LM4HPC: Towards Effective Language Model Application in High-Performance Computing

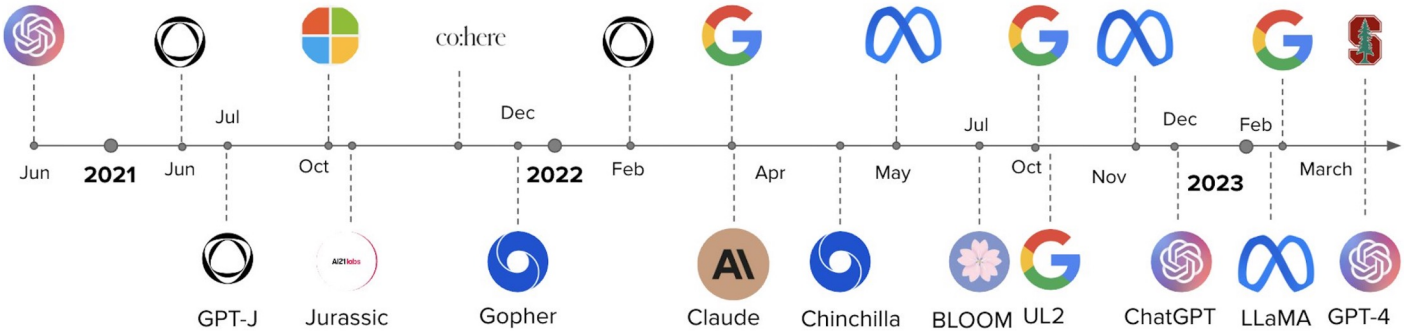
Second International Workshop on Conversational AI Interfaces for HPC
July 22, 2024

Le Chen (Iowa State Uni), Pei-Hung Lin, Tristan Vanderbruggen, Chunhua Liao, Bronis de Supinski (LLNL)
Murali Emani (ANL)

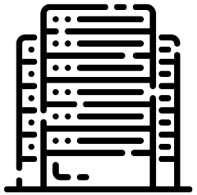
Background - Large Language Models (LLMs):

A **LLM** is an artificial intelligence (AI) model designed to **understand** and **generate** human-like texts.

- Architecture: deep learning architectures
- Training data: vast amounts of text data
- Capabilities:
  - Exhibit early indications of artificial general intelligence capabilities.
  - Have been applied to a wide range of Natural Language Processing (NLP) tasks: translation, summarization, generation, etc.



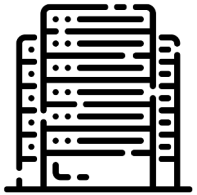
Background - LMs, PLP and HPC:



Due to their impressive success in NLP, LMs have been:

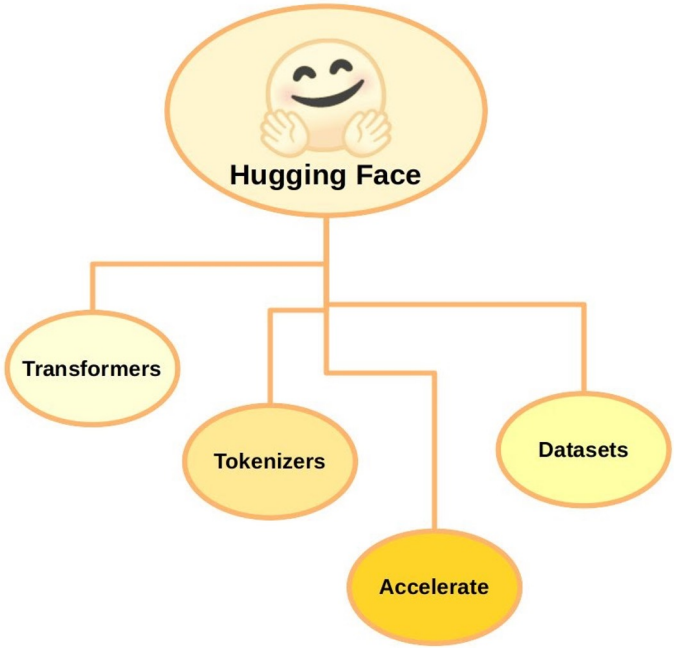
- applied to Programming Language Processing (PLP) tasks such as auto-completion, code translation and algorithm classification;
- gained growing interest in leveraging LMs to tackle High-Performance Computing tasks.

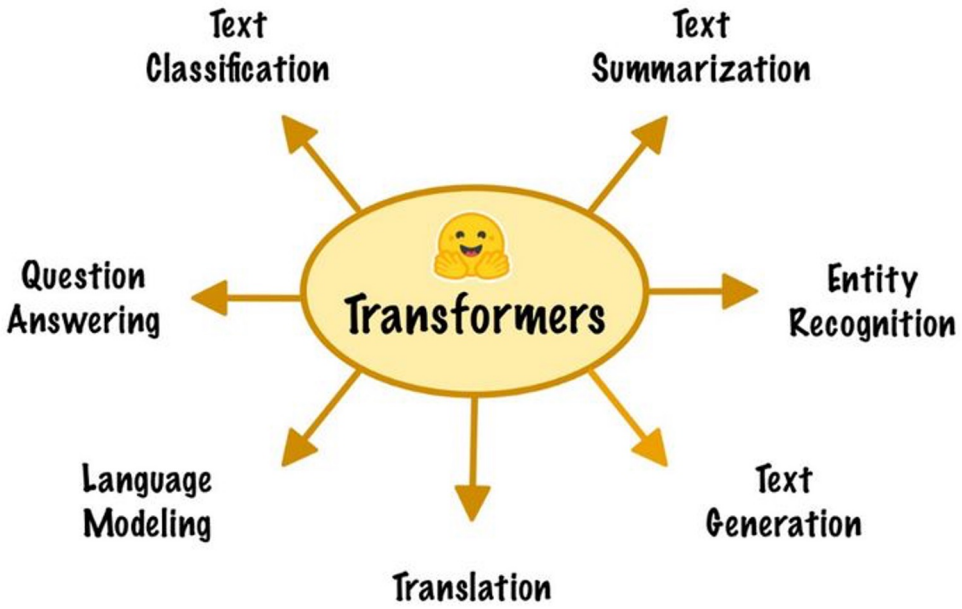
Background - LMs, PLP and HPC:



Code Analysis	Code Generation	Others
Compiler Analysis	Code Completion	Test Case Generation
Algorithm Classification	Natural Language-to-Code	Code Search
Code Similarity Analysis	Code Translation	Question Answering
Documentation Generation	Code Repair	Code Review
Parallelism Detection	Code Migration	Decompilation
Defect Detection	Compilation	IR-to-Source Translation

Related Work - Hugging Face





Motivation

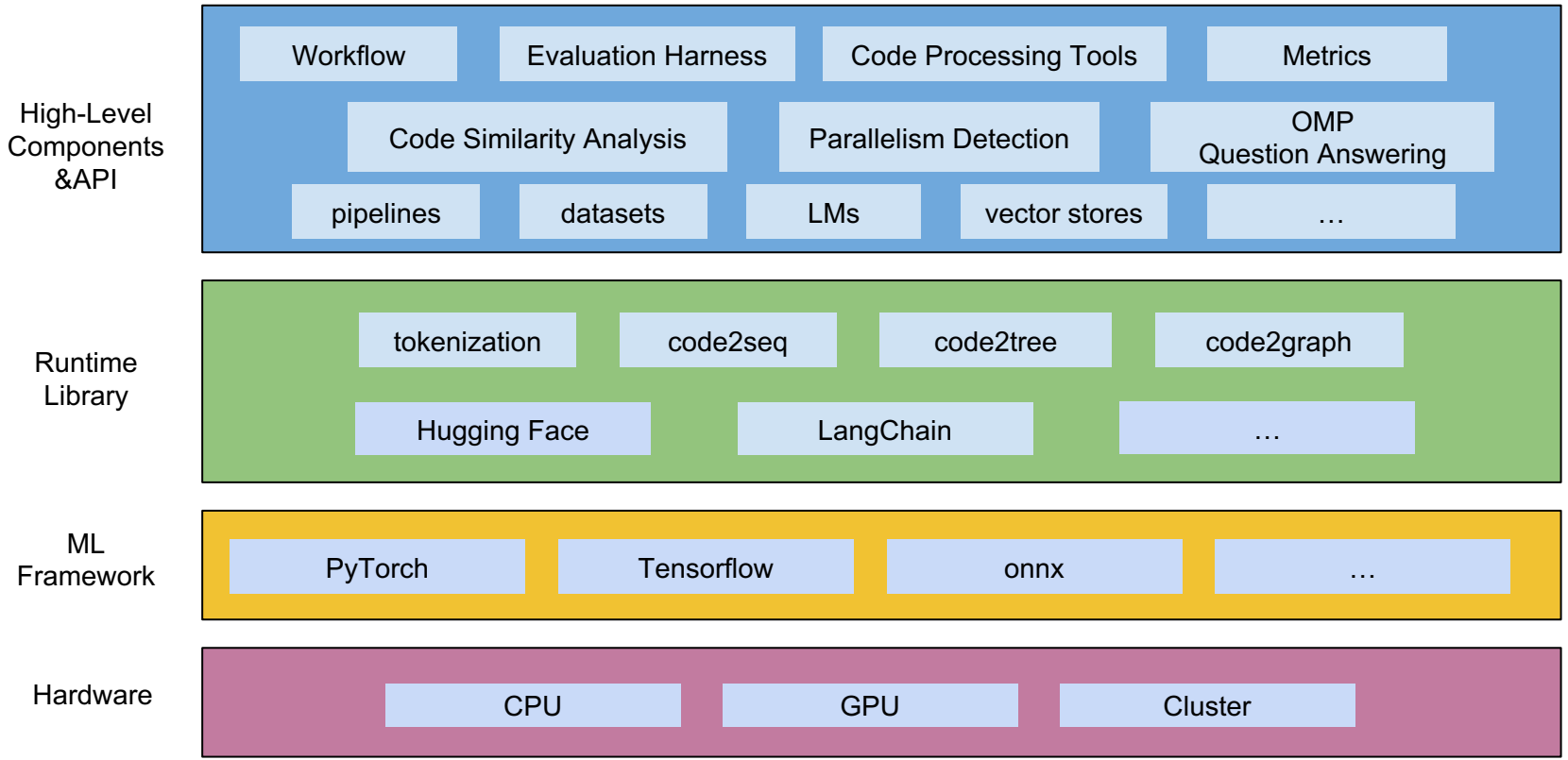
However, it is difficult for machine learning newcomers to quickly leverage LMs for HPC tasks:

* HPC-specific pipelines are missing
* HPC-specific datasets
* Dataset preprocessing
* Incorporating domain-specific knowledge
* Model selection
* Model fine-tuning
* Results evaluation
* etc.

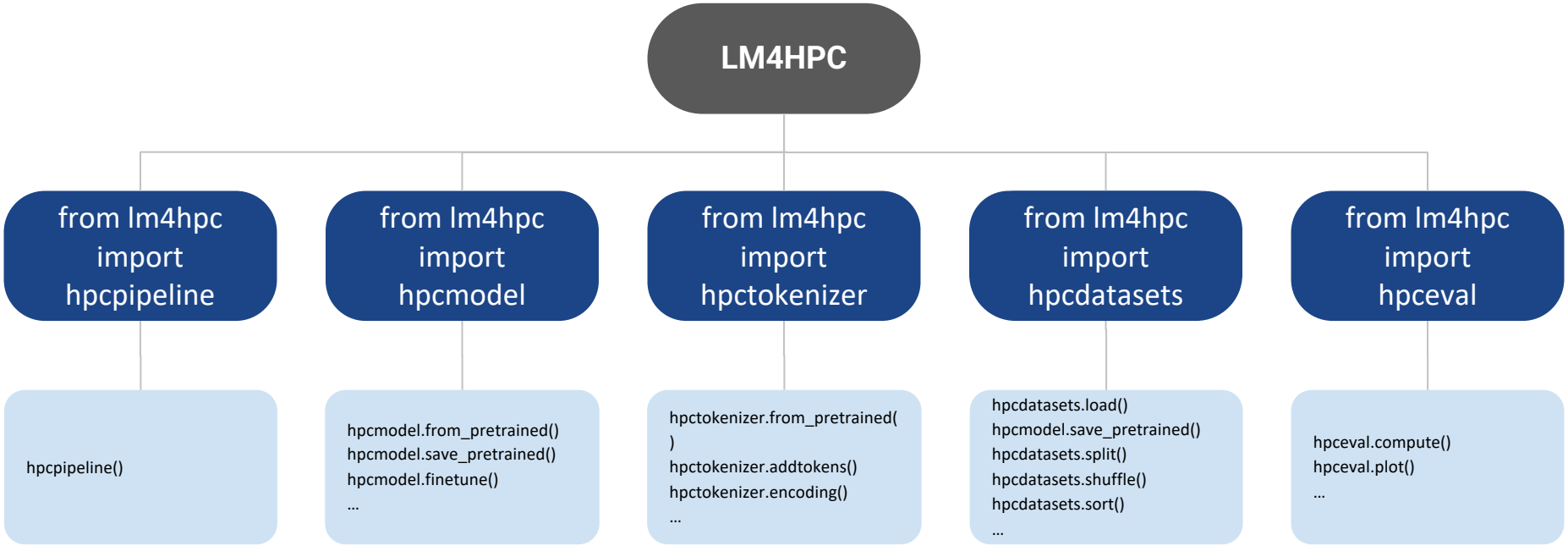
Design Goals of LM4HPC

1. Utilize state-of-the-art ML techniques for HPC tasks.
2. Offer APIs that align with widely accepted standards in the ML domain.
3. Simplify HPC-related training and inferencing tasks with HPC-specific pipelines.
4. Collect, create and process HPC datasets, ensuring they're ready for language models.
5. Generate leaderboards by evaluating language models on HPC tasks:

LM4HPC - Design Overview: Architecture



LM4HPC - Design Overview: APIs



LM4HPC classes	Description	Example API functions
`hpcmodel`	Fine-tune text-based (HF, OpenAI) and graph-based models, including local private ones, for HPC tasks	`hpcmodel.from_pretrained(model_name_or_path: Optional[str], *model_args, **kwargs)`
		`hpcmodel.save_pretrained(model_name_or_path: str, *model_args, **kwargs)`
		`hpcmodel.finetune()`
`hpctokenizer`	APIs to represent code in either tokenized text, trees, or graphs	`hpctokenizer.from_pretrained(model_name_or_path: Optional[str], *model_args, **kwargs)`
		`hpctokenizer.addtokens(contentsingle_word=False, strip=False, normalized=True)`
		`hpctokenizer.encoding()`
`hpcdatasets`	Load and process HPC datasets	`hpcdatasets.load(path: str, data_files: Union[str, List, Dict, None], **kwargs)`
		`hpcdatasets.split(dataset: hpcdatasets, partition: [float, float, float], **kwargs)`
		`hpcdatasets.shuffle(dataset: hpcdatasets, **kwargs)`
		`hpcdatasets.sort(dataset: hpcdatasets, **kwargs)`
`hpcpipeline`	Pre-built pipelines for common PLP tasks	`hpcpipeline(task: str, model_name_or_path: str, *model_args, **kwargs)`
`hpceval`	Evaluate the results of various models	`hpceval.compute(task: str, models_name_or_path: [[str]], data_files: Union[str, List, Dict, None], *model_args, **kwargs)`
		`hpceval.plot(shape: str)`

LM4HPC APIs: Compared with Hugging Face APIs

Functionality	Hugging Face APIs	LM4HPC APIs
models	NLP models (BERT, T5, etc.)	Fine-tune text-based (HF, OpenAI) and graph-based models, including local private ones, for HPC tasks
Tokenization	NLP tokenizers (text-based)	APIs to represent code in either tokenized text, trees or graphs
Pipelines	Pre-built pipelines for common NLP tasks	Pre-built pipelines for common HPC PLP tasks
Model training	Fine-tune training for NLP	Fine-tune training for HPC PLP
Deployment	Deployment to web and mobile apps	Support HPC build environments like Makefiles in a large scale codebase.

LM4HPC Datasets

Our **LM4HPC** framework aims to provide simple interfaces to publish and consume HPC data. Currently we have generated:

1. DRB-ML: parsed and tokenized code from DataRaceBench.
2. OMP4Par: OpenMP code crawled from GitHub and processed for parallelization detection analysis.
3. OpenMP QA: created for evaluating LLMs for OpenMP related questions.

Dataset: DRB-ML

Data collection:

- Source: DataRaceBench available at https://github.com/LLNL/dataracebench
- Process:
  - Trim code
  - extract kernels
  - tokenization
- Generated data: tokenized C/C++ OpenMP code.
- Dataset statistics: 658 kernels

Dataset: DRB-ML

Data example:


```
{
    "name": "DRB001-antidep1-orig-yes.c.1.txt",
    "func": "#pragma omp parallel for\n    for (i = 0; i < len - 1; i++)\n        a[i] = a[i + 1]
+ 1;\n",
    "token": ["<s>", "<s>", "#", "pr", "ag", "ma", "\u0120o", "mp", "\u0120parallel",
"\u0120for", "\u010a", "\u0120", "\u0120", "\u0120", "\u0120for", "\u0120(", "i", "\u0120=",
"\u01200", ";", "\u0120i", "\u0120<", "\u0120len", "\u0120-", "\u01201", ";", "\u0120i", "++)",
"\u010a", "\u0120", "\u0120", "\u0120", "\u0120", "\u0120", "\u0120", "\u0120", "\u0120a", "[",
"i", "]", "\u0120=", "\u0120a", "[", "i", "\u0120+", "\u01201", "]", "\u0120+", "\u01201", ";",
"\u010a", "</s>", "</s>"],
    "tokens_ids": [0, 0, 10431, 4862, 1073, 1916, 1021, 6195, 12980, 13, 50118, 1437, 1437, 1437,
13, 36, 118, 5457, 321, 131, 939, 28696, 25528, 111, 112, 131, 939, 49346, 50118, 1437, 1437,
1437, 1437, 1437, 1437, 1437, 1437, 10, 10975, 118, 742, 5457, 10, 10975, 118, 2055, 112, 742, 2055,
112, 131, 50118, 2, 2],
    "idx": "0"
}
```

Dataset: OMP4Par

Data collection:

- data content: crawled from GitHub to get OpenMP C/C++ code.
- data label:
  - extract code into loop snippets. loops with OpenMP annotation are marked as parallelizable (label: 1); the rest without OpenMP clauses are marked as non-parallelizable (label: 0);
  - extract the OpenMP clauses: simd, task, reduction, private, target
- data validation: examine the correctness with open-source parallelization tools like DiscoPoP, autoPar and Pluto.
- dataset statistics:
  - total:32,570 loops
  - label 1: 17,349; label 0: 15,221

Dataset: OMP4Par

Data example:


```
{
    "name": "fusiled_large-graphs-openmp_graph_algo.c_0",
    "code": "for (int i = 0; i < (empty_flag * size); i++){for (int i = 0; i < (empty_flag * size);
i++) \n { \n   if (F[i] != 0)\n  {\n        empty_flag = 0;\n  }\n\n}",
    "pragma": "omp parallel for"
    "idx": "0"
    "label": "1"
}
```

Dataset: OMPQA

An OpenMP question and answering dataset for evaluating LLMs' knowledge about HPC.

Data collection:

- Set data category: basics, examples, compilers, and benchmarks
- Data collect: extracted from OpenMP documentations, quiz, and tutorials to cover various OpenMP topics.
- Answer generation: extract from OpenMP documentations, quiz, tutorials, and generated by human experts.

Dataset: OMPQA

Category	Count	Example Questions
Basics	40	Q: What does the #pragma omp for construct do in OpenMP?
		A: #pragma omp for construct specifies that the iterations of associated loops will be executed in parallel by threads in the team in the context of their implicit tasks.
Code Examples	20	Q: Give an example OpenMP C code for computing PI using numerical integration.
		A: #include <stdio.h> #include <omp.h> int main() { int i; int array_size = 10; int array[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}; int result[10]; #pragma omp parallel for for (i = 0; i < array_size; i++) { result[i] = array[i] * array[i]; printf("Thread %d calculates element %d\n", omp_get_thread_num(), i); } printf("Result array: "); for (i = 0; i < array_size; i++) { printf("%d ", result[i]); } printf("\n"); return 0; }
Compilers	24	Q: In what language is LLVM written?
		A: All of the LLVM tools and libraries are written in C++ with extensive use of the STL.
Benchmarks	23	Q:What are the NAS Parallel benchmarks?
		A: NAS Parallel Benchmarks are a set of benchmarks targeting performance evaluation of highly parallel supercomputers.

Popular Models and Our Picks

Model			Training data		Token Limit	Avail.
Name	Released	Size	Type	Size		
BERT	2018/10	340M	Text	3.5B words	512	Weights
CodeBERT	2020/11	125M	Mixed	2.1M(bimodal) 6.4M (unimodal)	512	Weights
Megatron	2021/04	1T	Text	174 GB	512	Weights
GraphCodeBERT	2021/05	110M	Code	2.3M functions	512	Weights
CodeT5	2021/11	770M	Code	8.35M instances	512	Weights
GPT-3	2022/03/15	175B	Mixed	500B tokens	4096	Weights
LLaMA	2023/02/24	7~65B	Mixed	1.4T tokens	4096	Weights*
GPT-4	2023/03/14	1T	Mixed	undisclosed	8k/32k	API*
BARD	2023/03/21	1.6B	Mixed	1.56T words	1000	API
Cerebras-GPT	2023/03/28	0.11~13B	Text	800 GB	2048	Weights
Dolly 2.0	2023/04/12	3~12B	Text	15k instr./resp.	2048	Weights
StarCoder	2023/05/4	15B	Code	1T tokens	8192	Weights
StarChat-Alpha	2023/05/4	16B	Code	31k instr./resp.	8192	Weights

LM4HPC Pipelines

Our **LM4HPC** framework extends the pipeline function provided by Hugging Face, adapting it for HPC tasks. We have implemented three pipelines in this paper:

1. Code similarity analysis: for a given pair of code snippets, return the result of code similarity checking.
2. Parallelism detection: for a given loop snippet, return if the loop is parallelizable.
3. OpenMP question answering: for a open-answer OpenMP question, give the answer.

LM4HPC Pipelines - Code Similarity Analysis

Task: deploy a pipeline for code similarity checking.

Example of usage:

```
In [1]: from lm4hpc import hpcpipeline
In [2]: pipe = hpcpipeline(model="/path/to/model/", task="code-similarity")
In [3]: c1 = "def fibonacci(n):\n    if n <= 1:\n        return n\n    else:\n        retu
  ...: rn fibonacci(n-1) + fibonacci(n-2)"
In [4]: c2 = "def fib(n):\n    if n <= 1:\n        return n\n    else:\n        return fib
  ...: (n-1) + fib(n-2)"
In [5]: c3 = "def fib(n):\n    if n <= 1:\n        print(n)\n    else:\n        return fib
  ...: (n-1)"
In [5]: pipe(c1, c2)
similarity results:  99.86%
In [6]: pipe(c1, c3)
similarity results:  74.81%
```

LM4HPC Pipelines - Parallelism Detection

Task: Identify whether a given code loop is parallelizable with OpenMP. we treat the task as binary classification. 1 stands for parallelizable and 0 for non-parallelizable

Example of usage:

```
In [1]: from lm4hpc import hpcpipeline
In [2]: pipe = hpcpipeline(model="/path/to/model/", task="parallelism-detection")
In [3]: func0 = "for (i = 0; i < len - 1; i++)\n    a[i] = a[i + 1] + 1;}\n"

  ...:   return 0;\n\n    }\n\n\n\n    if (!bs->backing_hd) {\n\n        return 0;\n\n    }\n\n\n\n
  ...: return 1 + bdrv_get_backing_file_depth(bs->backing_hd);\n\n}\n"
In [4]: func1 = "for (c4 = nk; c4 <= ((16 * c2 + 15 < nj + -1 ? 16 * c2 + 15 : nj + -1)); c4++) {\n
  ...:     B[c3][c4] = ((double)c3) * (c4 + 1) / nj;\n}\n"
In [5]: pipe(func0)
parallelism detection results: 0
In [6]: pipe(func1)
parallelism detection results: 1
```

LM4HPC Pipelines - OpenMP Question Answering

Task: a chatbot-like pipeline, answering OpenMP related questions

Example of usage:

```
In [1]: from lm4hpc import hpcpipeline
In [2]: pipe = hpcpipeline(model="gpt-3.5-turbo",task="openmp-question-answering", kb=1)
In [3]: question0 = "Could you explain what OpenMP is?"
In [4]: question1 = "Give a C code example for using #pragma omp atomic"
In [5]: pipe(question0)
answer: OpenMP (Open Multi-Processing) is an application programming interface (API) that enables multi-platform shared memory multiprocessing programming in C, C++, and Fortran on most platforms, including Unix and Microsoft Windows platforms. It consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior. The main aim of OpenMP is to increase the performance of parallel applications and to simplify the development of parallel applications for shared memory systems.'
In [6]: pipe(question1)
answer: Sure! Here\'s an example of using `#pragma omp atomic` in C code:\n\n```c\n#include <stdio.h>\n#include <omp.h>\n\nint main() {\n    int shared_var = 0;\n    #pragma omp parallel num_threads(4)\n    {\n        #pragma omp atomic\n shared_var++;\n    }\n    printf("The final value of shared_var is: %d\\n", shared_var);\n    return 0;\n}\n```\n\nIn this example, we have a shared variable `shared_var` that is initialized to 0. We then create a parallel region with 4 threads using `#pragma omp parallel num_threads(4)`. Within the parallel region, we use `#pragma omp atomic` to increment `shared_var` by 1 in a thread-safe manner. Finally, we print out the final value of `shared_var` to verify that it has been incremented by all 4 threads.'
```

Evaluating LM4HPC by Generating Leaderboards

Why are leaderboards important?

- Researchers and developers
  - Provide an objective benchmark to compare the performance of different models.
  - Drive innovation and encourage researchers and practitioners to continually improve their models.
- For end users
  - Finding the best models for a given task
- For sponsors
  - Track the progress of the research over time by recording the best performance.
  - Identify future priority research directions, based on exposed common challenges and limitations

Leaderboard Experiment Configuration

Hardware	CPU			RAM	Disk		GPU	
	model	Cores	Frequency	memory	type	size	model	memory
Google Colab VM	Intel Xeon	6	2.20GHz	84 GB	HDD	166 GB	NVIDIA A100	40 GB
Dell workstation	Intel Xeon	2	2.10Ghz	128 GB	SSD	1 TB	NVIDIA Quadro RTX 6000	24 GB

Hardware configuration for the leaderboard experiments

Leaderboard Experiment Configuration - Cont.

Software	Version
Python	3.10.6
transformers	4.29.0
LangChain	0.0.174
datasets	2.12.0

Software configuration

Hyperparameters	Value
Temperature	0
input token len.	256
output token len.	256

Hyperparameters configurations for LLMs

LM4HPC - LeaderBoard: Code Similarity Analysis (T1)

Dataset: POJ-104

Model	Precision	Recall	F1
GPT-3.5-turbo	**78.4**	**74.2**	**76.2**
Dolly 2.0 12B	61.9	61.3	61.6
StarChat-Alpha	59.4	56.2	57.8
GraphCodeBERT	52.7	60.3	56.3
CodeBERT	51.5	59.4	55.2

Dataset: DRB-ML

Model	Precision	Recall	F1
GPT-3.5-turbo	**82.4**	**81.3**	**81.8**
StarChat-Alpha	79.6	77.4	78.5
Dolly 2.0 12B	74.3	73.2	73.7
GraphCodeBERT	79.4	77.9	78.6
CodeBERT	76.9	74.5	75.7

LM4HPC - LeaderBoard: Parallelism Detection (T2)

Dataset: OMP4Par

Model	Precision	Recall	F1
GPT-3.5-turbo	90.6	**89.3**	**89.9**
augAST	**92.1**	82.4	87.0
DeepSCC	82.7	81.4	82.0
StarChat-Alpha	85.7	68.2	75.9
Dolly 2.0 12B	64.2	63.7	63.9

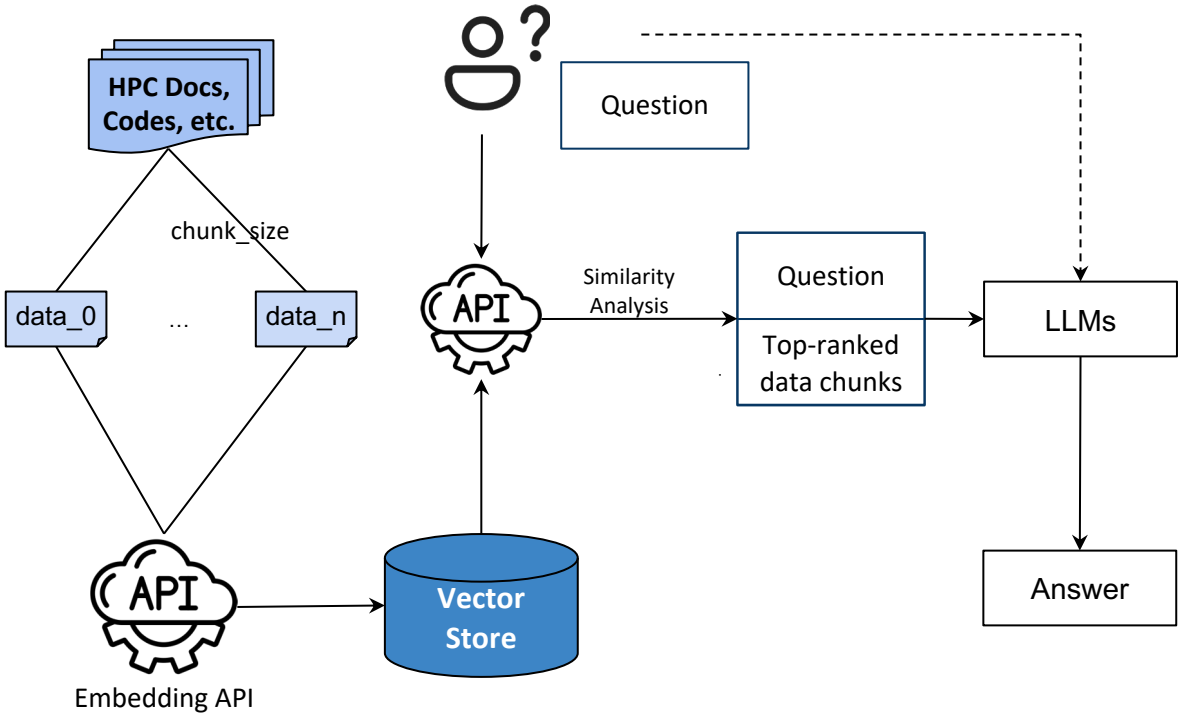
Dataset: DRB-ML

Model	Precision	Recall	F1
GPT-3.5-turbo	90.0	**98.9**	**94.2**
augAST	**91.4**	72.3	80.7
DeepSCC	80.4	79.5	79.9
StarChat-Alpha	81.9	20.3	32.5
Dolly 2.0 12B	40.0	11.2	2.17

LM4HPC - External Knowledge with LangChain (T3:OpenMP QA)

Why do we need LangChain?

- General-purpose LLMs are not trained with enough HPC data.
- LLMs has a huge amount of parameters, making it expensive to fine-tune.
- Limited input token lengths allowed
- LangChain: help LLM access the up-to-date external information.



Leaderboards Generated using our APIs and Datasets: 3/3

Model	GPU Mem. (GB)	CPU Mem. (GB)	Avg Time(s)	BLEU	AVG ROUGE-L		
					Recall	Precision	F1
GPT-3.5-turbo+LangChain	4.1	0.1	21.452	0.147↑	0.347↓	0.262↑	0.259↑
GPT-3.5-turbo	4.2	0.1	12.749	0.139	0.446	0.231	0.257
Dolly 2.0 12B+LangChain	27.4	39.8	7.217	0.084↑	0.228↑	0.232↓	0.182↑
Dolly 2.0 12B	27.1	39.2	8.147	0.06	0.208	0.312	0.148
StarChat-Alpha	6.8	18.9	29.732	0.082	0.322	0.149	0.172
Cerebras-GPT 13B	52.6	11.7	590.763	0.071	0.319	0.089	0.112

Q&A Leaderboard using the OMPQA dataset

Open Challenges

LLMs: limited input sequence lengths, expensive to train/fine-tune

Hardware issues:

- Platform tested: Pascal, Lassen, Tux
- SSL errors, python building on IBM machines, disk quotas, etc.

How to verify the correctness of answers

- open questions v.s. multiple-choice question
- answer in natural language vs. answer in programming language

Limited HPC datasets:

- time consuming to create questions and ground truth answers.

Conclusion

Summary:

- We presented the LM4HPC framework to facilitate the application of language models for tasks specific to High-Performance Computing.
- We have developed the LM4HPC APIs, new datasets, and pipelines.
- Our experimental findings suggest that GPT-3 performs competitively among tested models, despite not being specifically designed for HPC tasks.
- There is significant room for improvement in answering OpenMP questions.

Our future work includes:

- Explore automated approaches to generating HPC-specific datasets.
- Support the fine-tuning of models for HPC-related tasks
- Support MPI, performance analysis and optimization tasks.
- Support better metrics for measuring quality of generated codes.

LM4HPC - Q&A

Thank you!

Questions?