Contrastive Code Representation Learning

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

Code and paper:
parasj.github.io/contracode

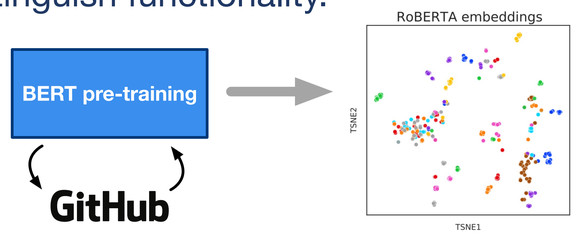
Email me:
ajayj@berkeley.edu

Motivation

- ML improves & accelerates code analysis
- Problem: Learned code analysis is **data hungry** and **sensitive to minor implementation details**
- How to represent the *functionality* of a program in an unsupervised manner?

Unsupervised representation of code

RoBERTa can pre-train on large corpus of unlabeled code, but representations do not distinguish functionality.

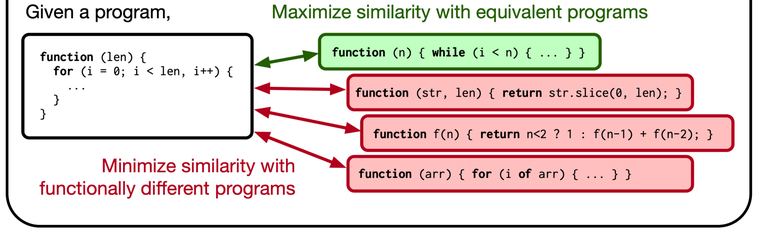


Contrastive learning is great for code

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

ContraCode pre-trains code encoders with an unsupervised contrastive objective to encode prior knowledge of functionality into the representation.

$$\mathcal{L}_{q,k^+,k^-} = -\log \frac{\exp(q \cdot k^+/t)}{\exp(q \cdot k^+/t) + \sum_{k^-} \exp(q \cdot k^-/t)}$$

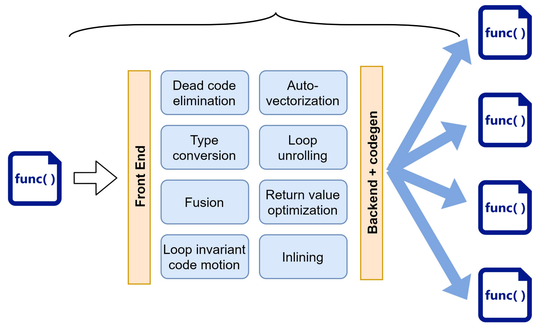


Data generation with compilers

- Using source-to-source compiler transforms, we generate diverse pairs of equivalent of programs.

	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




```
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

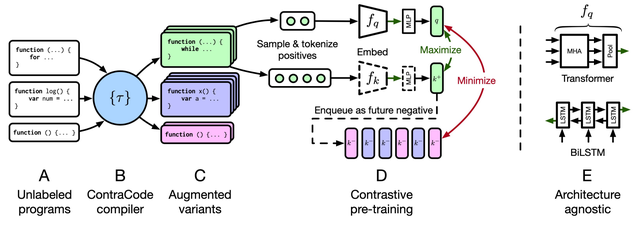
```
function x(t) {
  const n = {
    'text': '',
    'data': data
  };
  for (;i < t; i += 1) {
    n.text += lines[i] + '\n';
  }
  n && parsingCtx.sections.push(n);
}
```

Renamed variables, explicit object style, explicit concatenation, inline conditional

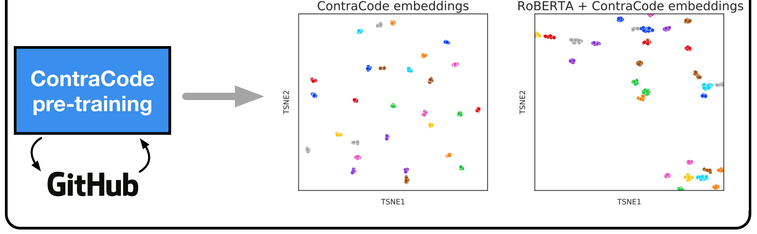
```
function x(t){const
n={'text':'','data':data};for(;i<t;i+=
1)n.text+=lines[i]
+'\n';n&&parsingCtx.sections.push(n)}
```

Mangled source with compressed whitespace

Training pipeline



Visualizing the representation



Type inference JavaScript → TypeScript

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%
+ ContraCode pre-train	46.86%	**81.85%**
+ ContraCode + MLM (hybrid)	**47.16%**	81.44%
DeepTyper BiLSTM	51.73%	82.71%
+ RoBERTa MLM pre-train	50.24%	82.85%
+ ContraCode pre-train	**54.01%**	**85.55%**

Contrastive pre-train
Hybrid pre-train
+ 1.5% top-1
Contrastive pre-train
+ 2.3% top-1
+ 8.9% vs static analysis

```
function rendererError ( message: string 99.5% ): void 99.7% {
```

Code summarization (JavaScript → English)

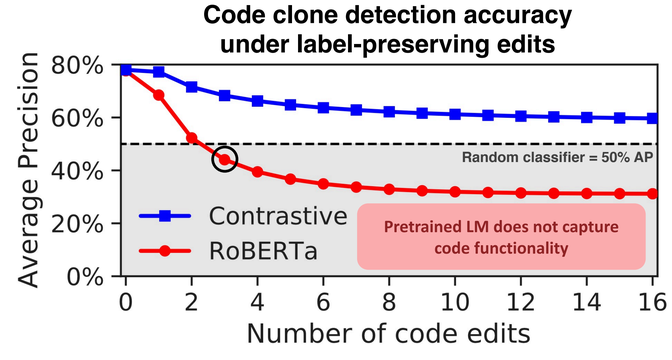
Generate short English summary of a method.

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%
+ ContraCode	20.34%	14.96%	**17.24%**

Contrastive pre-train

Zero-shot Clone Detection (JavaScript)

Detect when human-written programs are equivalent.



	Natural code AUROC	Natural code AP
Edit distance heuristic	69.55±0.81	73.75
Randomly initialized Transformer	72.31±0.79	75.82
+ RoBERTa MLM pre-train	74.04±0.77	77.65
+ ContraCode pre-train	75.73±0.75	78.02
+ ContraCode + RoBERTa MLM	**79.39**±0.70	**81.47**

RoBERTa MLM pre-train
+ 1.8% AP
Contrastive pre-train
+ 2.2% AP
Hybrid pre-train
+ 5.56% AP

riselab BAIR EMNLP 2021