

Can pruning be used to understand how language models for text differ from language models for code?

Exciting Directions for Exploration!

Can we look holistically at other code tasks/metrics to understand what is lost when we prune? (CodeT5 on APPS)

| pass@k metric | Dense (%) | OMP(%) | IMP(%) |
|---------------|--------------|--------|--------|
| pass@1 | 74.0 | 59.0 | 44.66 |
| pass@2 | 93.0 | 82.56 | 69.35 |
| pass@10 | 99.99 | 99.93 | 99.71 |

How much information do code models really need to produce and/or summarize code (data pruning)?

```

1 public String readFile (String filename) {
2   String content = null;
3   File file = new File(filename);
4   FileReader reader = null;
5   try {
6     reader = new FileReader(file);
7     char[] chars = new char[(int)file.length()];
8     reader.read(chars);
9     content = new String(chars);
10    reader.close();
11  } catch (IOException e) {
12    e.printStackTrace();
13  } finally {
14    if(reader != null){
15      reader.close();
16    }
17  }
18  return content;
19 }
  
```

(a) The original source code

```

1 public String readFile(String filename){
2   String content = null;
3   File file = new File(filename);
4   FileReader reader = null;
5   try {
6     reader = new FileReader(file);
7     reader.read(chars);
8     content = new String(chars);
9     reader.close();
10  } catch (IOException e) {
11  } finally {}
12  return content;
13 }
  
```

(b) Code after statement selection

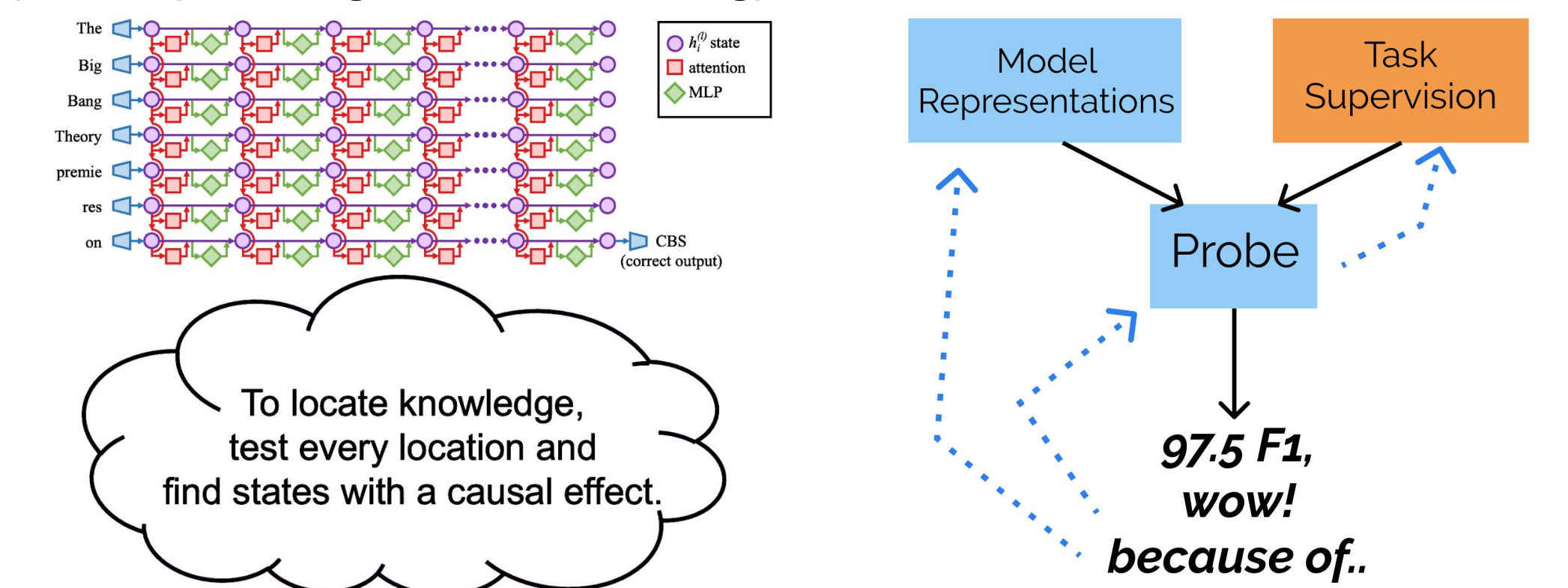
```

1 public String readFile (String filename){
2   File file = new File(filename);
3   FileReader
4   try {
5     reader = new FileReader(file);
6     reader.read(chars);
7     content = new String(chars);
8     reader.close();
9   } finally {}
10  return content;
11 }
  
```

(c) Code after token pruning

Diet Code Is Healthy: Simplifying Programs for Pre-trained Models of Code [Zhang, et al. 2022]

Can we combine pruning and interpretability techniques (linear probing, model editing) to understand what's learned?



Locating and Editing Factual Associations in GPT [Meng, et al.] John Hewitt's blog post on Designing and Interpreting Probes

How can we leverage the structured and precise nature of code to design more efficient language models on them?

- Abstractions
- Precise semantics
- Access to execution information
- Test cases (correctness)
- Compositionality

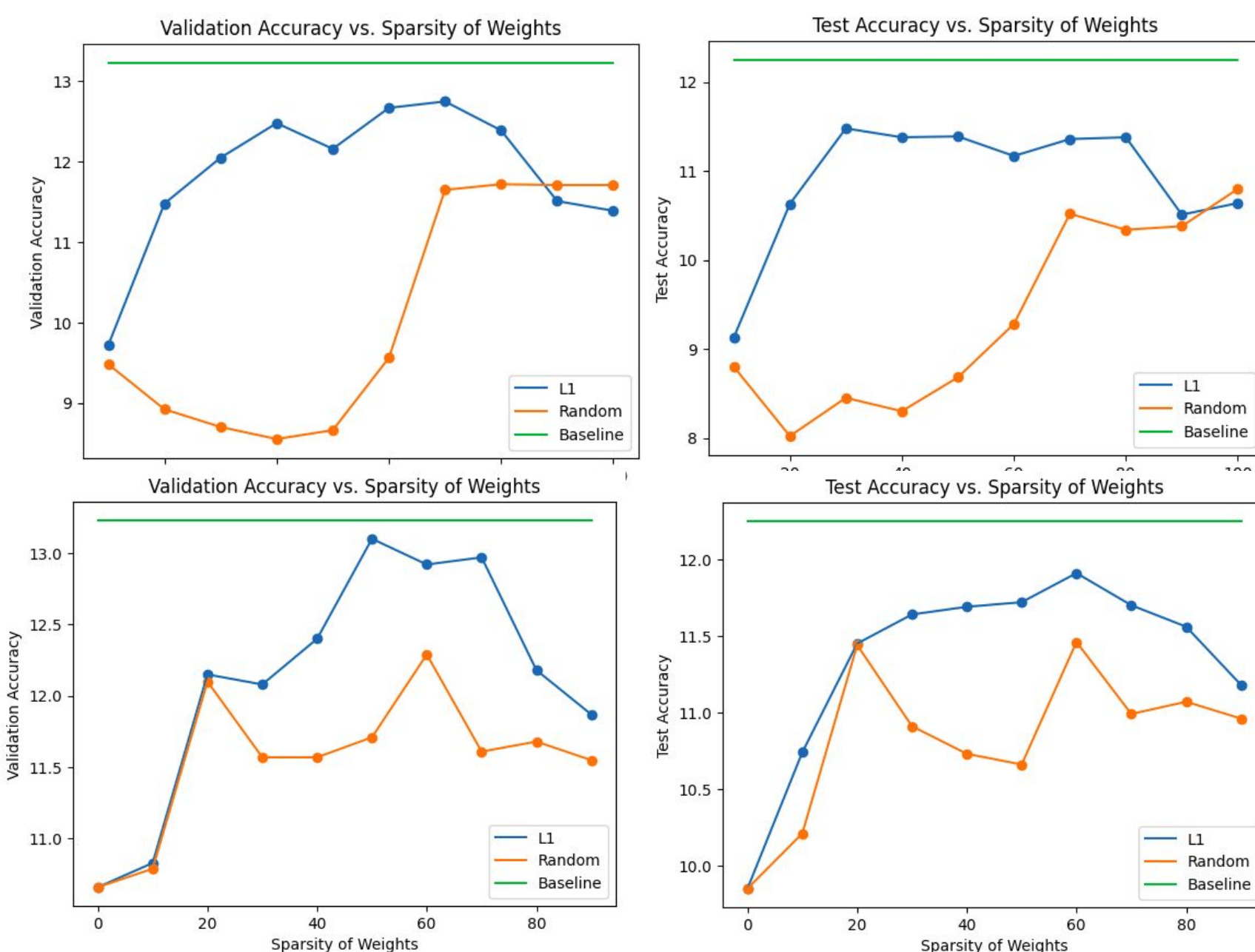
```

1. Signature vector<int> fun(int a, vector<int> b)
2. Input-output examples [0,1,2] -> [2], [0,1,2,3] -> [2,3], [0,1,2,3,0,5] -> [2,3,5]
3. C++ function
vector<int> fun(int a, vector<int> b) {
  vector<int> c;
  for(int d = 0; d <= a; d++){
    if(b[d] > 0){
      c.push_back(b[d]);
    }
  }
  return c;
}
4. Text context
"Given a positive integer n≤10, return the number of primes psn."
5. Program Expression Graph
  
```

A large-scale benchmark for few-shot program induction and synthesis [Alet, et al.]

Setting: Ruby code summarization with CodeBERT measured with BLEU score

Failed Attempt I: Rewinding to CodeBERT?



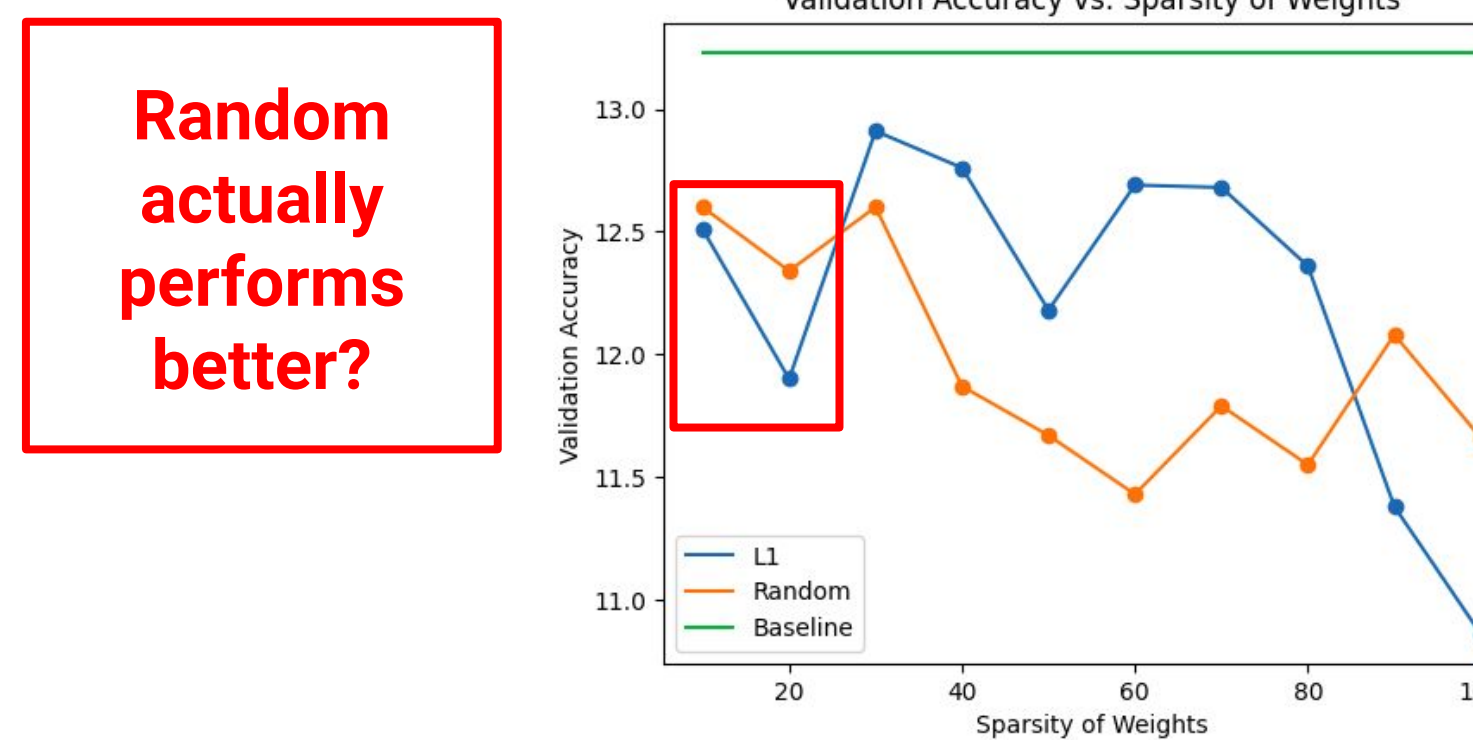
Batch Size 16

Batch Size 8

Lesson 1: batch size matters a lot

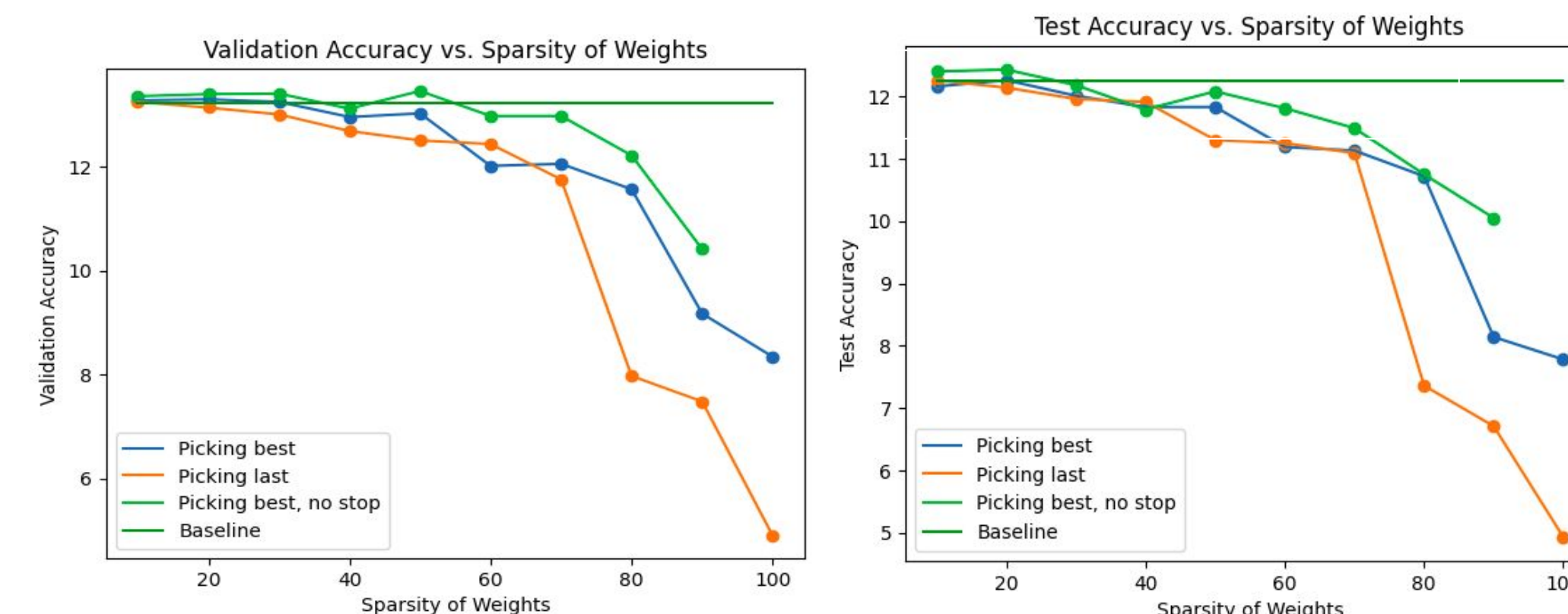
Lesson 2: need to rewind to finetuned CodeBERT?

Failed Attempt II: Overfitting?



Hypothesis: are we overfitting every time we finetune?

Final Result



A large-scale benchmark for few-shot program induction and synthesis [Alet, et al.]