Scan me!

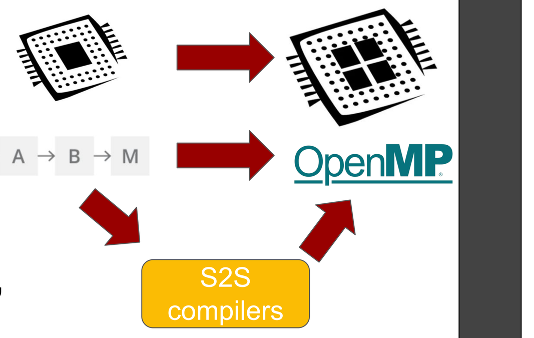
# Learning to Parallelize Source Code via OpenMP with Transformers


Re'em Harel[1,2,3], Yuval Pinter[1], Gal Oren[3,4]* * galoren@cs.technion.ac.il

[1] Department of Computer Science, Ben-Gurion University of the Negev, Israel
[2] Department of Physics, Nuclear Research Center – Negev, Israel
[3] Scientific Computing Center, Nuclear Research Center – Negev, Israel
[4] Faculty of Computer Science, Technion, Israel


## Introduction

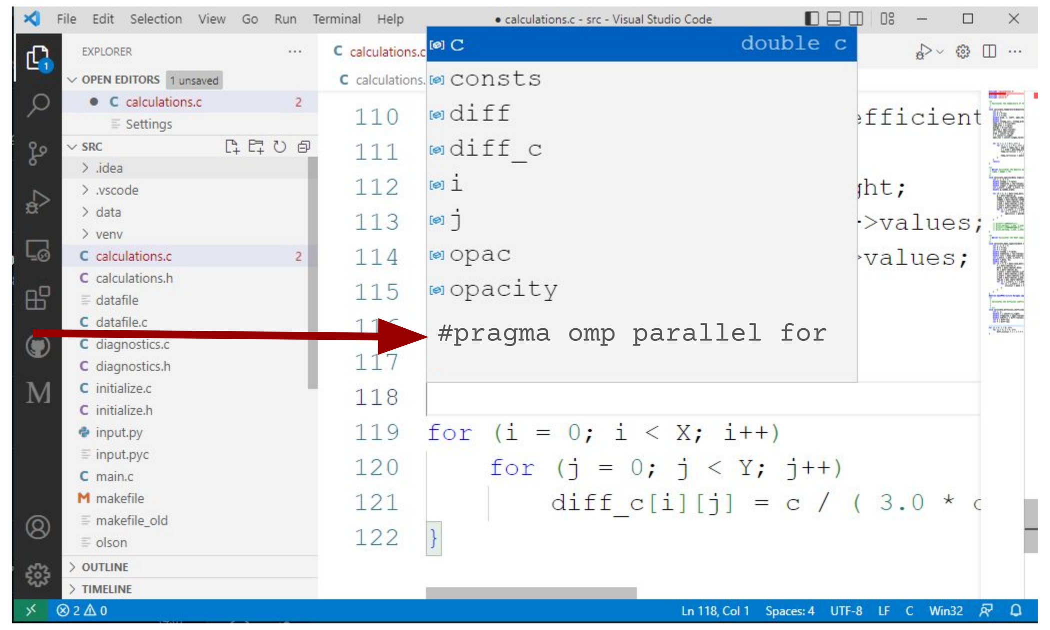
- In order to fully utilize **shared-memory architectures**, developers needed to introduce **OpenMP** to their code.
- Introducing OpenMP schemes is a **hard** and **tedious** task, especially in legacy codes. Thus, many automatic source-to-source compilers (S2S) have been created to cope with this task.
- Nevertheless, S2S compilers have many **pitfalls** due to the **complexity** in parsing the source code; **limited robustness** to the input; and **time consuming** data-dependence algorithms.
- Due to recent innovations in **natural language processing (NLP)**, such as **Transformers**, the possibility of tackling the task of introducing OpenMP directives with **NLP models**, rises.




## Research Objective

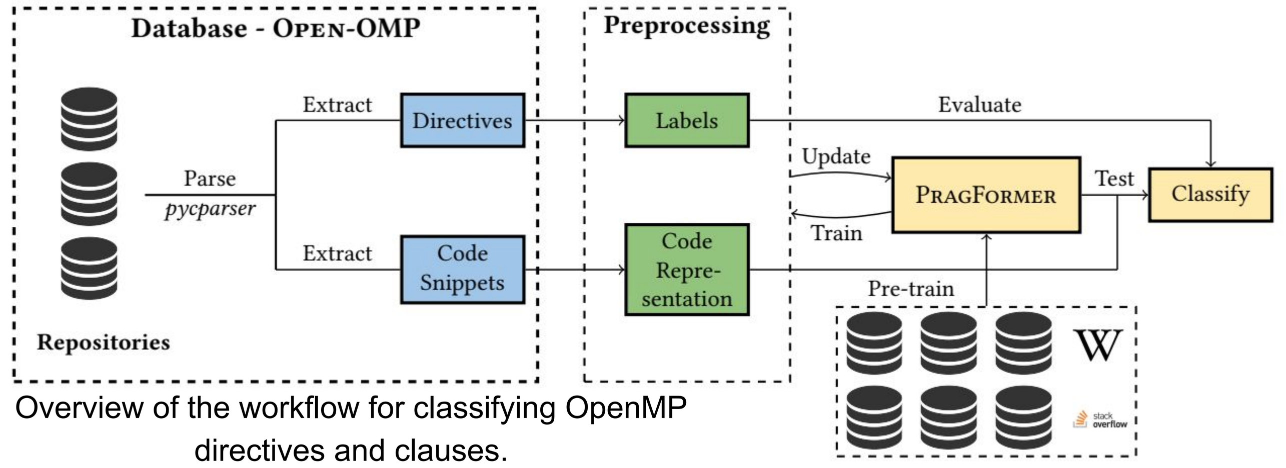
- *We propose an* ***NLP model*** *– based on a novel* ***parallel code database*** *– that will* ***suggest*** *locations in need for an* ***OpenMP directive*** *and even* ***specific clauses*** *such as private and reduction.*
- *Due to negligible inference time, the model can* ***suggest immediate on-the-fly advice*** *for the developer:*

Illustration of said model incorporated with an IDE, suggesting OpenMP directive



## *PragFormer*

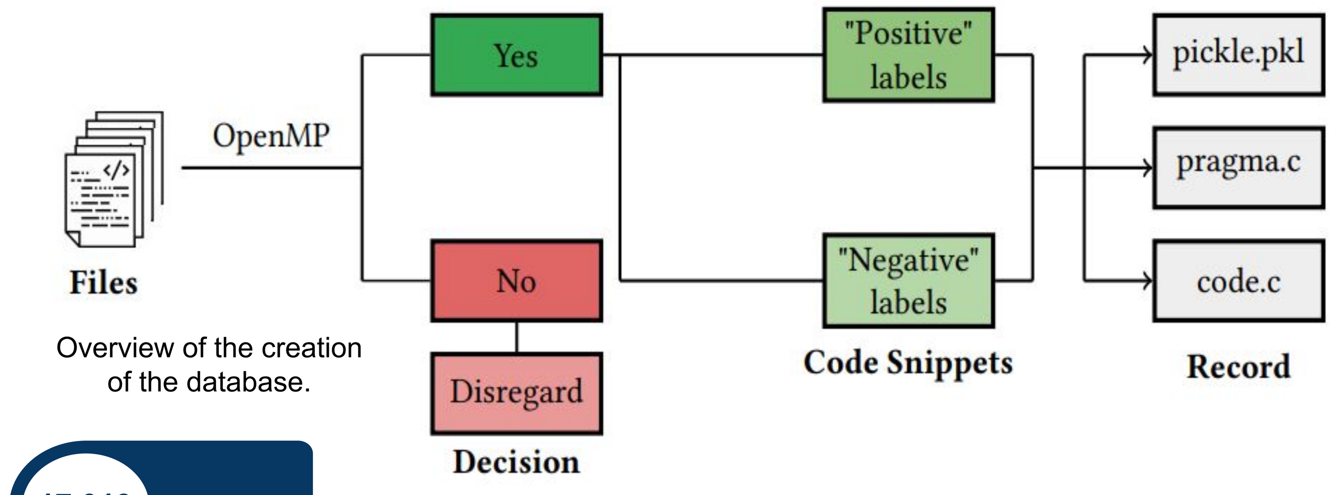
- We propose a **novel model** named ***PragFormer*** for **identifying the need for an** OpenMP directive and specific clauses based on the transformer architecture.




Overview of the workflow for classifying OpenMP directives and clauses.

## Corpus

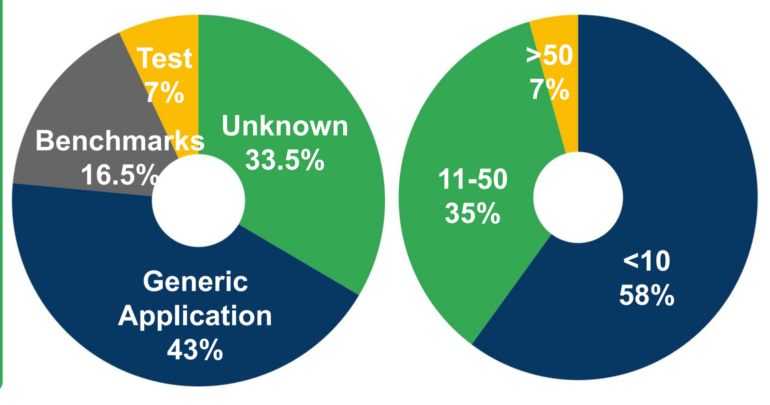
- We created a database named ***Open-OMP*** with 17,000 unique code snippets containing the for-loop, OpenMP directive (if exists), and an AST representation of the two.
- Half of the code snippets are labeled with an OpenMP directive, while the other half with a high probability, not.
- The code snippets were extracted from ***C*** **files exclusively** that were gathered with ***Github*** by searching the phrase "OpenMP".
- The data-validation-test were split in an 80%-10%-10% ratio.




Overview of the creation of the database.

**17,013**
- Query GitHub.com searching "OpenMP"
- Contains snippets with and without OpenMP directives
- Unique
- Extracted from C files exclusively

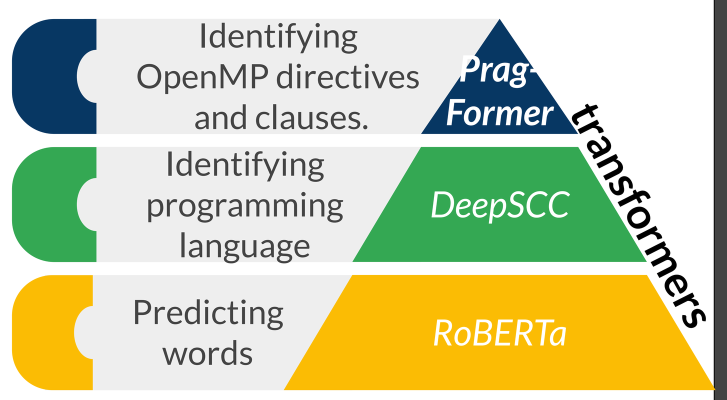
**7,613** OpenMP directives
- 3,403 *private clause*
- 1,455 *reduction clause*
- 374 *schedule dynamic*




The distribution of OpenMP snippets sources. The line length distribution.

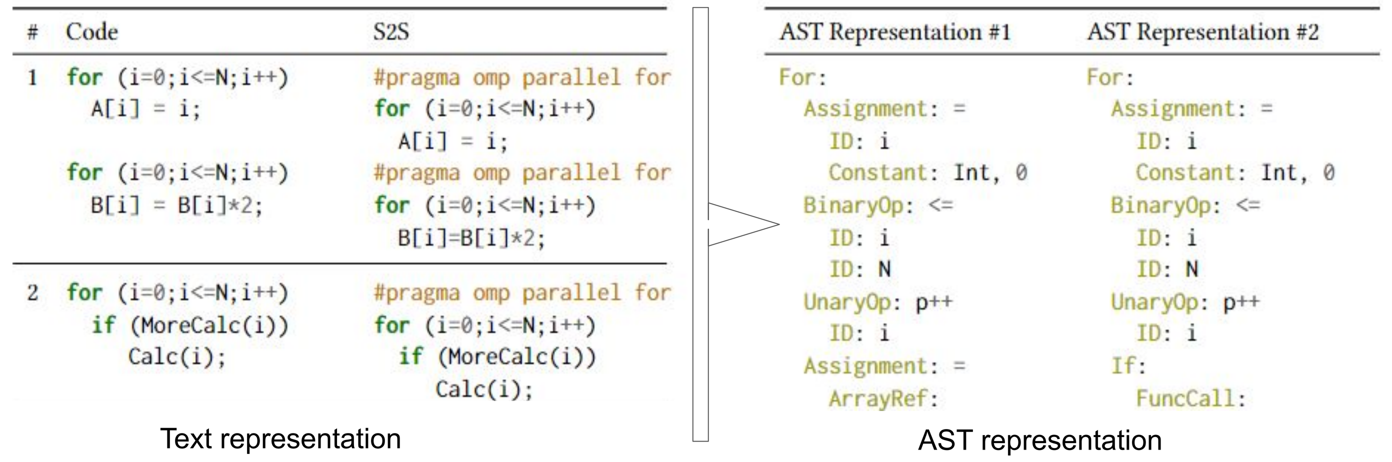
## Model

- The **self-attention mechanism** in the transformer architecture calculates a **score** for each element with respect to any other element. The score determines the **amount** of **consideration** between elements in each consecutive layer.
- Identifying OpenMP directives is mostly hinted from the **dependencies** between variables and statements. Therefore, the **self-attention mechanism is crucial** for the task at hand.
- *PragFormer* is **based** on the **pre-trained model** ***DeepSCC***, a fine-tuned ***RoBERTa*** model for source code.
- To perform the **classification**, the transformer architecture feeds its output to an FC layer that **predicts** a **binary label** through a **softmax layer**.



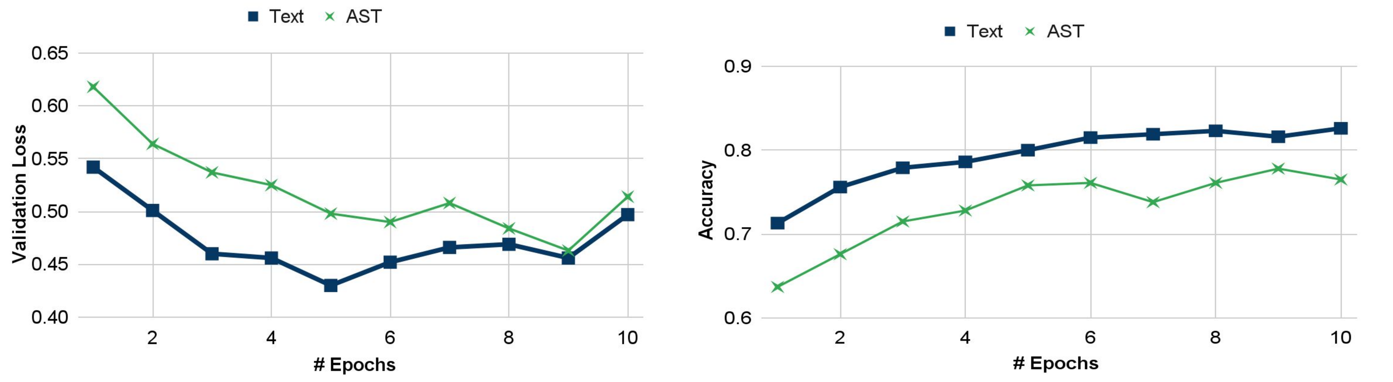

## Code Representation

- The source code is represented as a **sequence of tokens** (from a predefined vocabulary), each token is associated with a numerical vector – the vector in turn is fed to ***PragFormer***.
- We test two source code representations: natural **text** and **AST**.




## Results

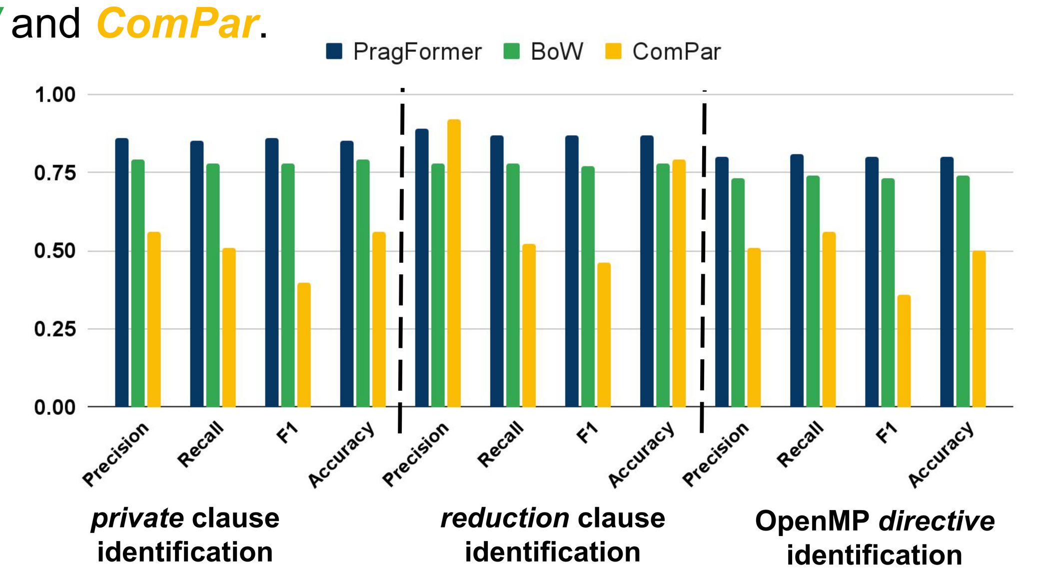
- We present the validation loss and accuracy of the **textual** and **AST representation** over the validation set:




- The **text representation** achieves the **best performance**, likely due to DeepSCC and RoBERTa **familiarity** with this representation.

## OpenMP Directive and Clause Classification

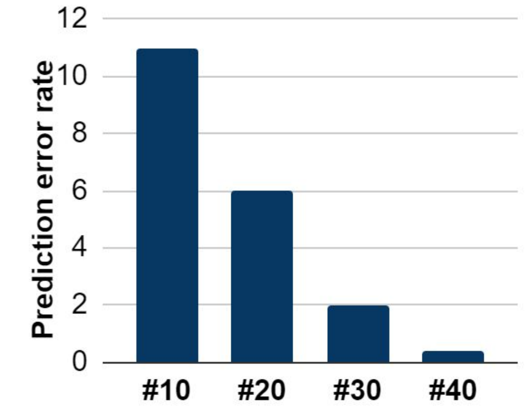
- We compare ***PragFormer*** with the text representation to the S2S compiler ***ComPar*** which incorporates three S2S compilers - *AutoPar*, *Par4All* and *Cetus* and produces their combined best results; and a simple classification model **Bag-of-Words** (**BoW**).
- For the two research questions presented, the **precision**, **recall**, **f1** and **accuracy** scores are calculated over the test for ***PragFormer***, ***BoW*** and ***ComPar***.



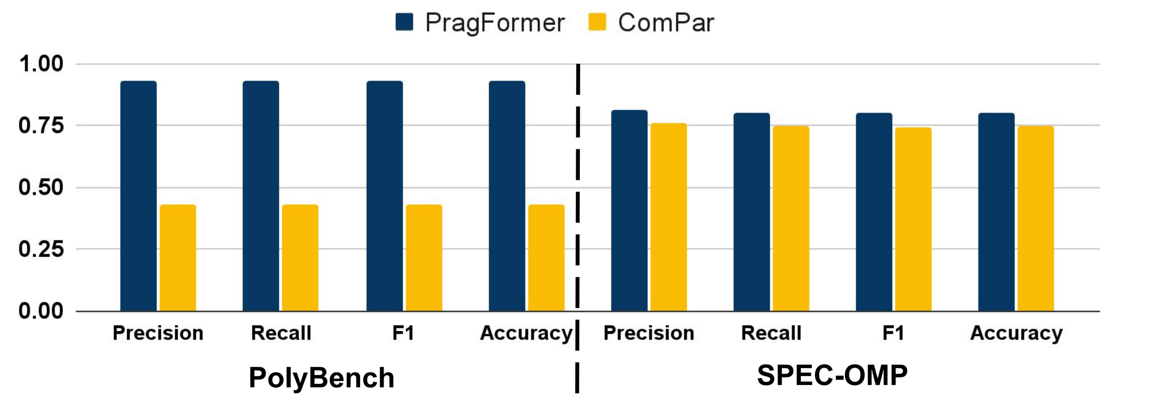

- ***PragFormer*** **achieves the best results on all four metrics on all three test cases.**

## Explainability & Benchmarks

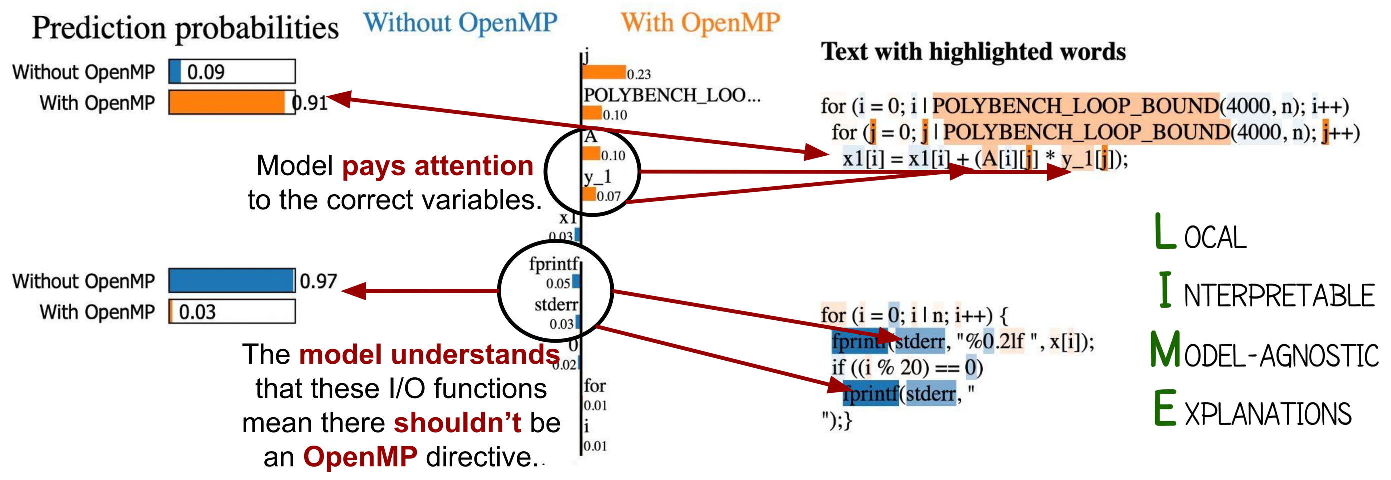
- The prediction error rate as a function of the **code length** is presented.
- Relatively, examples with length >10 and length <10 produced the same error rate of ~18%.
- It might indicate that **length doesn't** effect the **decision** of the model.




- To further test the **generality** of *PragFormer*, two benchmarks are tested: **SPEC-OMP** and **PolyBench**:



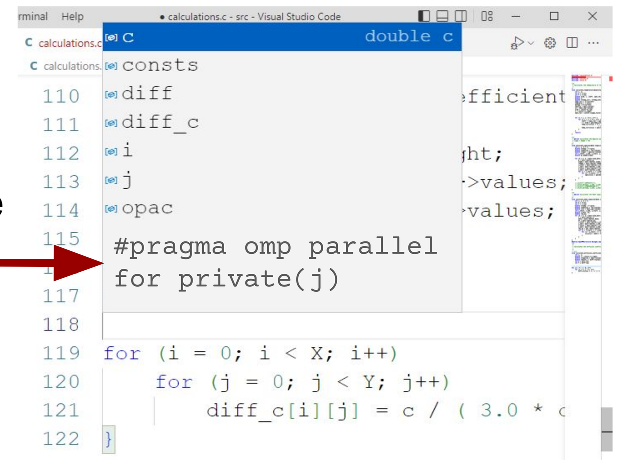

- *PragFormer* produces **excellent results** on OpenMP benchmarks.
- The following figure presents the result of the **explainability** tool **LIME** on two representative examples from the benchmarks:




- Attention to array variables and indices.
- ***fprintf*** is the main reason for predicting without OpenMP.

## Future Work

- Providing the **full context** of the source code rather the for-loop segment.
- Exploring new **source code representations** such as *IR2vec*.
- Enhancing the model to **generate** the OpenMP directive.
- Exploring other parallelization paradigms such as ***MPI*** and **heterogeneous systems**.




Acknowledgments: This work was supported by the Israeli Council for Higher Education (CHE) via the Data Science Research Center, the Lynn and William Frankel Center for Computer Science, and Intel Corporation (oneAPI Center of Excellence program). Computational support was provided by the ***NegevHPC project***.