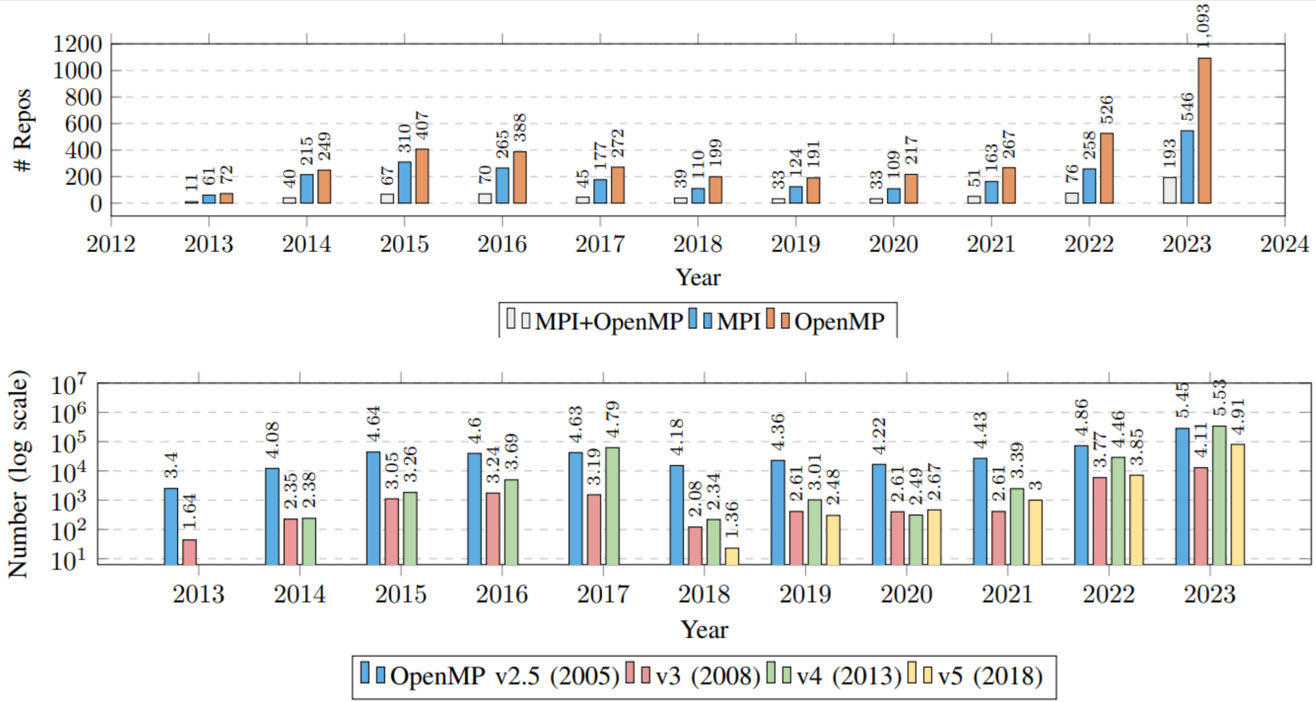


Parallel programming API usage in HPCorpus

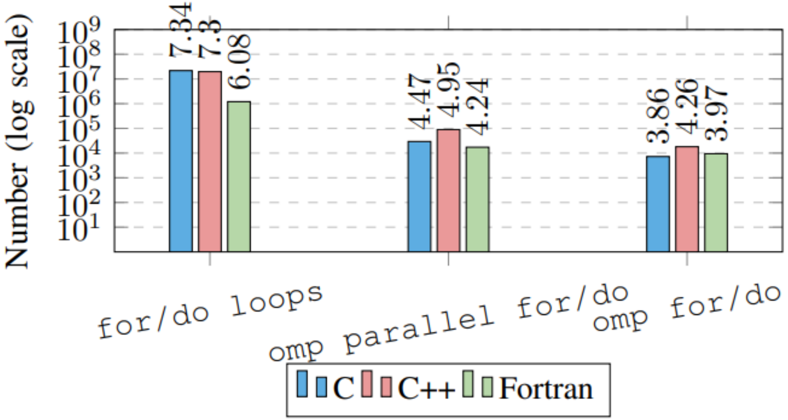


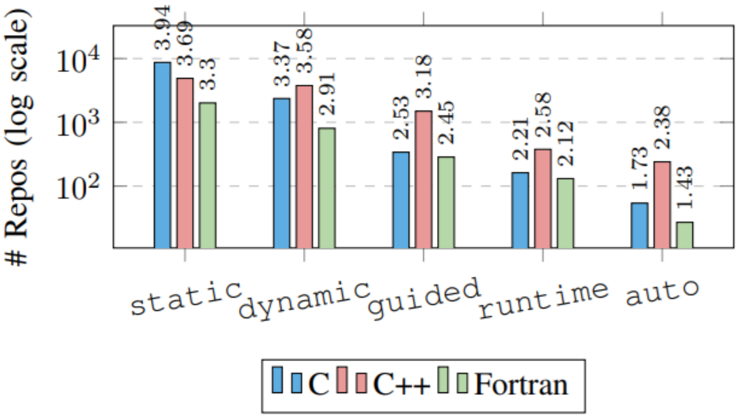
Usage trends in HPCorpus over the last decade

OpenMP and MPI usage breakdown



OpenMP loops and scheduling attributes



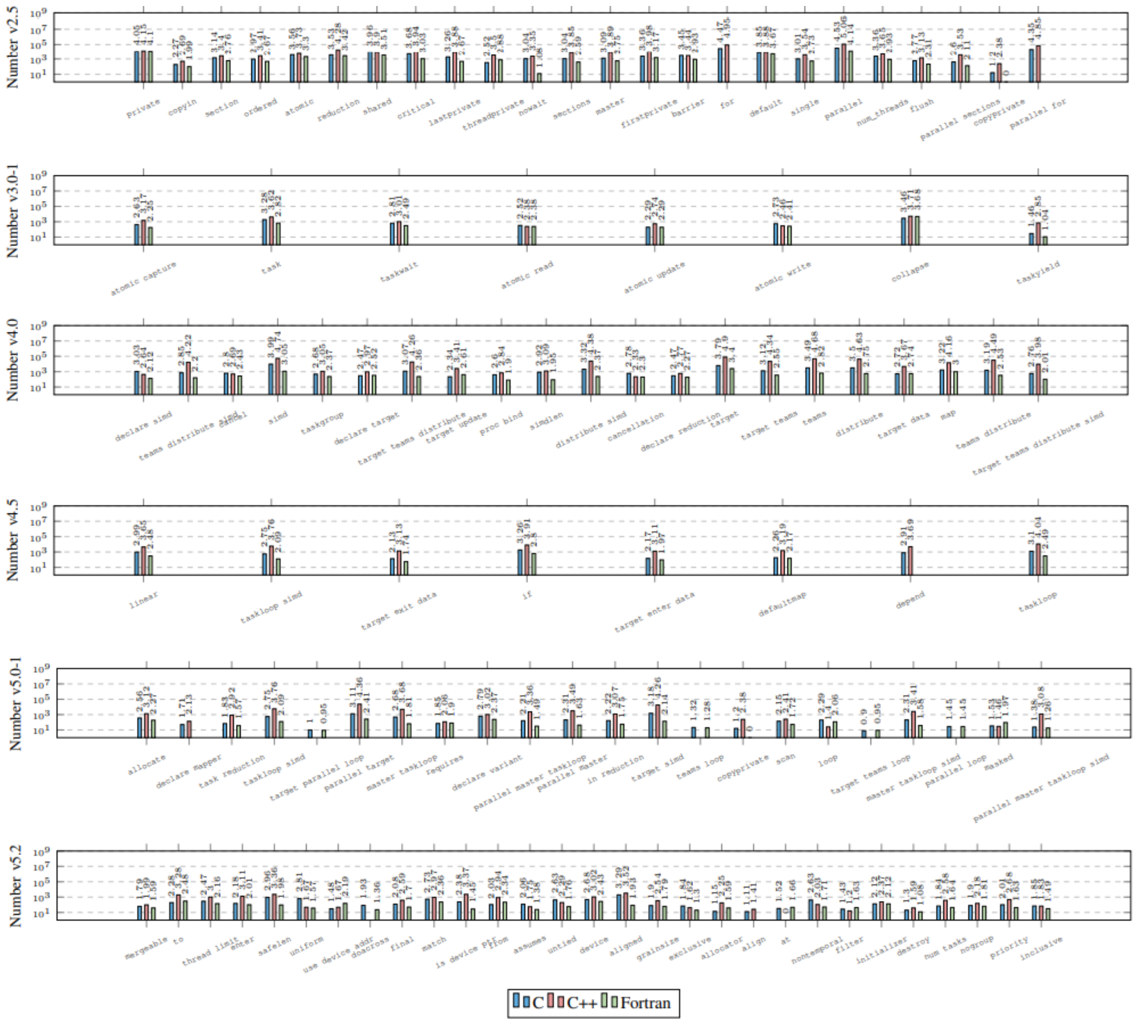


OpenMP meltdown

Each and every directive, clause etc. was carefully analysed for any future usage, including:

- SIMD
- teams distribute
- tasks
- Data-sharing attribute clauses

…

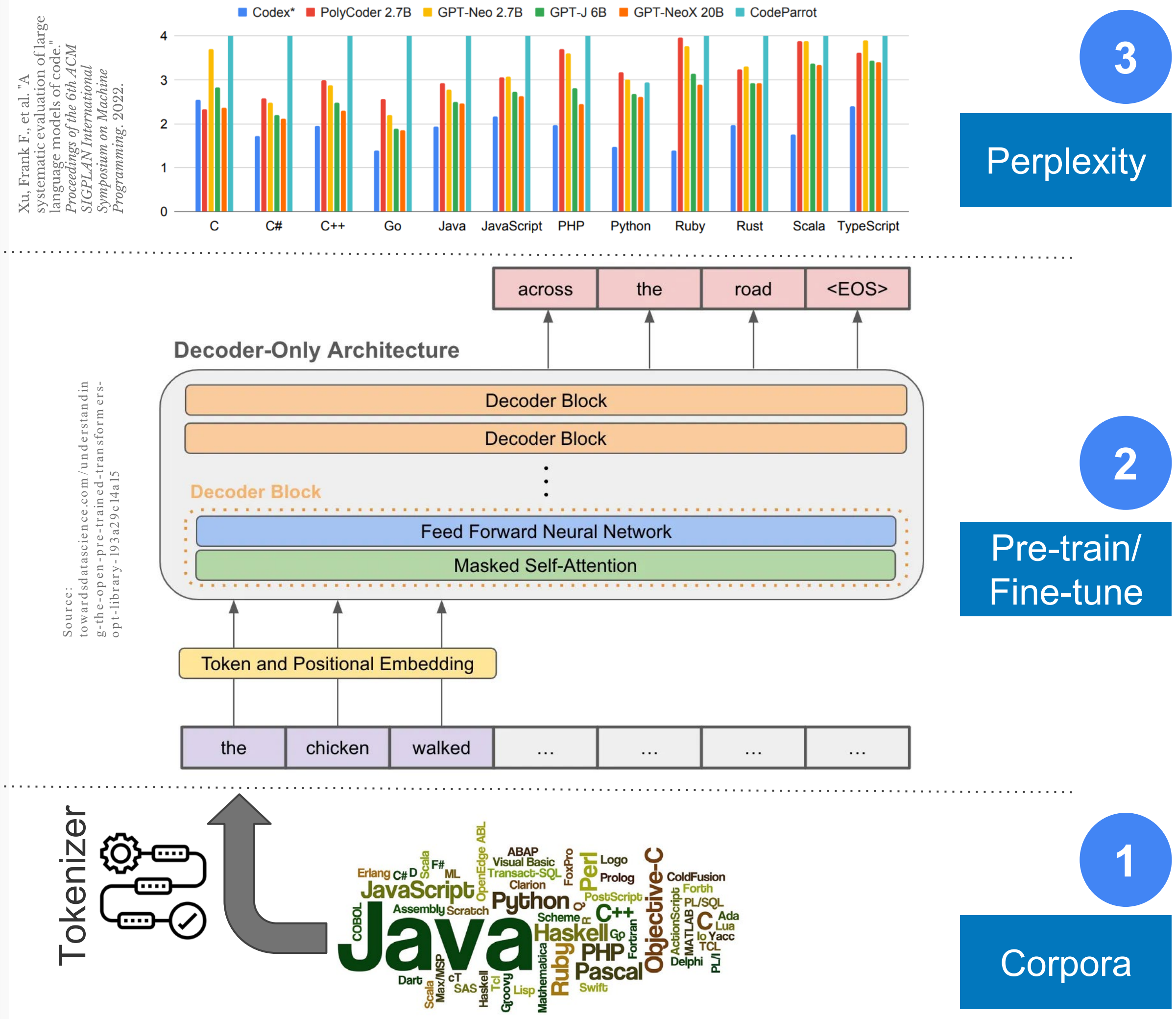


Scope is all you need:

Intro and objectives - LLMs for HPC:	Gal Oren
Novel HPC code database - HPCorpus:	Niranjan Hasabnis
Compiler-oriented tokenizer - Tokompiler:	Vy A. Vo
Downstream Task #1 - OpenMP:	Tal Kadosh
Downstream Task #2 - MPI:	Nadav Schneider

Pre-train or fine-tune?

- Why do we need an LLM trained on Java or Python (i.e., PolyCoder, StarCoder) for HPC-specific tasks?
- Can we use non-trivial or domain-specific tokenization for HPC tasks?
- Can domain-specificity improve the computational and therefore financial efficiency of pretraining?

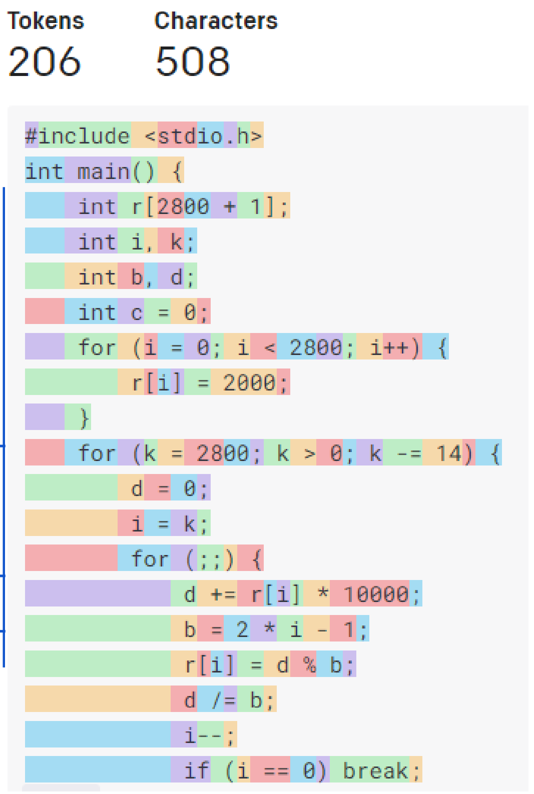


Current Common Tokenization

- Current tokenization strategies for NL likely do not address the needs of HPC tasks
- Can we do better if we will find the way to:
  - Remove NL
  - Remove local semantics
  - Keep structure
  - Add more representations?

GPT3 Tokenizer of the Pi C code: https://platform.openai.com/tokenizer

Codex Tokenizer of the Pi C code: https://platform.openai.com/tokenizer



Tokompiler

- **Generate Replaced Code:**
  Create a version of the original code with anonymized variable names, numbers, and strings
- **AST Parsing**:
  Parse the anonymized code using tree-sitter or any suitable parser to generate an AST
- **Recreate AST Changes**:
  Update the AST to reflect changes made during anonymization. Keep a dictionary of all changes done per file/function to facilitate restoring semantics back later
- **AST to Code-Tokenize**:
  Transform the updated AST back into code, eliminating any comments, new lines, and READMEs that may have been introduced during anonymization. This code-tokenized version will have a much smaller number of tokens
- **Token Splitting**:
  Split multi-part tokens (e.g., "var_1" to ["var", "1"]) to ensure that the model comprehends variable names as a combination of type and a unique identifier
- **Random Number Attachment**:
  For recurrent tokens (e.g., "var_1" or "num_2"), use statistics to attach random numbers from a predefined range (e.g., 1 to 1000) during each tokenization. The attached numbers are randomly chosen without any relation to the type or order of the replaced tokens or the file/function length

HPCorpus-Fortran

Tokompiler

Tokenized code	Tokenized AST
['function', 'func', '48', '(', 'arg', '128', ',', 'arg', '807', ')', 'result', '(', 'func', '180', ')', 'implicit', 'none', 'integer', ',', 'intent', '(', 'in', ')', '::', 'arg', '128', ',', 'arg', '807', 'real', '(', 'num', '5', ')', '::', 'func', '180', 'real', '(', 'num', '5', ')', '::', 'var', '377', ',', 'var', '84', ',', 'var', '967', 'integer', '::', 'var', '821', 'external', '::', 'func', '123', 'integer', '::', 'var', '63', 'var', '63', '=', 'num', '156', 'call', 'var', '719', '(', 'arg', '807', ')', 'do', 'var', '821', '=', 'num', '315', ',', 'arg', '128', 'var', '84', '=', 'func', '123', '(', ')', '*', 'num', '357', '-', 'num', '315', 'var', '967', '=', 'func', '123', '(', ')', '*', 'num', '357', '-', 'num', '315', 'if', '(', 'var', '84', '*', 'var', '84', '+', 'var', '967', '*', 'var', '967', '<', 'num', '315', ')', 'then', 'var', '63', '=', 'var', '63', '+', 'num', '315', 'end', 'if', 'var', '377', '=', 'num', '539', '*', 'func', '937', '(', 'var', '63', ')', '/', 'func', '937', '(', 'var', '821', ')', 'end', 'do', 'func', '180', '=', 'num', '539', '*', 'func', '937', '(', 'var', '63', ')', '/', 'func', '937', '(', 'arg', '128', ')', 'end', 'function']	'translation_unit', '(', 'declaration', '(', 'function', 'function_declarator', '(', 'func', '48', 'parameter_list', '(', '(', 'parameter_declaration', '(', 'arg', '128', ')', ',', 'parameter_declaration', '(', 'arg', '807', ')', ')', ')', ')', ')', 'declaration', '(', 'macro_type_specifier', '(', 'result', '(', 'type', 'descriptor', '(', 'func', '180', ')', ')', ')', 'implicit', ')', 'declaration', '(', 'none', 'integer', ',', 'function_declarator', '(', 'intent', 'parameter_list', '(', '(', 'parameter_declaration', '(', 'in', ')', ')', ')', ')', '(', ':', ':', 'arg', '128', ')', ',', '(', 'arg', '807', 'function_declarator', '(', 'real', 'parameter_list', '(', '(', 'parameter_declaration', '(', 'num', '5', ')', ')', ')', ')', ':', ':', 'func', '180', 'function_declarator', '(', 'real', 'parameter_list', '(', '(', 'parameter_declaration', '(', 'num', '5', ')', ')', ')', ')', ':', ':', ')', 'var', '377', ',', 'var', '84', ',', '(', 'var', '967', 'integer', '::', 'var', '821', 'external', ':', ':', 'func', '123', 'integer', ':', ':', 'var', '63', ')', 'init_declarator', '(', 'var', '63', '=', 'num', '156', ')', ')', 'declaration', '(', 'call', 'function_declarator', '(', 'var', '719', 'parameter_list', '(', '(', ...
687.72	1099.1

177 primitives + 1000 locals = 1177

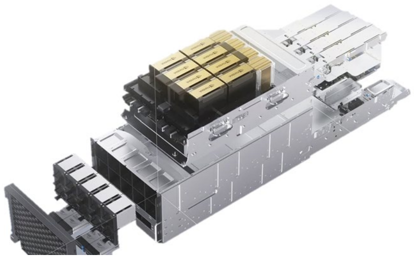
Pre-training on HPCorpus Fortran

- New given form of data (stripped of NL and only using structured blocks of code) help the models in improving the results
- The usage of Tokompiler increased performance by 230%
- Training speed improved by an average of 10% (as vocabulary and tokenized corpus size are smaller, 1177 against 50K).

HPCorpus Fortran Pre-training

	PolyCoder		SPT-Code	
	BPE	Tokompiler	NLTK	Tokompiler
Code	2.46	**1.60**	2.50	**1.08**
Code+AST	NA	NA	2.72	**1.08**
Model Size	162M	**88M**	238M	**201M**
Time to train (min)	435	**390**	186	**150**

4 x A40 48GB GPUs



8 x H100 80G GPUs

Pre-training on HPCorpus C/C++

- We pretrained an LLM with a standard size with both tokenizers
- On architectures that only vary in the size of the embedding, performance is better with Tokompiler
- Reducing the size of the Tokompiler model yields better performance

HPCorpus C/C++ Pre-training

	PolyCoder		
	BPE	Tokompiler	Tokompiler-small
Code	1.93	**1.71**	**1.65**
Code+AST	NA	NA	NA
Model Size	2.8B	**2.5B**	**638M**
Time to train (min)	8300	**8125**	**6386**

4 x A40 48GB GPUs

Scope is all you need:

Intro and objectives - LLMs for HPC:	Gal Oren
Novel HPC code database - HPCorpus:	Niranjan Hasabnis
Compiler-oriented tokenizer - Tokompiler:	Vy A. Vo
Downstream Task #1 - OpenMP:	Tal Kadosh
Downstream Task #2 - MPI:	Nadav Schneider

Our Vision

Does a language model can answer the following questions?

- Can we detect opportunity for parallelization?
- What should the variables scoping be?
- Can we completely generate a pragma given an input?

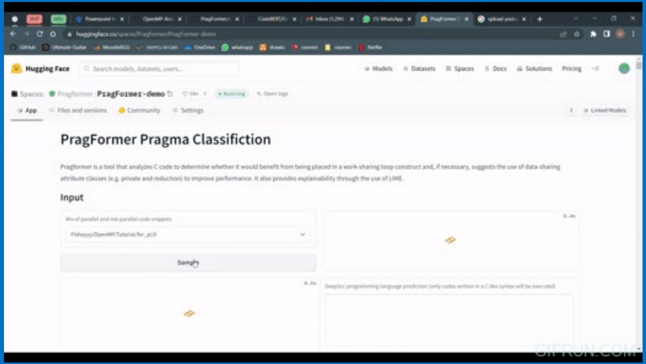
```
void dotProduct(int* A, int* B, int size) {
    int sum = 0;
     for (int i = 0; i < size; ++i) {
        sum += A[i] * B[i];}
    return sum;}
```

Large Language Model

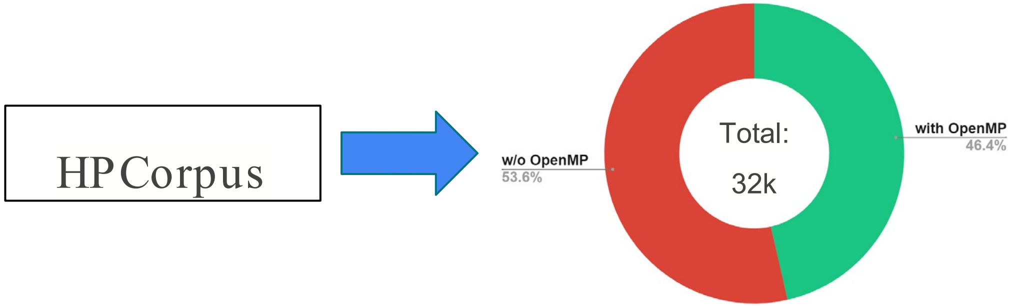
```
void dotProduct(int* A, int* B, int size) {
    int sum = 0;
    #pragma omp target data map(to: A[0:size], B[0:size]) map(from: sum){
        #pragma omp target teams distribute parallel for reduction(+:sum)
        for (int i = 0; i < size; ++i) {
            sum += A[i] * B[i];}}
    return sum;}
```

PragFormer

- Can we detect opportunity for parallelization?
- Is it possible to classify the need for OpenMP pragma?



PragFormer @ **Hugging Face**



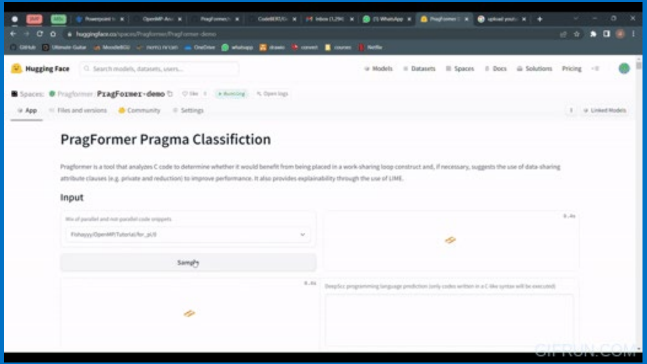

```
int print_dotProd(double *A, double *B) {

  for (int i = 0; i < size; ++i) {
     printf("%f * %f", A[i], B[i]);
  }

  #pragma omp reduction(+:sum)
  for (int i = 0; i < size; ++i) {
     sum += A[i] * B[i];
  }
}
```

Negative Example

Positive Example

PragFormer

- Can we detect opportunity for parallelization?
- Is it possible to classify the need for OpenMP pragma?



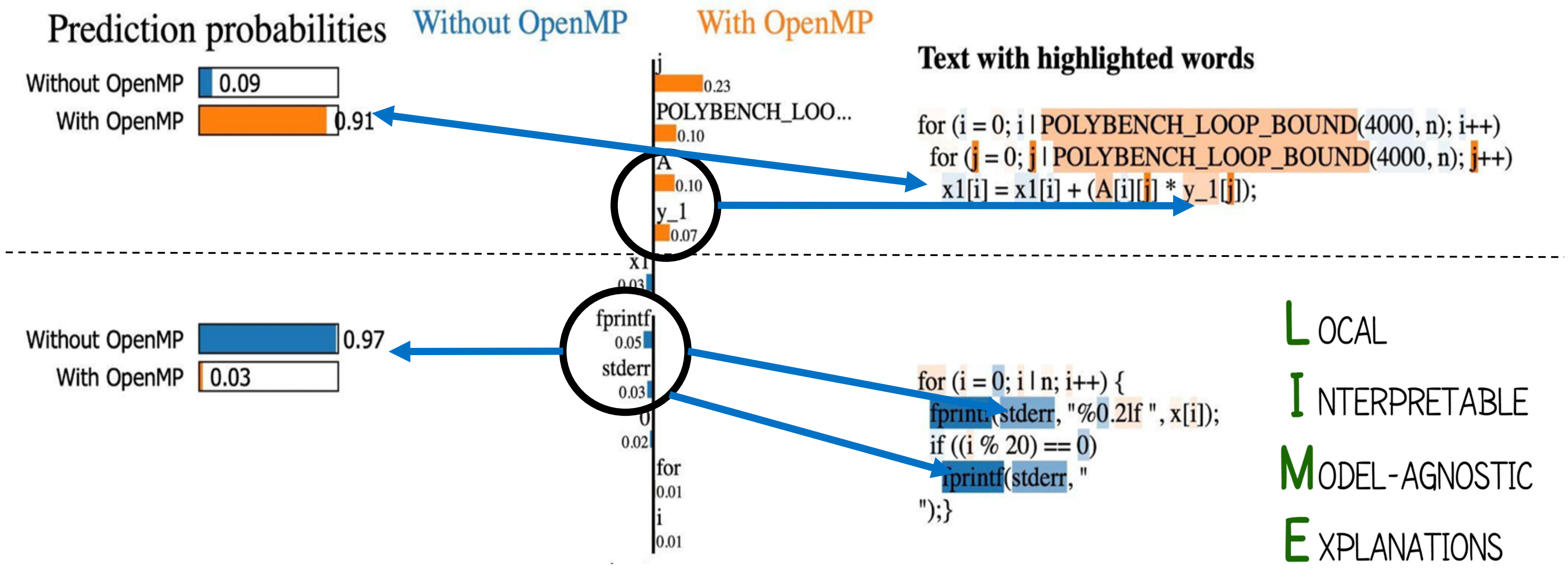
PragFormer @ Hugging Face

```
for (int i = 0; i < size; ++i) {
    sum += A[i] * B[i];
}
```

PragFormer Pragma	PragFormer Private	PragFormer Reduction
Yes / No	**Yes** / No	**Yes** / No

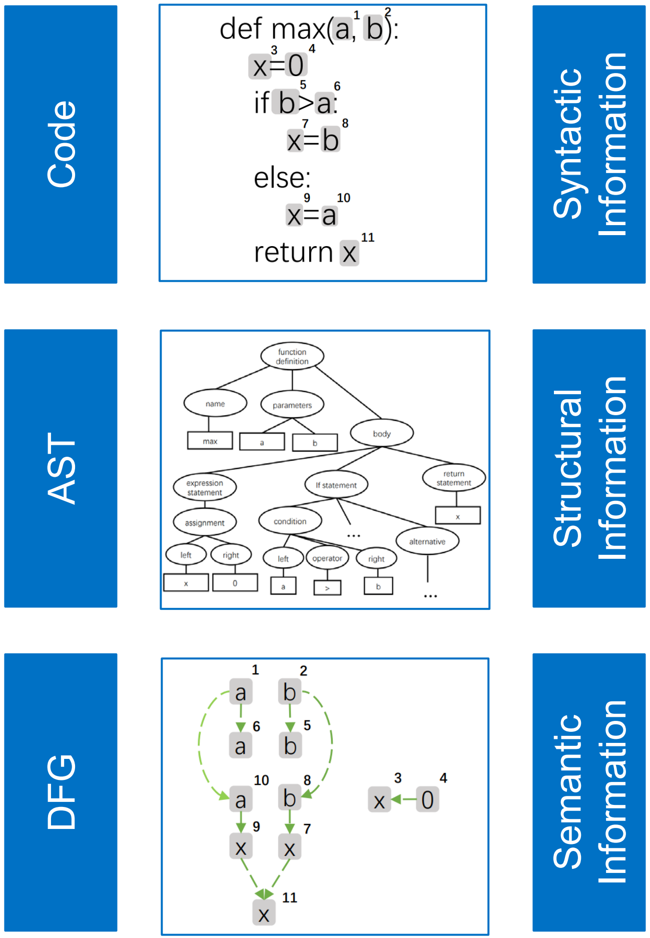
PragFormer

Strongly influenced by local semantics



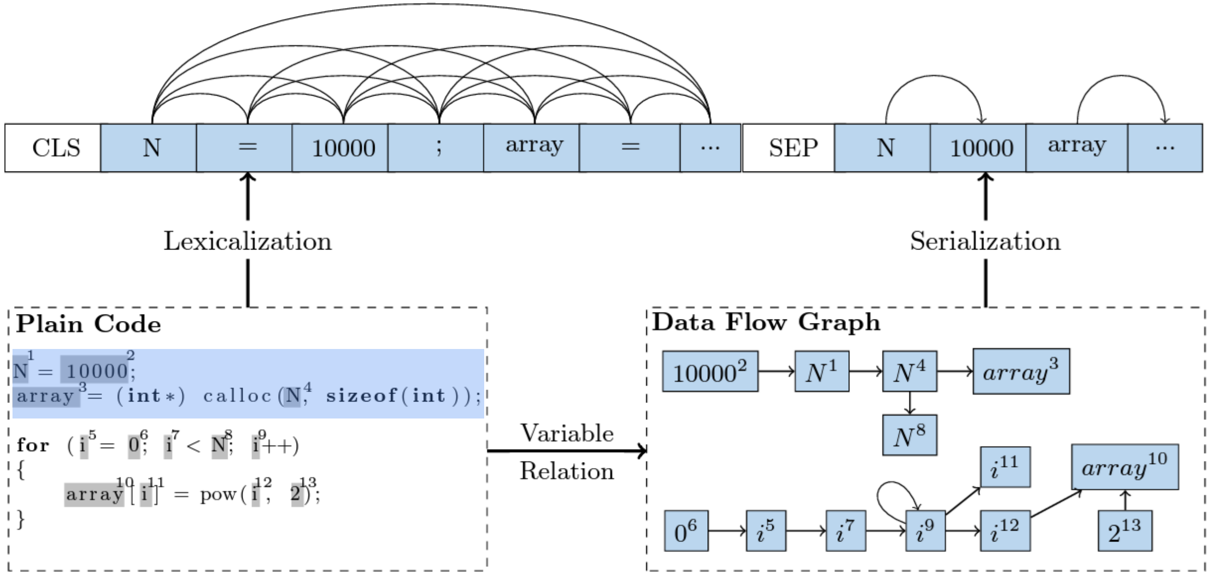
Can We Do Better?

- Exploit different code representations
- Provide extended scope of code
- Utilize an LLM pre-trained on codes
- Use a unified model



OMPify
Input Format

- Ompify classifies the need for the parallel for pragma and its clauses simultaneously
- It is utilizing the extended scope of the for loop as supplementary information
- It is using DFG as an additional code representation



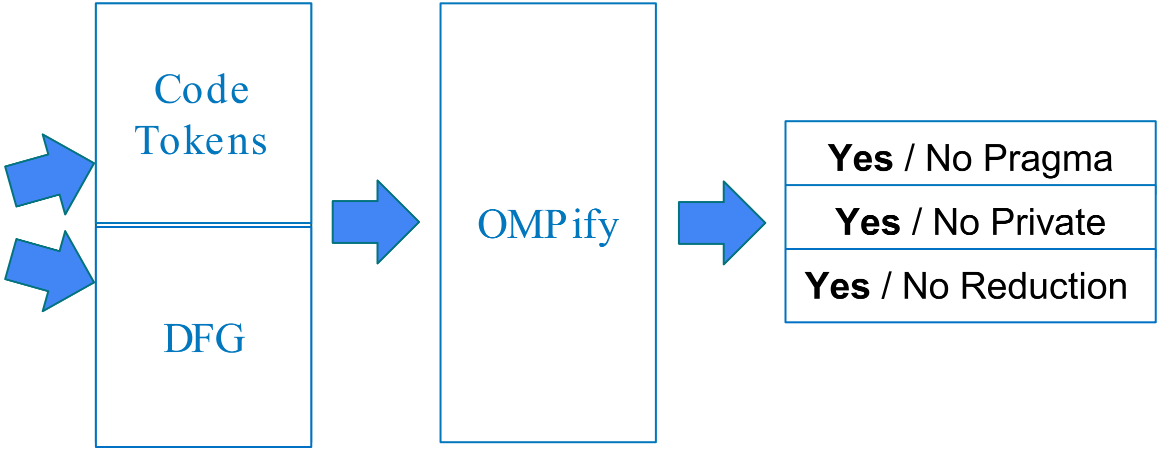
OMPify

Unified model for multiple code representations

Multiple Code Representations

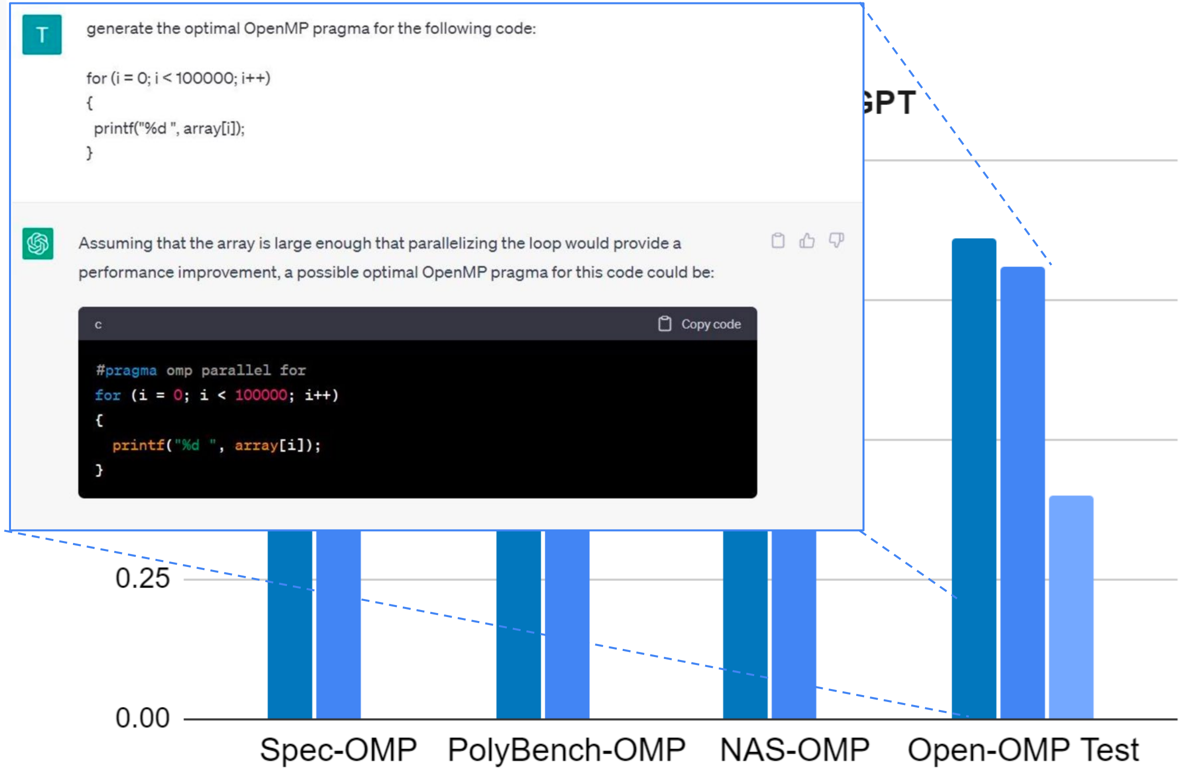
Unified Model

```
Int size = 10000;
int* A =  (int*)malloc(sizeof(int) * size);
int* B =  (int*)malloc(sizeof(int) * size);
int sum = 0;

for (int i = 0; i < size; ++i) {
    sum+= A[i] * B[i];
}

printf("sum = %d", sum);
```



Results

- The data flow graph is found to be the most influential representation.
- The extended scope provides a slight improvement.
- Using a unified model is found to be superior to using separate models.

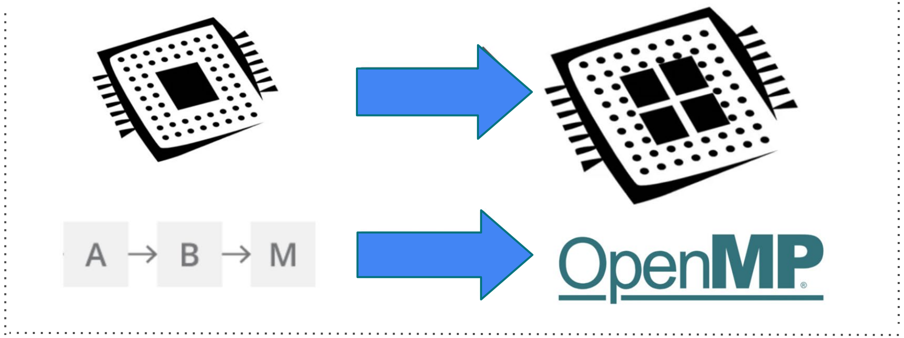


Pragma classification accuracy

What About Pragma Generation?

```
for (int i = 0; i < size; ++i) {
    sum+= A[i] * B[i];
}
```

TokomPiler




```
#pragma omp parallel for
reduction(+:sum)
```

TokomPiler

```
for (int var_64 = num_5; var_64 < var_826; ++var_64) {
    var_3+= arr_26[var_64] * arr_109[var_64];
}
```

Comp-Coder

```
#pragma omp parallel for
reduction(+:var_3)
```

Scope is all you need:

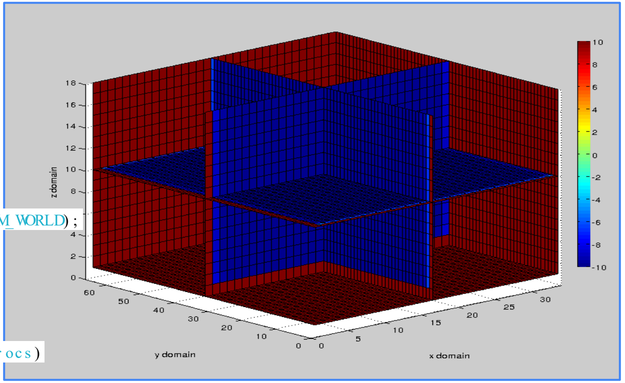
Intro and objectives - LLMs for HPC:	Gal Oren
Novel HPC code database - HPCorpus:	Niranjan Hasabnis
Compiler-oriented tokenizer - Tokompiler:	Vy A. Vo
Downstream Task #1 - OpenMP:	Tal Kadosh
Downstream Task #2 - MPI:	Nadav Schneider

Distributed Parallelism-Domain Decomposition

CHALLENGES

- Is parallelization possible?
- Comprehensive code understanding
- Misplace send/receive functions
- Selecting the right functions in the right locations

MPI programming poses challenges!




```
int main(argc,argv)
{
    int done = 0, n = 10000, rank, numprocs, i;
    double pi_total, pi, h, sum, x;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    while (!done)
    {
                MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
                if (n == 0) break;

                h   = 1.0 / (double) n;
                sum = 0.0;

                for (i = rank + 1; i <= n; i += numprocs)
        {
                    x = h * ((double) i - 0.5);
                    sum += 4.0 / (1.0 + x*x);
                }

                pi_total = h * sum;
                MPI_Reduce(&pi_total, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
    }
    MPI_Finalize();
    return 0;
}
```

Serial vs. Parallel Code

OpenMP
Pragma based

```
#pragma omp parallel for private(i) reduction(+:sum)
for (int i = 0; i < size; ++i) {
    sum+= A[i] * B[i];
}
```

Serial Code

Parallel Code

MPI
Function based

```
int main(argc,argv)
{
    int done = 0, n = 10000, rank, numprocs, i;
    double pi_total, pi, h, sum, x;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&ran

    while (!done)
    {
                MPI_Bcast(&n, 1, _INT, 0, MPI_COMM_WORLD);
                if (n == 0) break;

                h   = 1.0 / (double) n;
                sum = 0.0;

                for (i = rank + 1; i <= n; i += numprocs)
        {
                    x = h * ((double) i - 0.5);
                    sum += 4.0 / (1.0 + x*x);
                }

                pi_total = h * sum;
                MPI_Reduce(&pi_total, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
    }
    MPI_Finalize();
    return 0;
}
```

?

Serial Code

MPI ROADMAP (For Automatic Parallelization)

Q1 SCOPE

What is the model inputs?
Repositories?
For loops?
MPI_INIT to MPI_FINALIZE constructs?

Q2 SERIALIZATION

How to serialize the code: MPI Variables removal?
MPI Functions removal?
Rule based model?

Q3 TASK

Do we train a Generative or a Classification model

MPI-rical - Partial Solution

A1
SCOPE
What is the model inputs?
MPI_INIT to MPI_FINALIZE constructs?

A2
SERIALIZATION
How to serialize the code:
MPI Functions removal

A3
TASK
Train a Generative model

MPI-rical
Partial Solution

A1

SCOPE

What is the model inputs?
MPI_INIT to MPI_FINALIZE constructs?

```
int main(argc,argv)
{
    int done = 0, n = 10000, rank, numprocs, i;
    double pi_total, pi, h, sum, x;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);

    while (!done)
    {
                MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
                if (n == 0) break;

                h   = 1.0 / (double) n;
                sum = 0.0;

                for (i = rank + 1; i <= n; i += numprocs)
        {
                    x = h * ((double)i - 0.5);
                    sum += 4.0 / (1.0 + x*x);
                }

                pi_total = h * sum;
                MPI_Reduce(&pi_total, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
    }
    MPI_Finalize();
    return 0;
}
```

MPI-rical Partial Solution

A2

SERIALIZATION

How to serialize the code:
MPI Functions removal

```
int main(argc,argv)
{
    int done = 0, n = 10000, rank, numprocs, i;
    double pi_total, pi, h, sum, x;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);

    while (!done)
    {
                MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
                if (n == 0) break;

                h   = 1.0 / (double) n;
                sum = 0.0;

                for (i = rank + 1; i <= n; i += numprocs)
        {
                    x = h * ((double) i - 0.5);
                    sum += 4.0 / (1.0 + x*x);
                }

                pi_total = h * sum;
                MPI_Reduce(&pi_total, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
    }
    MPI_Finalize();
    return 0;
}
```

SEMI serial code

MPI-rical - Partial Solution

A3

TASK

Train a Generative model

Generative Model

Code Completion

```
int main(argc,argv) { int done
= 0, n = 10000, rank, numprocs,
i; double pi_total, pi, h, sum,
x; [MASK] [MASK] [MASK]
```

Code Translation

```
int main(argc,argv) { int done
= 0, n = 10000, rank, numprocs,
i; double pi_total, pi, h, sum,
x;
```

```
int main(argc,argv) { int done =
0, n = 10000, rank, numprocs, i;
double pi_total, pi, h, sum, x;
MPI_Init(..); MPI_Comm_size(..);
MPI_Comm_rank(..);
```

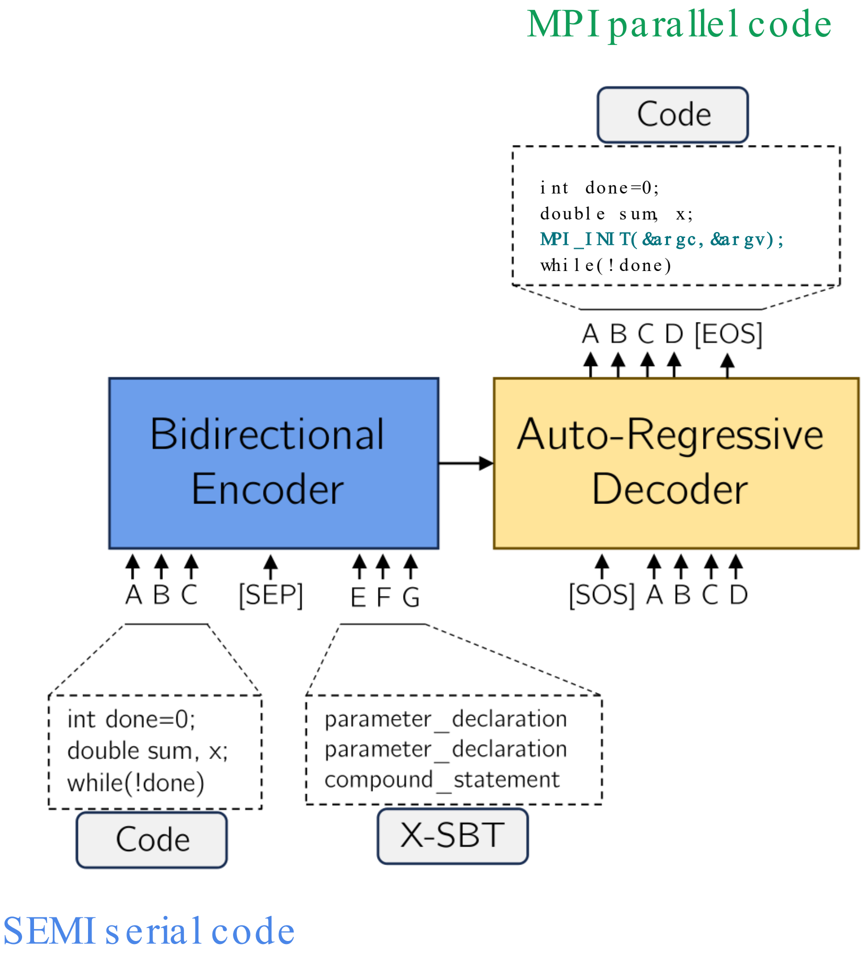
MPI-rical
Partial Solution

A3

TASK

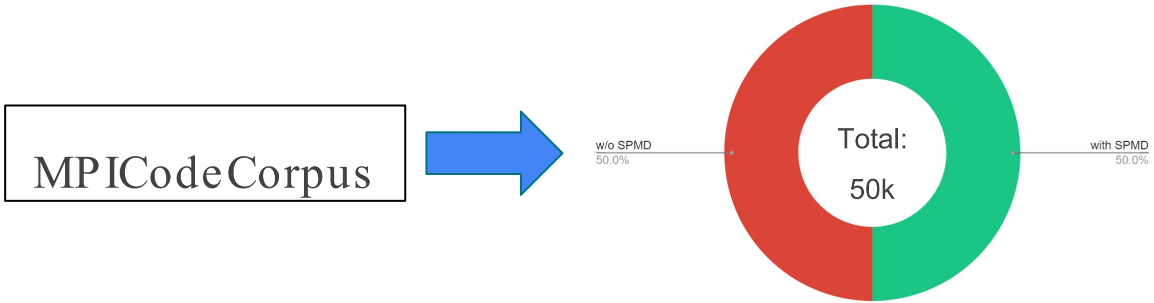
Train a Generative model

Code Translation



Dataset

- SCOPE: Init to Finalize Constructs.
- 25k SPMD and 25k non-SPMD.
- NEGATIVE EXAMPLES: created by MPI functions removal.




```
int main(argc,argv)
{
    int done = 0, n = 10000, rank, numprocs, i;
    double pi_total, pi, h, sum, x;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);

    while (!done)
    {
        MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
                if (n == 0) break;

                h   = 1.0 / (double) n;
                sum = 0.0;
```

Positive Example

```
int main(argc,argv)
{
    int done = 0, n = 10000, rank, numprocs, i;
    double pi_total, pi, h, sum, x;




    while (!done)
    {

                if (n == 0) break;

                h   = 1.0 / (double) n;
                sum = 0.0;
```

Negative Example

Dataset

- SCOPE: Init to Finalize Constructs.
- 25k SPMD and 25k non-SPMD.
- NEGATIVE EXAMPLES: created by MPI functions removal.

# Line	Amount
$\leq$ 10	2,670
11-50	22,361
51-99	14,078
$\geq$ 100	10,575

Function	Amount
MPI_Finalize	35,983
MPI_Comm_rank	32,312
MPI_Comm_size	28,742
MPI_Init	25,114
MPI_Recv	10,340
MPI_Send	9,841
MPI_Reduce	8,503
MPI_Bcast	5,296

GitHub → GitHub-Clone-all → **F, C,C++ files** → Parser → **MPI_INIT to MPI_FINALIZE**

MPI-rical - Partial Solution

```
int main(argc,argv)
{
    int done = 0, n = 10000, rank, numprocs, i;
    double pi_total, pi, h, sum, x;


    while (!done)
    {
                    if (n == 0) break;

                    h   = 1.0 / (double) n;
                    sum = 0.0;

                    for (i = rank + 1; i <= n; i += numprocs)
            {
                        x = h * ((double) i - 0.5);
                        sum += 4.0 / (1.0 + x*x);
                    }

                    pi_total = h * sum;

    }

    return 0;
}
```

MPIrical
Can it be
SPMD
parallelized?

MPI Function Classification

(MPI_Init(..), MPI_Comm_rank(..), MPI_Send(..))

MPI Function Location

Yes / No

Results

Generation as Classification

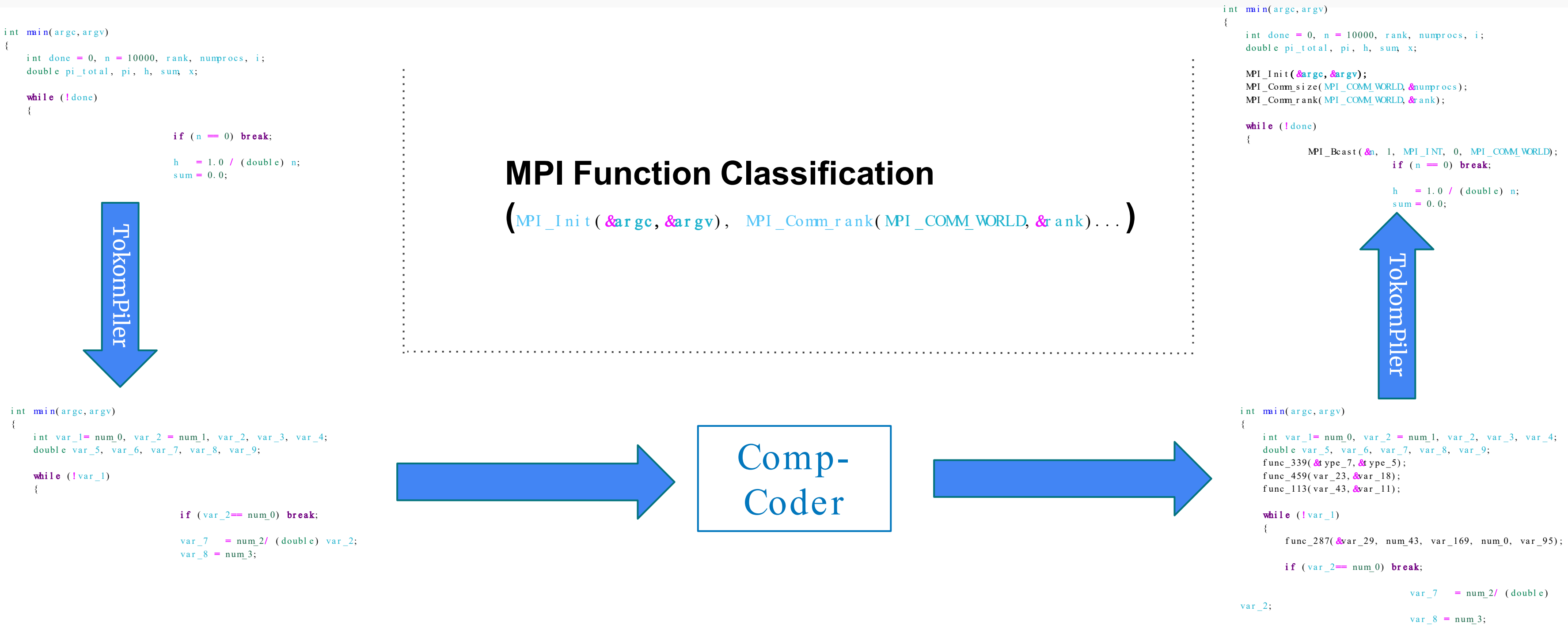
TP — FP — FN

```
Ground-Truth: int main(argc,argv){ int done = 0, rank, numprocs; MPI_Init(..); MPI_Comm_size(..); MPI_Comm_rank(..); while(!done){
  Prediction: int main(argc,argv){ int done = 0, rank, numprocs; MPI_Init(..); MPI_Bcast(..); while(!done){
```

Code	M-F1	M-Precision	M-Recall
Array Average	0.88	1.0	0.8
Vector Dot Product	0.88	1.0	0.8
Min-Max	0.66	1	0.5
Matrix-Vector Multiplication	0.9	0.83	1.0
Sum (Reduce & Gather)	0.8	1.0	0.6
Merge Sort	1.0	1.0	1.0
Pi Monte-Carlo	1.0	1.0	1.0
Pi Riemann Sum	1.0	1.0	1.0
Factorial	0.88	1.0	0.8
Fibonacci	1.0	1.0	1.0
Trapezoidal Rule (Integration)	1.0	1.0	1.0
Total	**0.91**	**0.98**	**0.86**

Quality Measure	*MPICodeCorpus*
M-F1	0.87
M-Precision	0.85
M-Recall	0.89
MCC-F1	0.89
MCC-Precision	0.91
MCC-Recall	0.87
BLEU	0.93
Meteor	0.62
Rouge-1	0.95
ACC	0.57

What Next?
Full MPI Program Generation



Scope is all you need:

Intro and objectives - LLMs for HPC:	Gal Oren
Novel HPC code database - HPCorpus:	Niranjan Hasabnis
Compiler-oriented tokenizer - Tokompiler:	Vy A. Vo
Downstream Task #1 - OpenMP:	Tal Kadosh
Downstream Task #2 - MPI:	Nadav Schneider

Thanks!

Contact us:

galoren@cs.technion.ac.il