EMNLP 2021

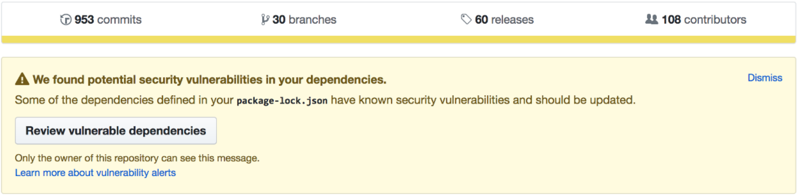
parasj.github.io/contracode

Contrastive Code Representation Learning

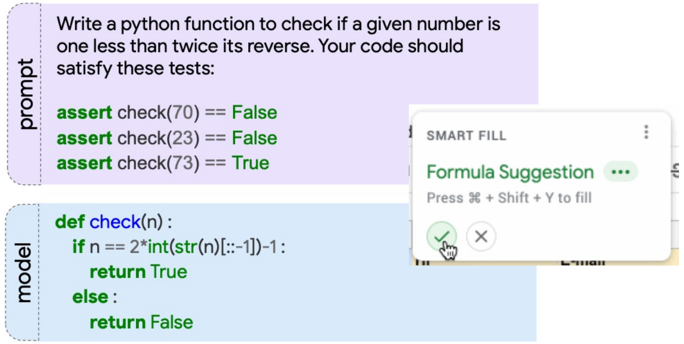
Paras Jain*, Ajay Jain*, Tianjun Zhang,
Pieter Abbeel, Joseph E. Gonzalez, Ion Stoica

UC Berkeley RISELab

Machine-aided programming tools are ubiquitous

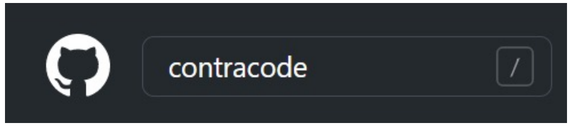


Security analysis



Program synthesis

Autocompletion



Code search

SOTA tools are increasingly learning based

Deep code search

CodeSearchNet [Husain et al. 2019]

CodeBERT [Feng et al. 2020]

Code completion

[Raychev et al. 2014]

[Svyatkovskiy et al. 2019]

OpenAI Codex [Chen et al. 2021]

Program synthesis

RobustFill [Devlin et al. 2016]

AutoPandas [Bavishi et al. 2019]

DreamCoder [Ellis et al. 2020]

[Austin et al. 2021]

Automated bug fixing

GenProg [Goues et al. 2011]

Getafix [Bader et al. 2019]

CuBERT [Kanade et al. 2019]

DrRepair [Yasunaga and Liang 2020]

Code summarization

[Allamanis et al. 2016]

[Iyer et al. 2016]

[Fernandes et al. 2018]

Current model: pretrain language models over GitHub

GitHub

```
function chunkify (str, size) {
  const chunks = []
```

```
export function shuffle(array) {
  let currentIndex = array.length;
```

```
export function knapSack(capacity, weights, values, n) {
  if (n === 0 || capacity === 0) {
    return 0;
  }
  if (weights[n - 1] > capacity) {
    return knapSack(capacity, weights, values, n - 1);
  }
  const a = values[n - 1] + knapSack(capacity - weights[n - 1], weights, values, n - 1);
  const b = knapSack(capacity, weights, values, n - 1);
  return a > b ? a : b;
}
```

Flatten program as text →

BERT
(Devlin et al. 2018)

RoBERTa
(Liu et al. 2019)

OpenAI GPT-3
(Brown et al. 2020)

OpenAI Codex
(Chen et al. 2021)

Challenge: many ways to express single program!

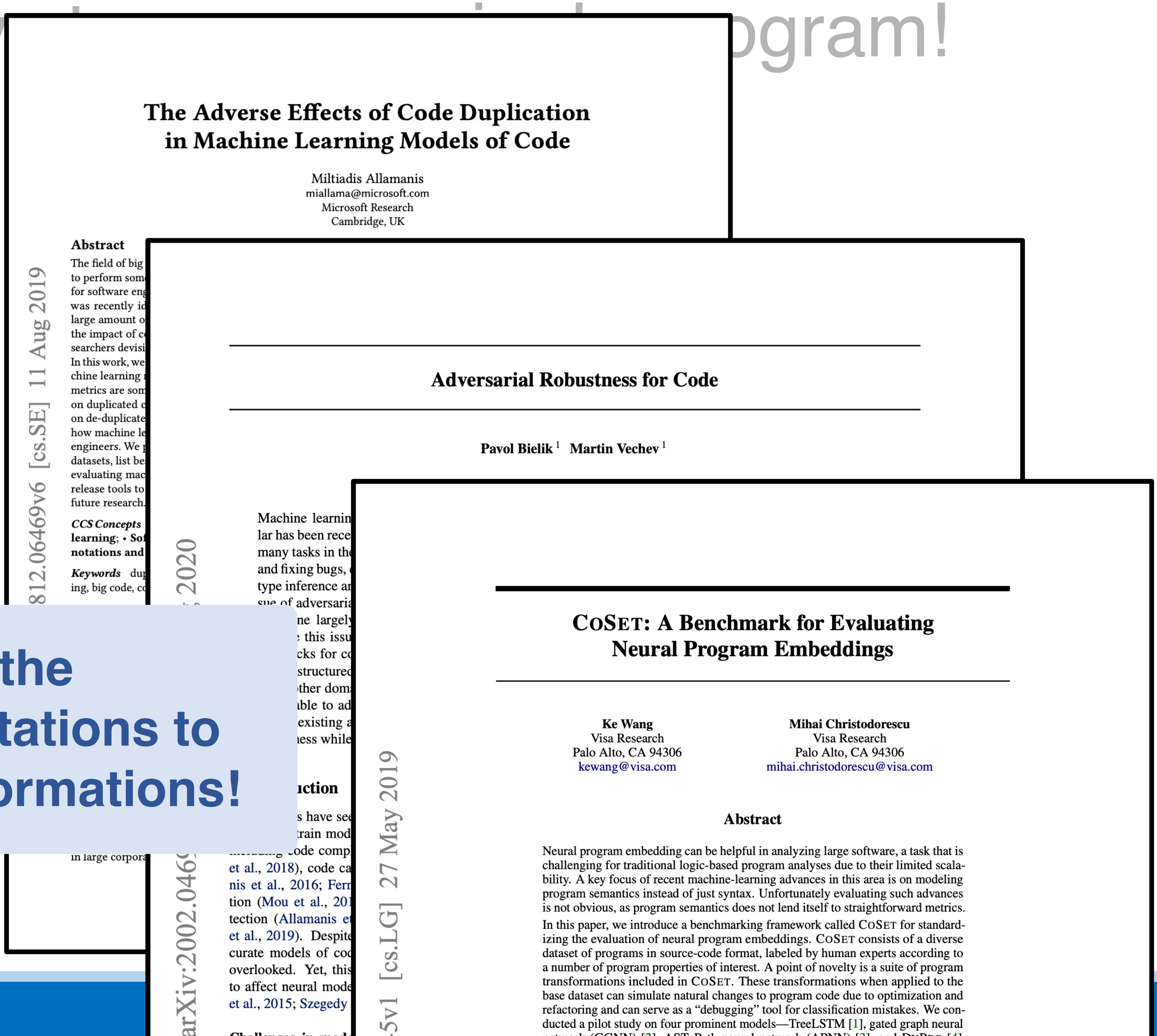
```
function x(maxLine) {
  const section = {
    text: '',
    data
  };

  for (; i < maxLine; i += 1) {
    section.text += `${lines[i]}\n`;
  }

  if (section) {
    parsingCtx.sections.push(section);
  }
}
```

Original JavaScript method

A single block of code can have **many possible rewrites w/ equivalent semantics**!

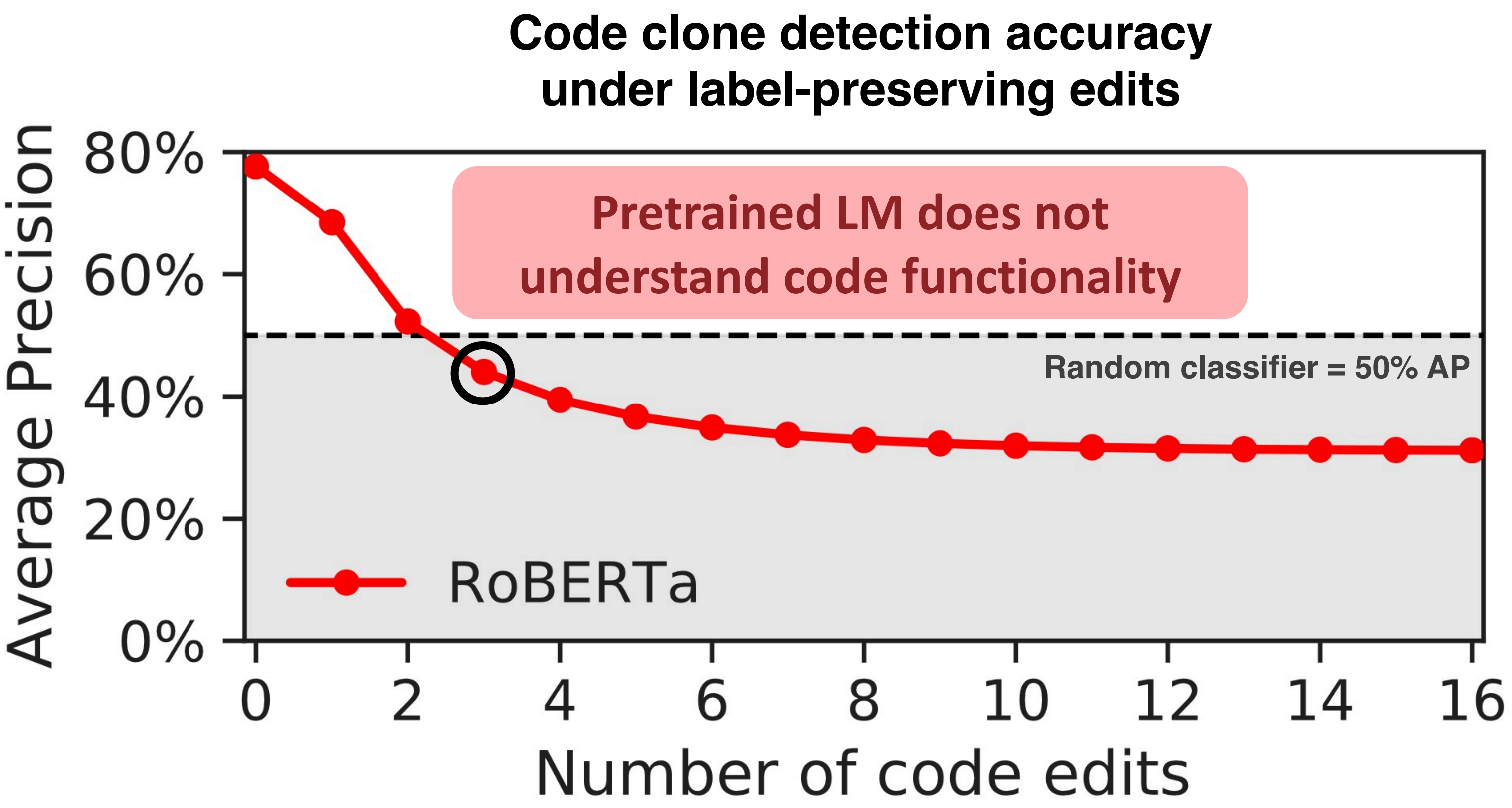
Challenge: many way ogram!

```
function x(maxLine) {
  const section = {
    text: '',
    data
  };

  for (; i < maxLine; i += 1) {
    section.text += `${lines[i]}\n`;
  }

  if (section) {
    parsingCtx.sections.push(section);
  }
}
```



Prior work discusses the sensitivity of code representations to semantics-preserving transformations!

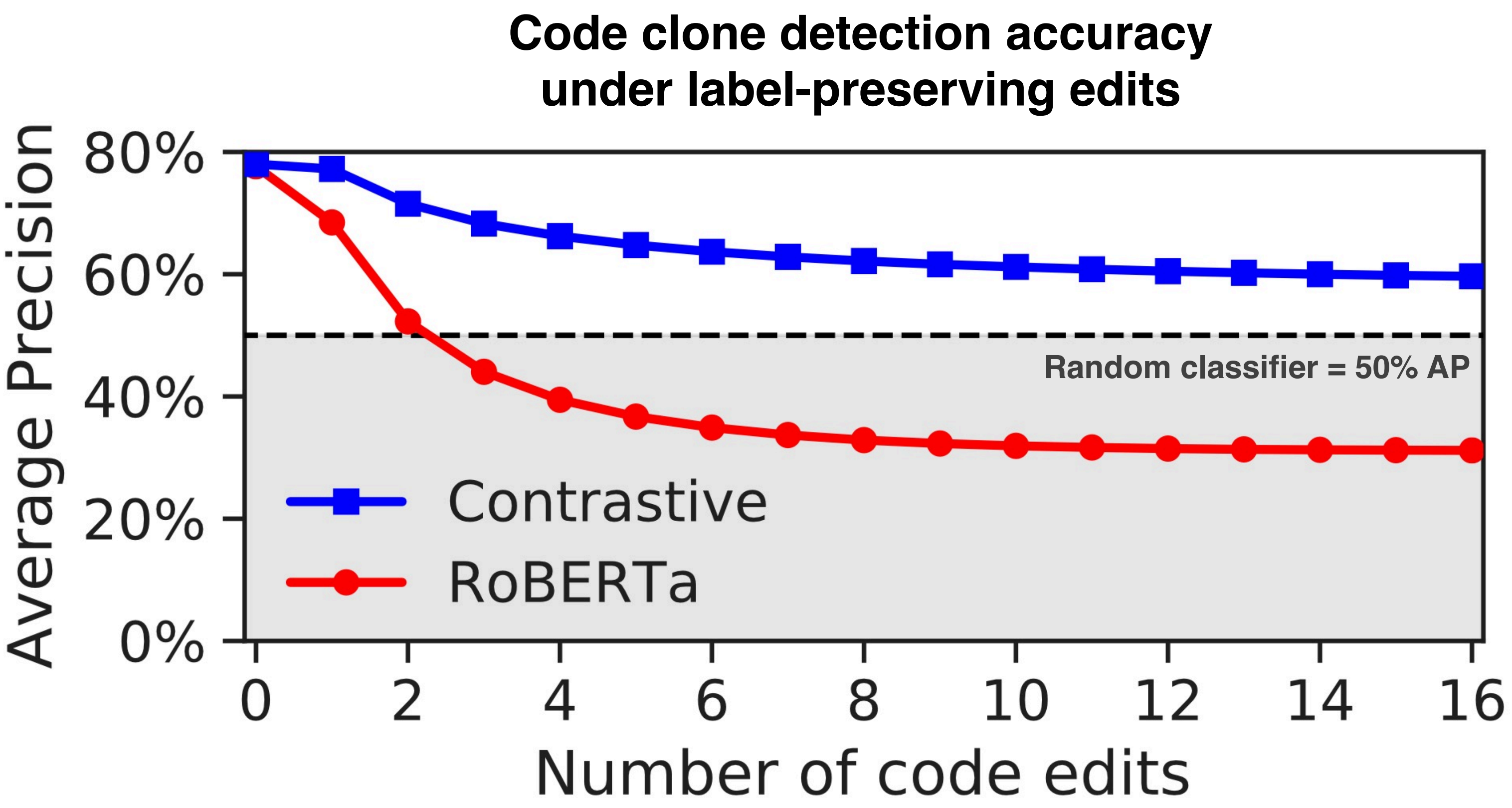
Challenge: pre-trained language models are <u>not robust</u> to code transforms!



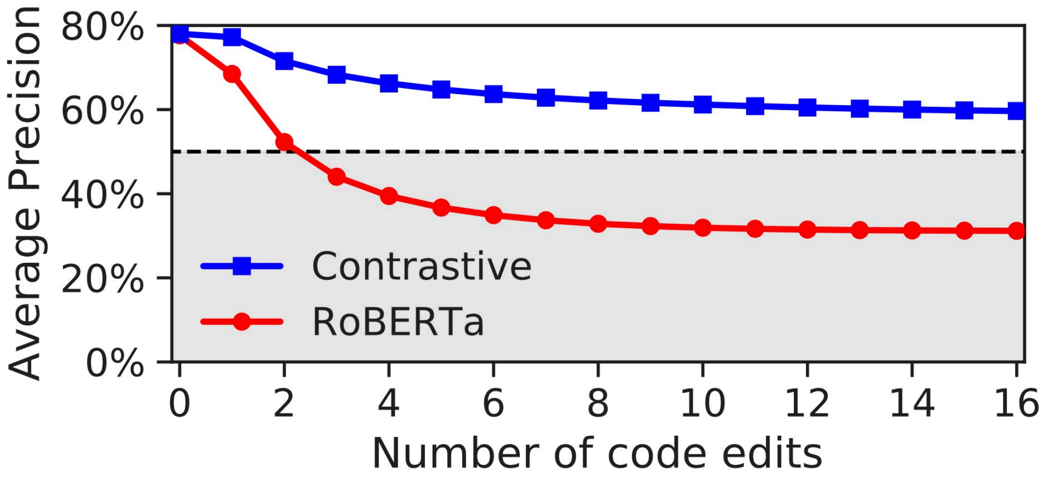
Challenge: pre-trained language models are not robust to code transforms!

RQ: How to learn robust representations of code functionality?

Key Result: ContraCode learns robust representations of source code.



Key Result: Robust representations translate to improved accuracy on natural code tasks.



Adversarial Code Clone
+29% AP, +54% AUROC

Type prediction
+2.4% acc@1, +2.8% acc@5

Code summarization
+0.38 F1

Zero-shot Code Clone
+3.8% AP, +5.4% AUROC

What makes a good code representation?

Programs with the **same functionality** should have **similar representations**!

Given a program,

```
function (len) {
  for (i = 0; i < len, i++) {
    ...
  }
}
```

Data augmentations from NLP are not effective for programs

Given a program,

```
function (len) {
  for (i = 0; i < len, i++) {
    ...
  }
}
```

Maximize similarity with equivalent programs

```
function (n) { while (i < n) { ... } }
```

Synonym substitution

```
purpose (len) {
  whereas (i = 0; i < len; i++) {
    ...
  }
}
```

Error injection

```
unction (len( {
  for (z + 5; i < len; i+=) {
    ...
  {
}
```

Word shuffling

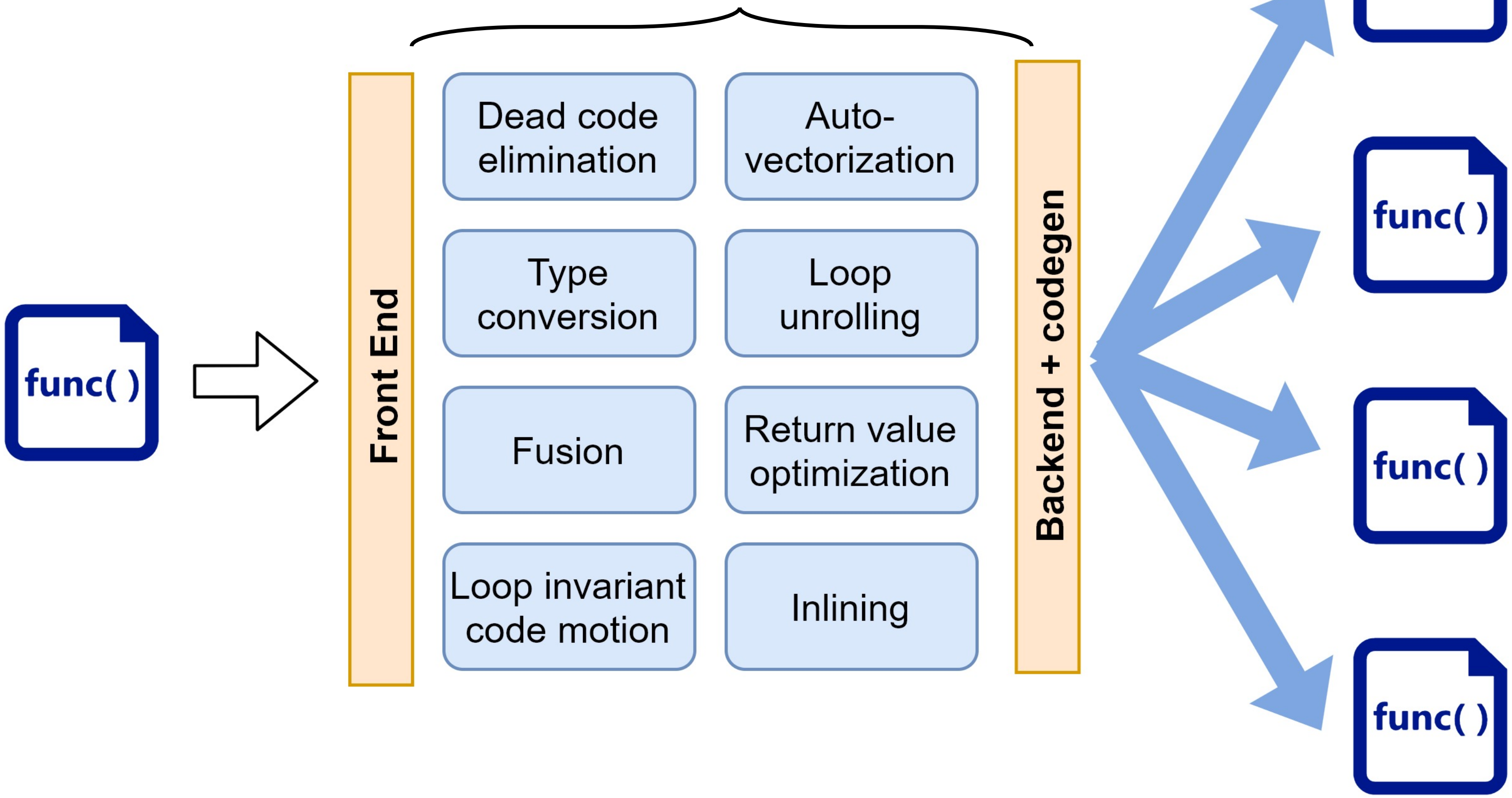
```
for {
 (i   i
    ...
  =  0; function i++) {
  (len) < } len;
```

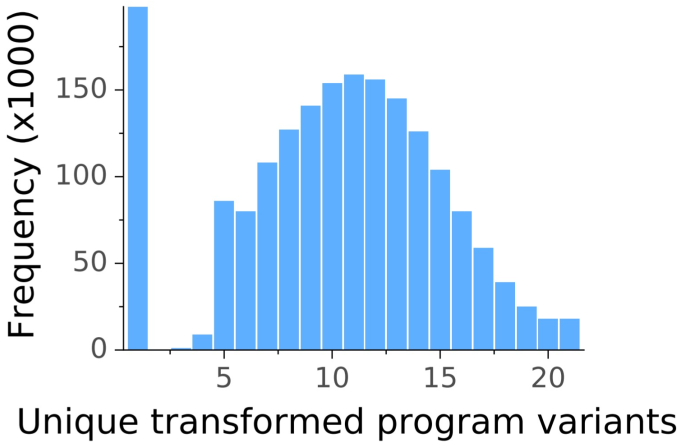
NLP data augmentations do not produce valid programs

Where to get functionally equivalent programs? Use a **compiler**

	Code compression		Identifier modification
✓	Reformatting (R)	✓	Variable renaming (VR)
✓	Beautification (B)	✓	Identifier mangling (IM)
✓	Compression (C)		**Regularization**
✓	Dead-code elimination (DCE)	✓	Dead-code insertion (DCI)
✓	Type upconversion (T)	✓	Subword regularization (SW)
✓	Constant folding (CF)	✗	Line subsampling (LS)

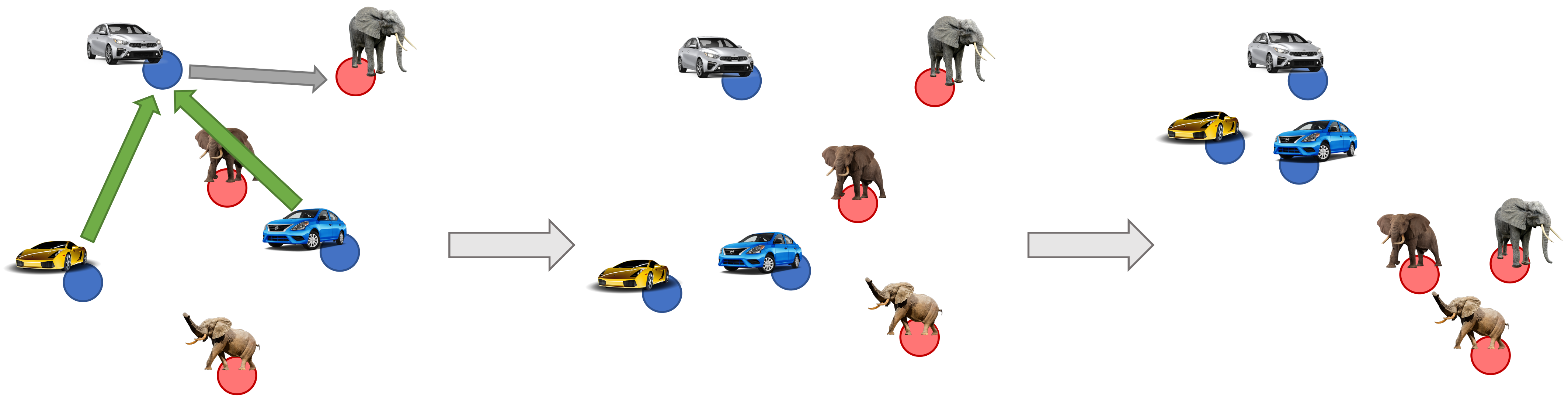
✓ = semantics-preserving transformation ✗ = lossy transformation





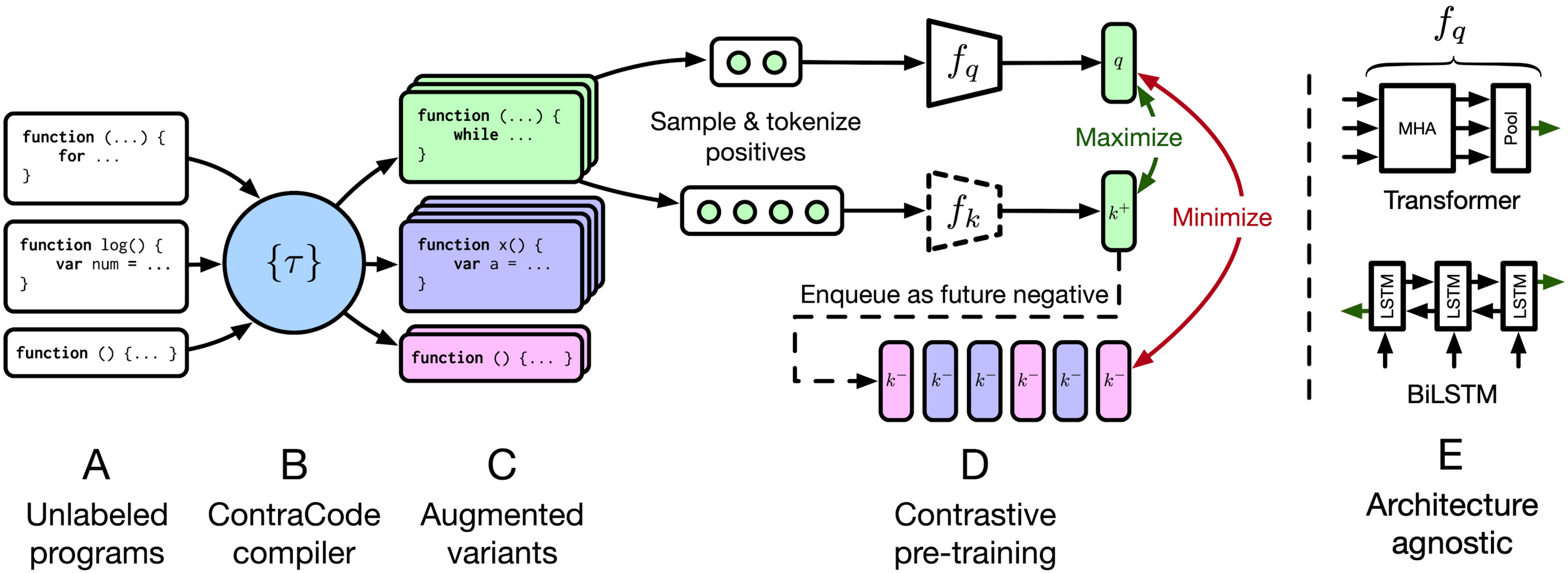
Contrastive learning

Pull similar pairs together, push dissimilar pairs apart



“Dimensionality Reduction by Learning an Invariant Mapping” Hadsell et al.
“Representation Learning with Contrastive Predictive Coding” van der Oord et al.
“A Simple Framework for Contrastive Learning of Visual Representations” Chen et al.

Contrastive Code Representation Learning



Evaluation: Type inference for TypeScript

(1) Type inference using the DeepTyper dataset (Hellendoorn et al. 2018)

```
import {
  write,
  categories,
  messageType
} from "s";
export const animationsTraceCategory = "s";
export const rendererTraceCategory = "s";
export const viewUtilCategory = "s";
export const routerTraceCategory = "s";
export const routeReuseStrategyTraceCategory = "s";
export const listViewTraceCategory = "s";
export function animationsLog ( message: string 100.0% ): void 99.9% {
  write(message, animationsTraceCategory);
}
export function rendererLog (msg): void 53.7% {
  write(msg, rendererTraceCategory);
}
export
  write
}
export
  write
}
export
  write(message, routerTraceCategory);
}
export function routeReuseStrategyLog ( message: string 99.8% ): void 99.98% {
  write(message, routeReuseStrategyTraceCategory);
}
export function styleError ( message: string 99.97% ): void 100.0% {
  write(message, categories.Style, messageType.error);
}
export function listViewLog ( message: string 100.0% ): void 100.0% {
  write(message, listViewTraceCategory);
}
export function listViewError ( message: string 99.93% ): void 100.0% ...
```

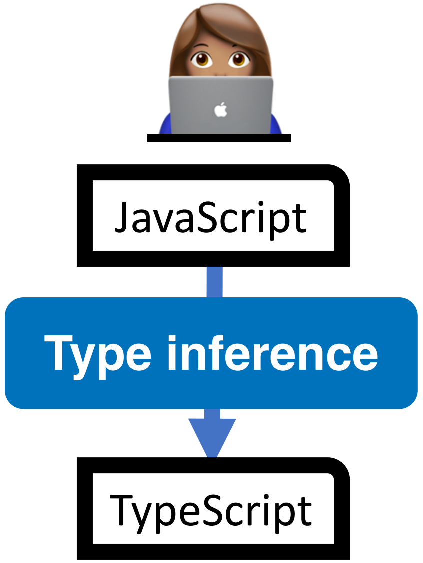
```
export function animationsLog ( message: string 100.0% ): void 99.9%
  write(message, animationsTraceCategory);
}
```

JavaScript

Type inference

TypeScript

Evaluation: Type inference for TypeScript



(1) Type inference using the DeepTyper dataset (Hellendoorn et al. 2018)

Method	Acc@1	Acc@5
TypeScript CheckJS	45.11%	—
DeepTyper, variable name only	28.94%	70.07%
GPT-3 Codex (zero-shot, 175B)	26.62%	—
GPT-3 Codex (few-shot, 175B)	30.55%	—
Transformer	45.66%	80.08%
+ RoBERTa MLM pre-train	40.85%	75.76%
DeepTyper BiLSTM	51.73%	82.71%
+ RoBERTa MLM pre-train	50.24%	82.85%

Contrastive pre-train

Hybrid pre-train

+ 1.5% top-1

Contrastive pre-train

+ 2.3% top-1

+ 8.9% vs static analysis

Evaluation: Code summarization

(2) Extreme code summarization (Alon et al 2019)

```
function (oFormElement) {
  var xhr = new XMLHttpRequest();
  xhr.onload = function(){ alert (xhr.responseText); }
  xhr.onerror = function(){ alert (xhr.responseText); }
  xhr.open (oFormElement.method, oFormElement.action, true);
  xhr.send (new FormData (oFormElement));
  return false;
}
```

→ **“Submit Form”**

Evaluation: Code summarization

(2) Extreme code summarization (Alon et al 2019)

Method	**Precision**	**Recall**	**F1**
code2vec	10.78%	8.24%	9.34%
code2seq	12.17%	7.65%	9.39%
RoBERTa MLM	15.13%	11.47%	12.45%
Transformer	18.11%	15.78%	16.86%

Contrastive pre-train

Outperforms AST-based models, from-scratch Transformer, MLM pre-training

Evaluation: Zero-shot code clone detection

(3) Detect whether two student programs are equivalent

We evaluate using linear probe + cosine similarity

Solved same problem?

```
function processData(input) {
    var parse_fun = function (s) { return parseInt(s, 10); };

    var lines = input.split('\n');
    var A = parse_fun(lines[0]);
    var B = parse_fun(lines[1])

    console.log(A + B);
}


process.stdin.resume();
process.stdin.setEncoding("ascii");
var _input = "";
process.stdin.on("data", function (input) { _input += input; });
process.stdin.on("end", function () { processData(_input); });
```

```
(function() {
  var input;

  process.stdin.setEncoding('ascii');

  input = "";

  var sum = function(a,b){return a+b}

  process.stdin.on('data', function(data) {
    if (data === "\n")
        process.stdin.emit("end");
    input += data;
  });

  process.stdin.on('end', function() {
    var sum = input.split("\n").reduce(function(a,b){return (+a)+(+b)});
    process.stdout.write(sum);
    process.exit(0);
  });
}).call(global);
```

Evaluation: Zero-shot code clone detection

(3) Detect whether two student programs are equivalent

We evaluate using linear probe + cosine similarity

Solved same problem?

	Natural code	
	AUROC	AP
Edit distance heuristic	$69.55_{\pm 0.81}$	73.75
Randomly initialized Transformer	$72.31_{\pm 0.79}$	75.82
+ RoBERTa MLM pre-train	$74.04_{\pm 0.77}$	77.65

RoBERTa MLM pre-train
+ 1.8% AP

Contrastive pre-train
+ 2.2% AP

Hybrid pre-train
+ 5.56% AP

Evaluation: Zero-shot code clone detection

(3) Detect whether two student programs are equivalent

We evaluate using linear probe + cosine similarity

Solved same problem?

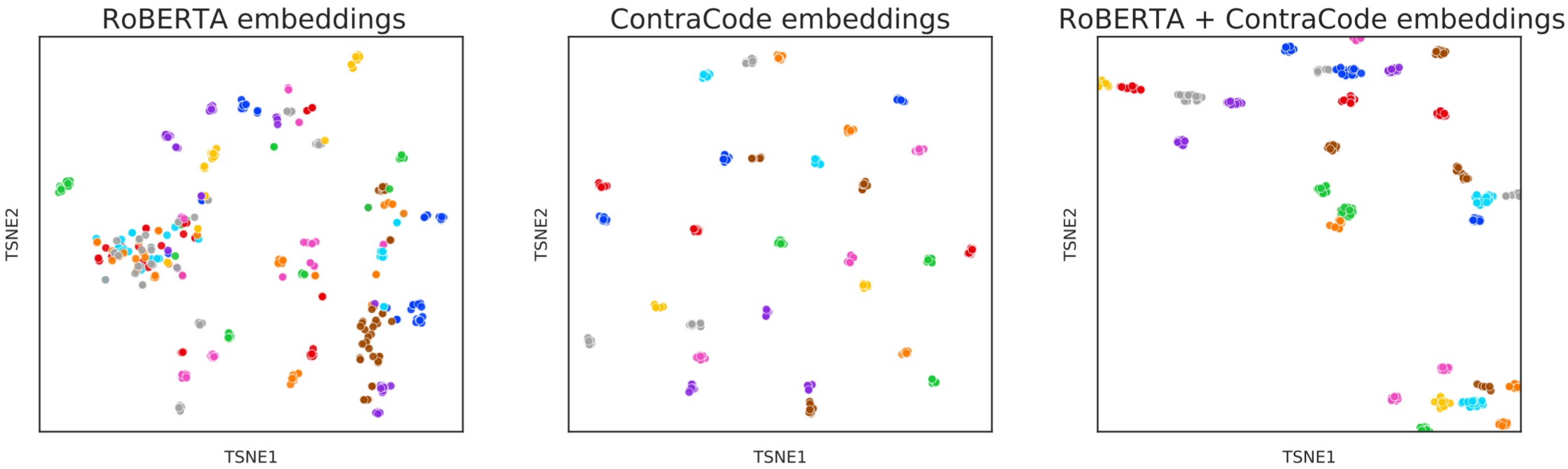
	Adversarial (N=4)	
	AUROC	**AP**
Edit distance heuristic	31.63±0.82	42.85
Randomly initialized Transformer	22.72±0.20	37.73
+ RoBERTa MLM pre-train	25.83±0.21	39.46

Underperform random guess

Contrastive pre-train

+ 28.5% AP

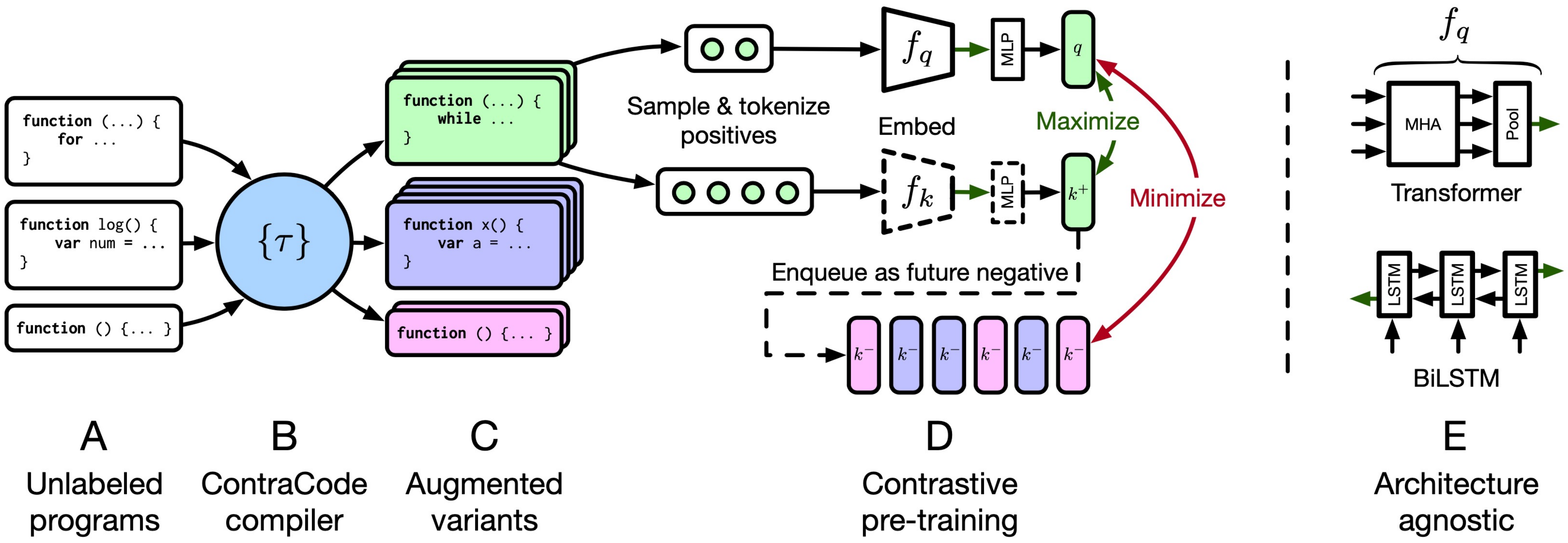
BERT vs Contrastive representation space



Contrastive Code Representation Learning

Paras Jain, Ajay Jain, Tianjun Zhang, Pieter Abbeel, Joseph E. Gonzalez, Ion Stoica

https://github.com/parasj/contracode
https://arxiv.org/abs/2007.04973



- We propose a self-supervised learning algorithm to **learn structured representations of code**.
- We leverage contrastive learning to induce the invariant that **semantically equivalent programs should have similar representations**.
- Our model demonstrates **consistent improvements over SOTA** for code summarization, type inference and zero-shot code clone detection.