Intro to LLMs for software engineering and a paper on code representations

The EarlyBIRD Catches the Bug: On Exploiting Early Layers of Encoder Models for More Efficient Code Classification

Anastasiia Grishina
(Angie, she/her)

Ph.D. candidate, Simula Research Laboratory, Oslo, Norway
Enrichment Student Oct’23-Mar’24, The Alan Turing Institute, London, UK

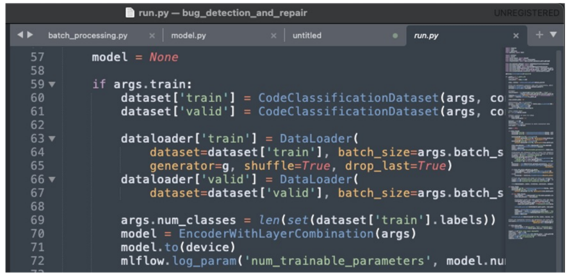
simula

Foundation Models Reading Group | The Alan Turing Institute
20 November 2023

Agenda

- About me
- Background: language models for code
  - Model types
  - Downstream tasks
  - Performance
  - Benchmarks
  - Open challenges
- Motivation for using early encoder layers
- Methodology of combining layers
- Empirical evaluation
- Conclusion and future work

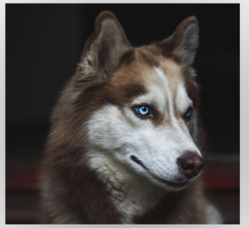
About me



PhD student in computer science,
finishing in Aug-Sep 2024

Oslo, Norway

Simula Research Laboratory
and the University of Oslo



I'm Siberian
but lived in 6 countries in the past 6 years

The Alan Turing Institute

London, UK
October 2023 – March 2024

About Simula

main activities

education

supervision
+ teaching

simula academy

researcher training
professional development
master students
internships

research

spin-outs
+ consulting

simula

scientific computing
software engineering

simulamet

communication systems
machine intelligence

Simula UiB

cryptography

simula@BI

data science

innovation

simula consulting

deep-tech consulting

150 employees;

founded in 2001;

offices in Oslo and Bergen, Norway

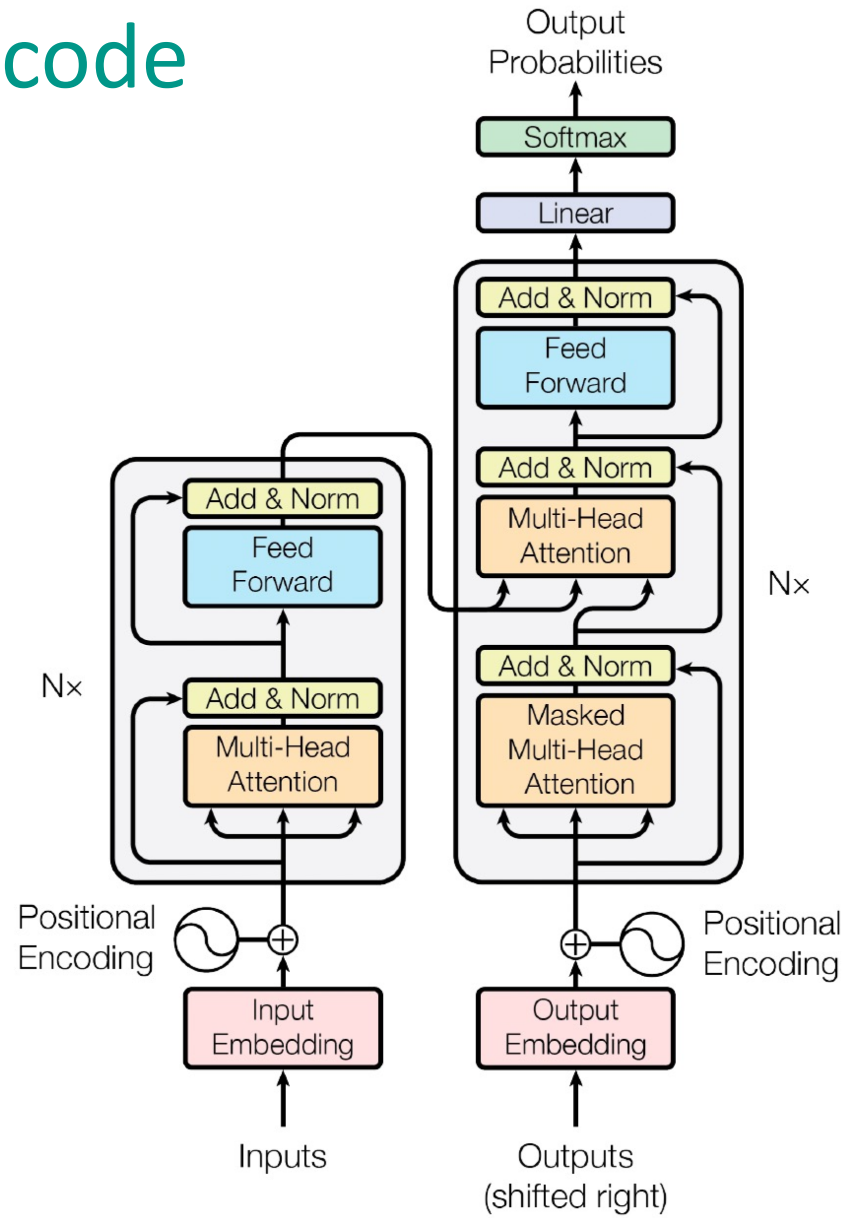
X

Overview of language models for code
I. Transformer model types

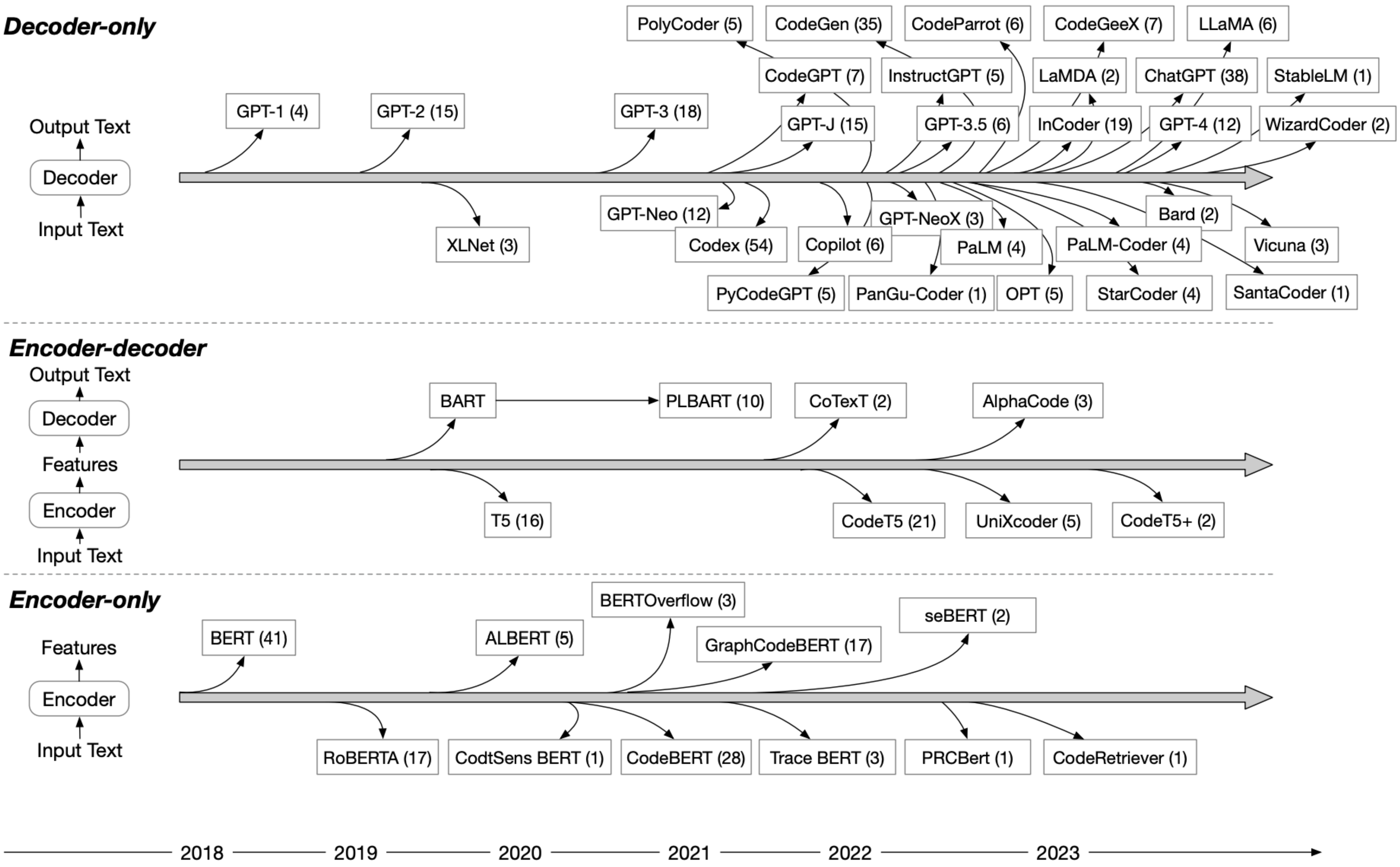
Architecture	Purpose	Example use cases	Example models
Encoder-only	Create a (low-dimensional) representation of input	Classification	NLP: BERT (Google), RoBERTa (Meta), DeBERTa (Microsoft) SE: CodeBERT, ContraBERT, StarEncoder
Encoder-decoder	Represent input sequence and generate output sequence token-by-token	Text and code representation and generation	NLP: T5 (Google) and BART (Meta) SE: CodeT5+, AlphaCode, PLBART, PYMT5
Decoder-only	Generate output sequence token-by-token based on input context and previoiusly generated tokens	Text and code generation	NLP: GPT models (OpenAI), Llama (Meta) SE: Codex (Copilot), CodeLlama, StarCoder, Incoder

Overview of language models for code
I. Transformer model types

Architecture	Purpose
Encoder-only	Create a (low-dimensional) representation of input
Encoder-decoder	Represent input sequence and generate output sequence token-by-token
Decoder-only	Generate output sequence token-by-token



Overview of language models for code



Source:
Hou, et al. 2023,
Large Language Models for Software Engineering:
A Systematic Literature Review
https://arxiv.org/pdf/2308.10620.pdf

Overview of language models for code
II. Downstream Tasks

Benchmarks

CodeXGlue[1]
(program repair, bug detection, and more)

HumanEval[2]
(code generation)

Category	Task	Dataset Name	Language	Train/Dev/Test Size	Baselines	Task definition
Code-Code	Clone Detection	BigCloneBench	Java	900K/416K/416K	CodeBERT	Predict semantic equivalence for a pair of codes.
		POJ-104	C/C++	32K/8K/12K		Retrieve semantically similar codes.
	Defect Detection	Devign	C	21k/2.7k/2.7k		Identify whether a function is vulnerable.
	Cloze Test	CT-all	Python, Java, PHP, JavaScript, Ruby, Go	-/-/176k		Tokens to be predicted come from the entire vocab.
		CT-max/min	Python, Java, PHP, JavaScript, Ruby, Go	-/-/2.6k		Tokens to be predicted come from {max, min}.
	Code Completion	PY150	Python	100k/5k/50k	CodeGPT	Predict following tokens given contexts of codes.
		GitHub Java Corpus	Java	13k/7k/8k		
	Code Repair	Bugs2Fix	Java	98K/12K/12K	Encoder-Decoder	Automatically refine codes by fixing bugs.
	Code Translation	CodeTrans	Java-C#	10K/0.5K/1K		Translate the codes from one programming language to another programming language.
Text-Code	NL Code Search	CodeSearchNet, AdvTest	Python	251K/9.6K/19K	CodeBERT	Given a natural language query as input, find semantically similar codes.
		CodeSearchNet, WebQueryTest	Python	251K/9.6K/1k		Given a pair of natural language and code, predict whether they are relevant or not.
	Text-to-Code Generation	CONCODE	Java	100K/2K/2K	CodeGPT	Given a natural language docstring/comment as input, generate a code.
Code-Text	Code Summarization	CodeSearchNet	Python, Java, PHP, JavaScript, Ruby, Go	908K/45K/53K	Encoder-Decoder	Given a code, generate its natural language docstring/comment.
Text-Text	Documentation Translation	Microsoft Docs	English-Latvian/Danish/Norwegian/Chinese	156K/4K/4K		Translate code documentation between human languages (e.g. En-Zh), intended to test low-resource multi-lingual translation.

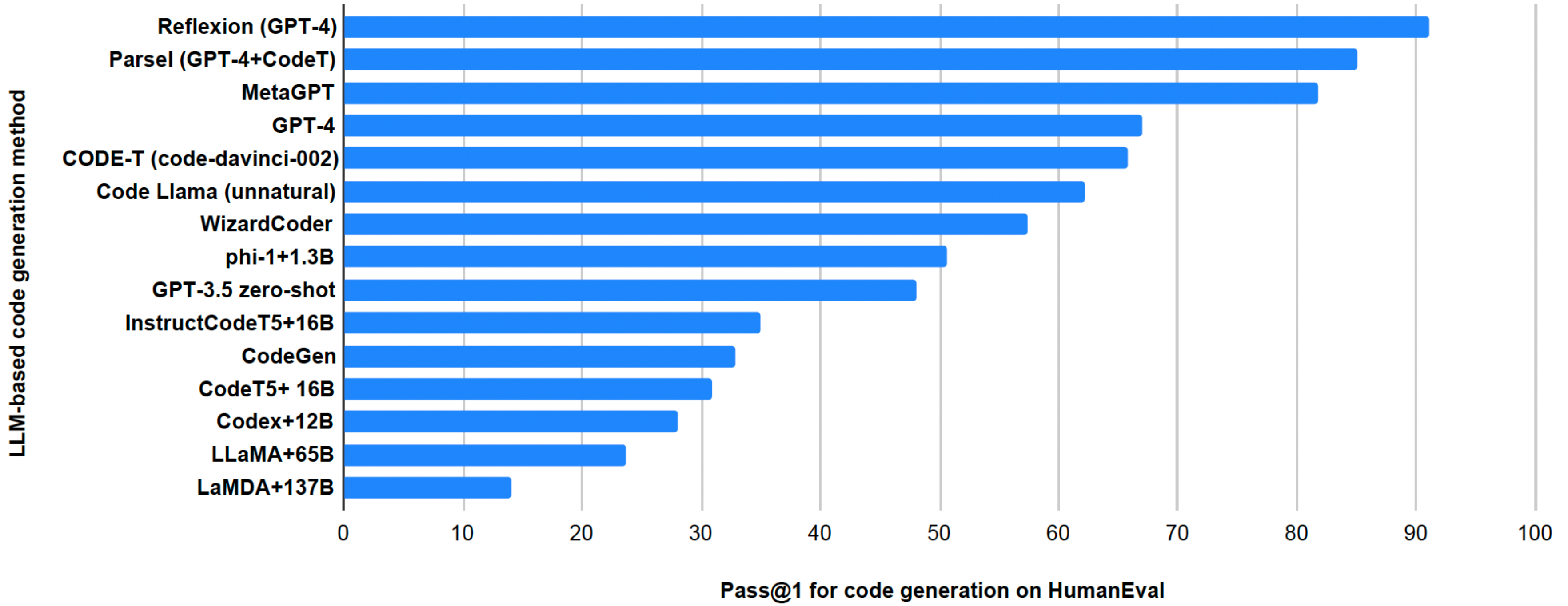
[1] S. Lu *et al.*, "CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation," in *NeurIPS*, Dec. 2021, pp. 1–16. Available: [paper] [dashboard]
[2] M. Chen *et al.*, "Evaluating Large Language Models Trained on Code." arXiv, Jul. 14, 2021. doi: 10.48550/arXiv.2107.03374.

Overview of language models for code
III. Model performance

Benchmarks

CodeXGlue[1]
(program repair,
bug detection,
and more)

HumanEval[2]
(code generation)



[Figure 4 in https://aps.arxiv.org/abs/2310.03533]

[1] S. Lu *et al.*, "CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation," in *NeurIPS*, Dec. 2021, pp. 1–16. Available: [paper] [dashboard]

[2] M. Chen *et al.*, "Evaluating Large Language Models Trained on Code." arXiv, Jul. 14, 2021. doi: 10.48550/arXiv.2107.03374.

Overview of language models for code
IV. Benchmarks

Benchmarks

CodeXGlue [paper] [dashboard] – program repair, bug detection, and more

HumanEval [HF card] – code generation

HumanEval-X [HF card] – multi-lingual code generation

Mostly Basic Python Problems (MBPP) [HF card]

Automated program repair: Defects4J, QuixBugs, ManyBugs

Overview of language models for code
V. Challenges

Tokenization: out of vocabulary words

Encoding: context and method calls from different locations

Evaluation of generated code: semantic equivalence and logic of code

Prompt engineering

Test correctness and coverage in test generation

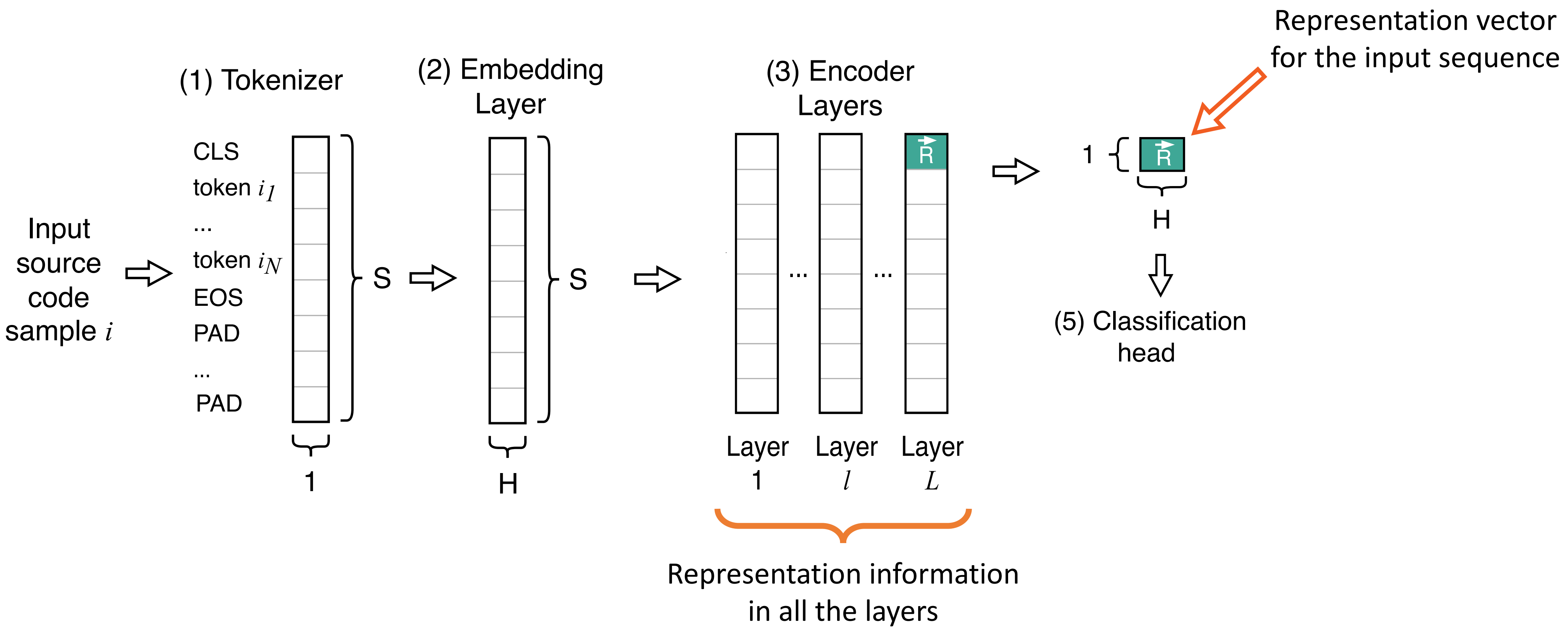
Hallucination

[1] A.Fan *et al.*, "Large Language Models for Software Engineering: Survey and Open Problems," https://aps.arxiv.org/abs/2310.03533

Overview of language models for code
I. Transformer model types

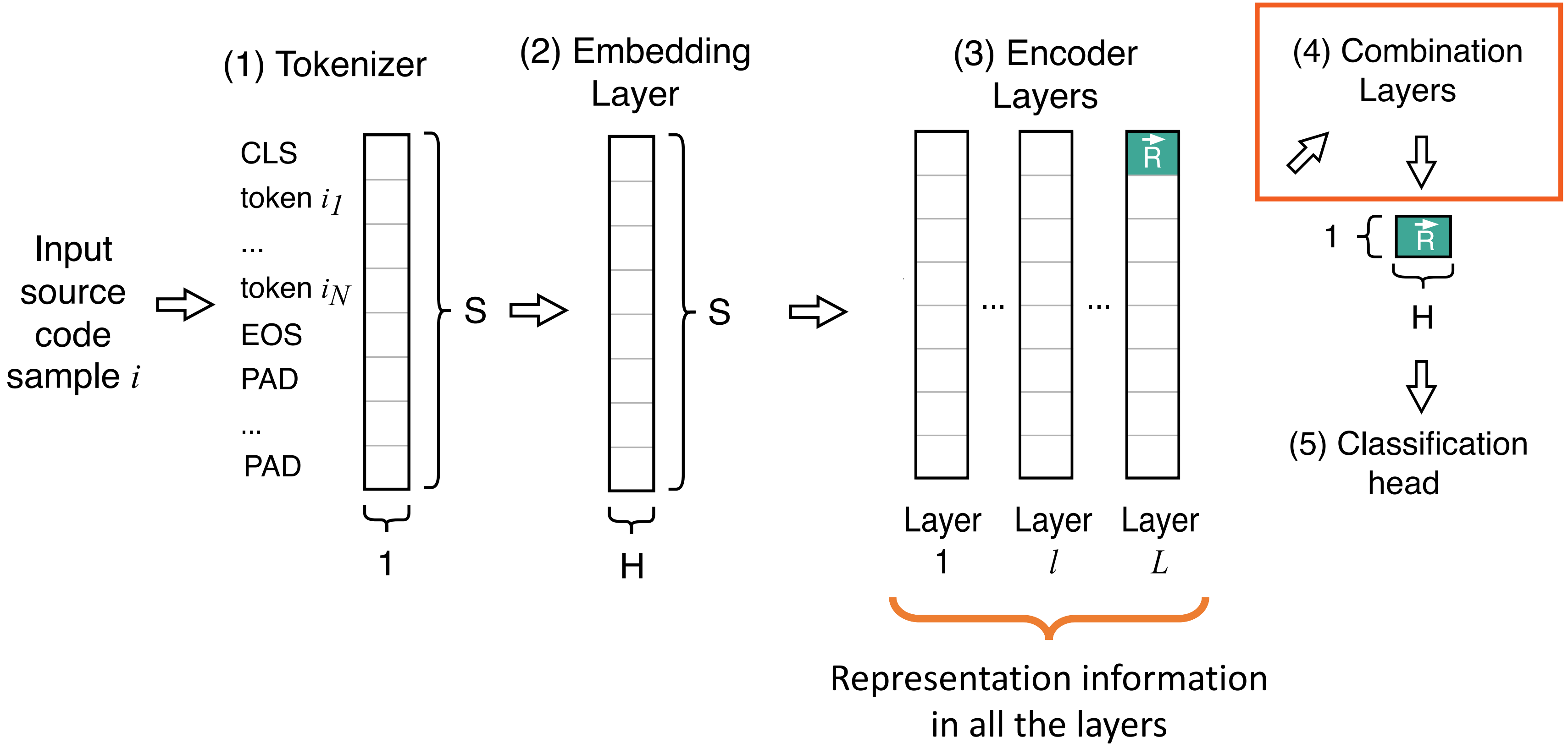
Architecture	Purpose	Example use cases	Example models
Encoder-only	Create a (low-dimensional) representation of input	Classification	NLP: BERT (Google), RoBERTa (Meta), DeBERTa (Microsoft) SE: CodeBERT, ContraBERT, StarEncoder
Encoder-decoder	Represent input sequence and generate output sequence token-by-token	Text and code representation and generation	NLP: T5 (Google) and BART (Meta) SE: CodeT5+, AlphaCode, PLBART, PYMT5
Decoder-only	Generate output sequence token-by-token based on input context and previoiusly generated tokens	Text and code generation	NLP: GPT models (OpenAI), Llama (Meta) SE: Codex (Copilot), CodeLlama, StarCoder, Incoder

Motivation for using early encoder layers



Input sequences are usually represented by the CLS (first) token of the last encoder layer

Methodology of combining encoder layers



Combination operations

max pooling
weighted sum
slicing
pruning

Performed over

layers
tokens

Methodology of combining layers

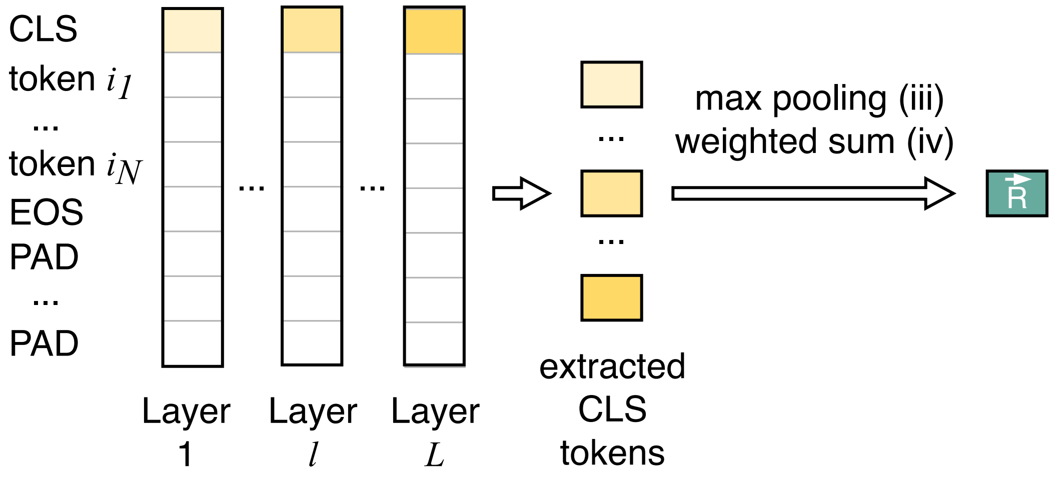
Combination operations

(2) max pooling
(2) weighted sum
(1) slicing CLS tokens
pruning

Performed over

(2) layers

tokens
(in one layer)



Methodology of combining layers

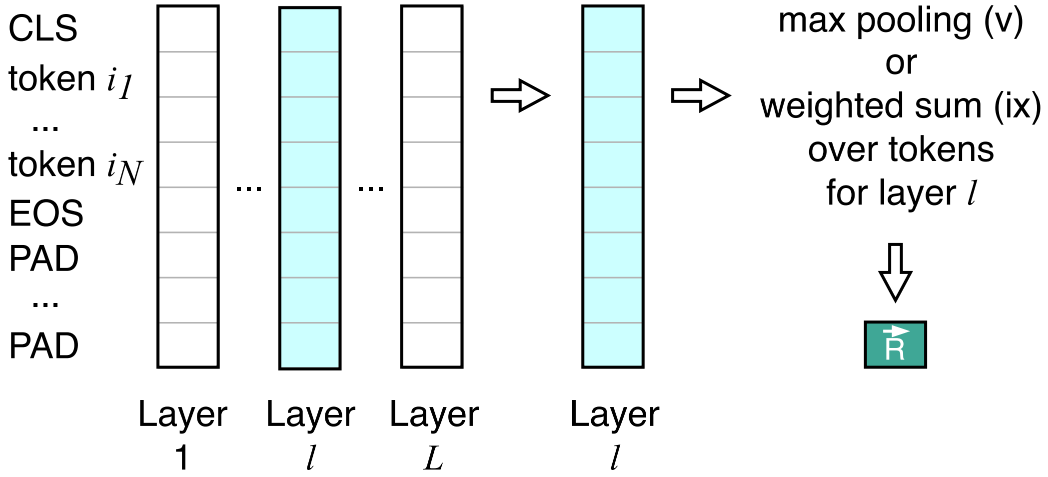
Combination operations

(2) max pooling
(2) weighted sum
(1) slicing one layer
pruning

Performed over

layers

(2) tokens
(in one layer)



Methodology of combining layers: pruning

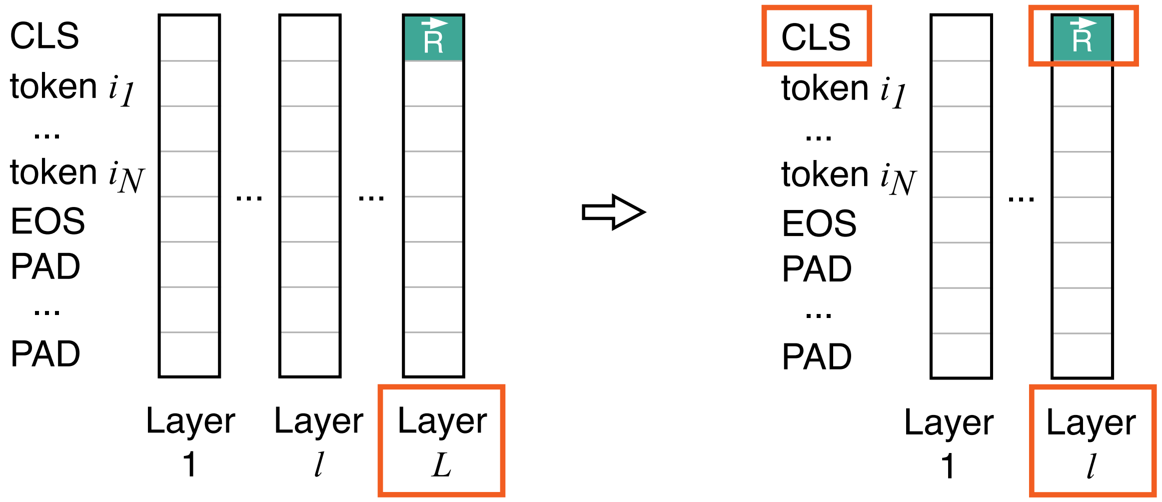
Combination operations

max pooling
weighted sum
(2) slicing CLS token
(1) pruning

Performed over

layers

(2) tokens
(in one layer)



Empirical evaluation

1. Build the model

encoder model → apply layer combination (12 different combinations) → get code representation → use classification head

2. Fine-tune the model x10 runs with different seeds

3. Evaluate classification performance x10 runs

Research objectives

RQ1. Study the performance of combined representations vs. baseline
on code classification
(same model size)

RQ2. Study the performance of code representations from pruned models
(smaller models)

Empirical evaluation setup

Model: CodeBERT

125M parameters

pre-trained on NL-PL multi-lingual data:

2.1M bimodal datapoints and 6.4M unimodal codes across six languages (Python, Java, JavaScript, PHP, Ruby, and Go)

dataset	language	# classes	avg # tokens	# code examples		
				train	valid	test
Devign	C/C++	2	614	22k	3k	2k
ReVeal	C/C++	2	512	18k	2k	2k
BIFI	Python	3	119	20k	2k	15k
Exception Type	Python	20	404	18k	2k	10k

Results

F1 weighted, difference with baseline (bsln)
* - differences that are statistically significant according to the Wilcoxon test

Devign (2 classes)

Layer	1	2	3	4	5	6	7	8	9	10	11	12
(ii) CLS token layer *l*	-2.5*	2.0*	2.4*	1.7	1.5	1.8	2.2*	1.2	0.7	0.2	0.1	bsln
(v) max pool all tokens	-1.4*	0.1	0.8	1.0	2.0*	2.3*	1.6	2.1*	2.1*	1.8	1.4	-0.4
(v) max pool code tokens	-1.4*	0.1*	0.8*	0.9*	2.0*	2.2*	1.6*	2.0*	1.7*	1.7*	1.4*	-0.1*
(ix) w sum all tokens	-5.4*	-6.0*	-4.1*	-5.2*	-6.8*	-5.2*	-5.9*	-5.5*	-5.8*	-3.9*	-4.6*	-6.6*
(ix) w sum code tokens	-5.0*	-3.7	-6.3	-6.0	-4.7*	-7.0*	-7.2	-4.1*	-5.5*	-3.8	-6.6	-10.8

ReVeal (2 classes)

Layer	1	2	3	4	5	6	7	8	9	10	11	12
(ii) CLS token layer *l*	-1.3*	0.6*	0.2	0.2	0.3	0.2	0.4*	-0.1	-0.2	0.04	0.05	bsln
(v) max pool all tokens	-1.1*	0.04	0.3	0.7*	0.4	0.1	0.2	0.3	0.2	-0.03	-0.2	-0.6*
(v) max pool code tokens	-1.1*	0.1*	0.3*	0.7*	0.4*	0.2*	0.3*	0.2*	0.2*	0.1*	-0.2*	-0.5*
(ix) w sum all tokens	-2.0*	-1.9*	-1.9*	-2.1*	-2.1*	-2.0*	-1.9*	-2.0*	-2.0*	-1.7*	-1.4*	-1.2*
(ix) w sum code tokens	-2.2*	-2.4	-1.9	-1.9*	-1.8	-2.0	-2.4	-1.7	-2.2	-1.8	-2.0	-1.5*

BIFI (3 classes)

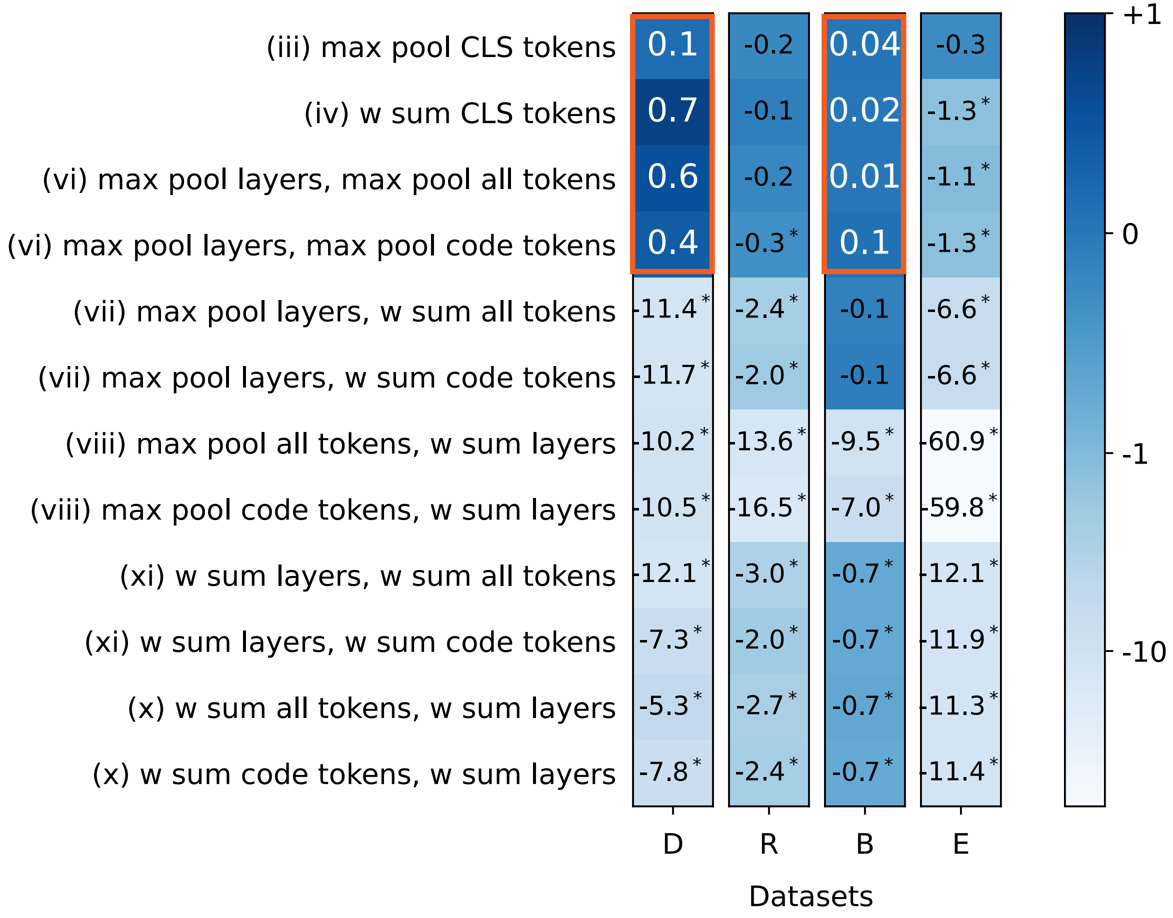
Layer	1	2	3	4	5	6	7	8	9	10	11	12
(ii) CLS token layer *l*	-27.2*	-14.8*	-11.2*	-6.8*	-2.8*	-1.8*	-1.0*	-0.7*	-0.4*	-0.2*	-0.03	bsln
(v) max pool all tokens	-26.8*	-16.3*	-11.7*	-8.3*	-3.4*	-1.6*	-1.1*	-0.8*	-0.5*	-0.2	-0.01	0.1
(v) max pool code tokens	-26.9*	-16.4*	-11.8*	-8.3*	-3.5*	-1.6*	-1.1*	-0.8*	-0.5*	-0.2*	0.01	-0.01
(ix) w sum all tokens	-28.5*	-20.1*	-15.7*	-11.5*	-7.1*	-4.2*	-1.9*	-1.5*	-1.1*	-0.8*	-0.5*	-0.2*
(ix) w sum code tokens	-28.5*	-20.1*	-15.7*	-11.6*	-7.4*	-4.4*	-1.8*	-1.3*	-1.2*	-0.7*	-0.6	-0.3

Exception Type (20 classes)

Layer	1	2	3	4	5	6	7	8	9	10	11	12
(ii) CLS token layer *l*	-30.8*	-17.2*	-12.0*	-9.0*	-7.9*	-6.5*	-4.4*	-2.9*	-2.4*	-1.8*	-1.1*	bsln
(v) max pool all tokens	-33.3*	-18.8*	-12.8*	-10.1*	-8.3*	-5.7*	-3.6*	-2.6*	-2.2*	-1.8*	-0.8*	-0.1
(v) max pool code tokens	-33.4*	-18.9*	-12.8*	-10.1*	-8.2*	-5.8*	-3.5*	-2.6*	-2.2*	-1.8*	-1.0*	0.1*
(ix) w sum all tokens	-37.8*	-32.7*	-26.7*	-23.3*	-23.3*	-15.2*	-13.8*	-10.8*	-9.6*	-9.0*	-6.8*	-1.1
(ix) w sum code tokens	-37.7*	-32.8*	-26.8*	-23.3*	-23.4*	-15.3*	-13.8*	-10.8*	-9.8*	-9.2*	-7.6*	-0.9

Results

F1 weighted, difference with baseline (bsln)
* - differences that are statistically significant according to the Wilcoxon test



RQ1. Performance of combined representations vs. baseline on code classification

(same model size)

Combination operations

max pooling
weighted sum
slicing CLS token
pruning

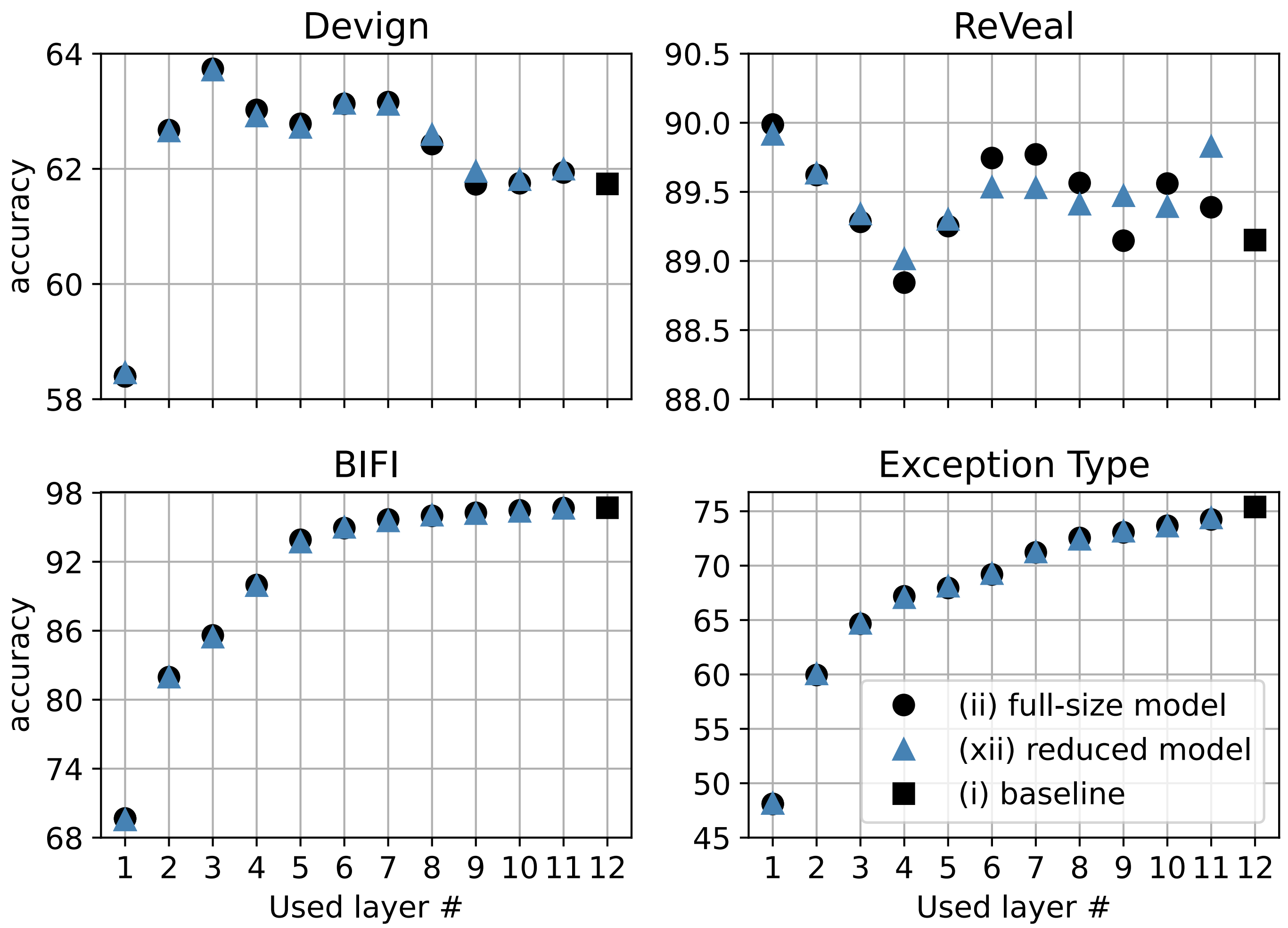
Performance over

layers

tokens

Results

Accuracy, difference with baseline (bsln)
* - differences that are statistically significant according to the Wilcoxon test



RQ2. Performance of code representations from pruned models

(smaller models)

✓ same or better performance with the pruned model vs. the full-size model for bug detection (2 classes) datasets

✓ performance vs. speed up (model size) trade-off for datasets with more classes

Results

F1 weighted, difference with baseline (bsln)
* - differences that are statistically significant according to the Wilcoxon test
large, **medium**, and **small** - effect sizes
Time – for 1 epoch of fine-tuning
Speed-up – gained by pruning layers and fine-tuning a smaller model

	Devign				ReVeal				BIFI				Exception Type			
l	Time	Speed-up	Acc	F1(w)	Time	Speed-up	Acc	F1(w)	Time	Speed-up	Acc	F1(w)	Time	Speed-up	Acc	F1(w)
12	8:50	1.0x	61.7	60.4	6:57	1.0x	89.2	88.5	8:41	1.0x	96.7	96.7	7:22	1.0x	75.4	75.3
11.0	8:03	1.1x	+0.3	-0.1*	6:56	1.0x	**+0.6**	+0.3	7:07	1.2x	-0.1*	-0.1*	6:33	1.1x	-1.0	-1.0
10.0	7:13	1.2x	+0.1	+0.3	6:20	1.1x	+0.2	-0.0*	6:22	1.4x	-0.3	-0.3	6:01	1.2x	-1.8	-1.8
9.0	6:40	1.3x	+0.3	+0.5	5:53	1.2x	+0.3	-0.1*	5:46	1.5x	-0.5	-0.5	5:29	1.3x	-2.2	-2.3
8.0	5:52	1.5x	**+0.9**	+1.1	5:15	1.3x	+0.2	-0.1*	5:13	1.7x	-0.6	-0.6	4:55	1.5x	-3.0	-3.0
7.0	5:23	1.6x	**+1.4**	**+2.2**	4:44	1.5x	+0.3	+0.2	4:43	1.8x	-1.1	-1.1	4:20	1.7x	-4.1	-4.4
6.0	4:54	1.8x	**+1.4**	**+2.0**	4:25	1.6x	+0.3	+0.1	4:10	2.1x	-1.7	-1.7	3:50	1.9x	-6.1	-6.6
5.0	4:03	2.2x	**+1.0**	**+1.5**	3:45	1.9x	+0.1	+0.2	3:31	2.5x	-3.0	-3.0	3:15	2.3x	-7.3	-7.7
4.0	3:22	2.6x	**+1.2**	**+1.6**	3:06	2.2x	-0.2*	+0.2	2:53	3.0x	-6.8	-6.8	2:41	2.7x	-8.3	-9.2
3.0	2:41	3.3x	**+2.0**	**+2.4**	2:28	2.8x	+0.1	+0.3	2:15	3.8x	-11.3	-11.3	2:10	3.4x	-10.7	-12.0
2.0	2:00	4.4x	**+1.0**	**+1.9**	1:52	3.7x	+0.4	**+0.6**	1:34	5.5x	-14.7	-14.8	1:34	4.7x	-15.4	-17.1
1.0	1:18	6.8x	-3.2	-2.3	1:13	5.7x	**+0.8**	-1.2	0:58	9.0x	-27.2	-27.3	0:59	7.4x	-27.3	-30.7

simula

Conclusion and future work

Thank you!

EarlyBIRD is an approach to combine early layers of encoder models for code representation that are further used for downstream tasks

The approach is tested on 4 datasets for code classification

Max pooling and usage of CLS tokens from the last layer (baseline) and earlier layers yield better performance for defect detection

Pruned models performed better on defect detection datasets. They reduce resource usage for multi-class settings.

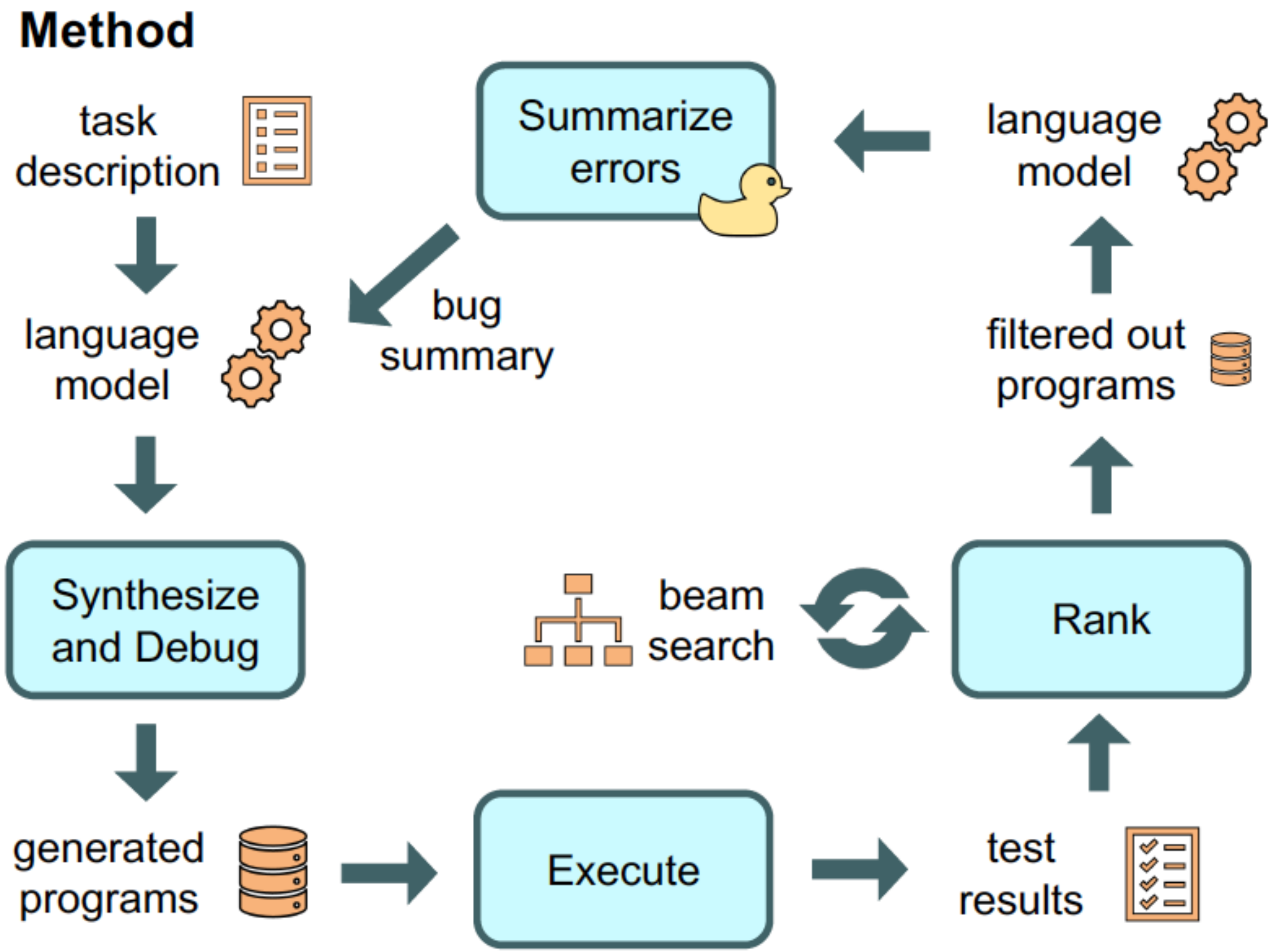
Future work involves testing on two more models, StarEncoder and ContraBERT, and possibly compare pruned models with parameter-efficient fine-tuning methods

Goal

Automate source code analysis and repair

Objectives

- Bug detection: reduce the number of layers without performance loss
- explore code repair abilities of code-to-code translation models
- build pipelines for fully autonomous programming



Wilcoxon test

The Wilcoxon test is a non-parametric test suitable for the setting in which different model variants are tested on the same test set, because it is a paired test

The Wilcoxon test checks the null hypothesis whether two related paired samples come from the same distribution.

We reject the null hypothesis if p-value is less than $\alpha = 0.05$.

Frank Wilcoxon. 1992. Individual Comparisons by Ranking Methods. In Breakthroughs in Statistics: Methodology and Distribution, Samuel Kotz and Norman L. Johnson (Eds.). Springer, New York, NY, 196–202. https://doi.org/10.1007/978-1-4612-4380-9_16

A12 statistics

```
The A12 statistics measures the
probability that running algorithm X
yields higher performance than running
another algorithm Y.

A12 = #(X > Y)/mn + 0.5*#(X=Y)/mn,

m – number of runs of algorithm X;
n – number of runs of algorithm Y.
```

For the pruned models, we compute Vargha and Delaney's $A12$ non-parametric effect size measure of the performance change for accuracy and F1(w) with thresholds of 0.71, 0.64 and 0.56 for large, medium and small effect sizes

András Vargha and Harold D. Delaney. 2000. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. Journal of Educational and Behavioral Statistics 25, 2 (June 2000), 101–132. https://doi.org/10.3102/10769986025002101

Pseudo-code: [github-gist]

Dataset details

Devign
Balanced (54% non-vulnerable functions), C/C++, vulnerability detection, 2 classes

ReVeal
Not balanced (90% non-vulnerable functions), C/C++, vulnerability detection, 2 classes

BIFI
Python, syntax error classification, 3 classes: Unbalanced Parentheses (43%), Indentation Error (31%), Invalid Syntax (26%)

Exception Type
Python, exception type prediction, 20 classes

	train	valid	test
ValueError	3417	415	2016
KeyError	3384	362	1926
AttributeError	2444	274	1372
TypeError	1564	156	809
OSError	1396	131	779
IOError	1318	136	721
ImportError	1180	170	690
IndexError	1035	139	586
KeyboardInterrupt	509	58	232
StopIteration	432	61	270
AssertionError	323	29	155
RuntimeError	247	34	107
NotImplementedError	206	24	119
SystemExit	200	16	105
ObjectDoesNotExist	197	16	95
UnicodeDecodeError	196	21	134
NameError	166	19	78
ValidationError	159	16	92
HTTPError	104	9	55
DoesNotExist	3	2	7
Total	18480	2088	10348

State-of-the-art results on selected benchmarks

CodeXGLUE

Defect Detection (Code-Code)

Rank	Model	Organization	Date	Accuracy
1	UniXcoder-nine-MLP	Academy of Milit...	2023-05-22	69.29
2	CoTexT	Case Western R...	2021-04-23	66.62
3	C-BERT	AI4VA (IBM Res...	2021-03-19	65.45

(c) Scenario-B: Using Retrained Models with Real-world Data.

Dataset	Input	Approach	Acc	Prec	Recall	F1
REVEAL dataset	Token	Russell *et al.*	90.98 (0.75)	24.63 (5.35)	10.91 (2.47)	15.24 (2.74)
	Slice + Token	VulDeePecker	89.05 (0.80)	17.68 (7.51)	13.87 (8.53)	15.7 (6.41)
		SySeVR	84.22 (2.48)	24.46 (4.85)	40.11 (4.71)	30.25 (2.35)
	Graph	Devign	88.41 (0.66)	34.61 (3.24)	26.67 (6.01)	29.87 (4.34)

Learning and Evaluating Contextual Embedding of Source Code

	Setting		Misuse	Operator	Operand	Docstring	Exception
BiLSTM (100 epochs)	From scratch		76.29 %	83.65 %	88.07 %	76.01 %	52.79 %
	CBOW	ns	80.33 %	**86.82 %**	89.80 %	**89.08 %**	**67.01 %**
		hs	78.00 %	85.85 %	**90.14 %**	87.69 %	60.31 %
	Skipgram	ns	77.06 %	85.14 %	89.31 %	83.81 %	60.07 %
		hs	**80.53 %**	86.34 %	89.75 %	88.80 %	65.06 %
CuBERT	2 epochs		94.04 %	89.90 %	92.20 %	97.21 %	61.04 %
	10 epochs		95.14 %	92.15 %	93.62 %	98.08 %	77.97 %
	20 epochs		95.21 %	92.46 %	93.36 %	98.09 %	79.12 %
Transformer	100 epochs		78.28 %	76.55 %	87.83 %	91.02 %	49.56 %

BIFI: repair accuracy (not classification) ~90