Are Large Language Models Good at Generating Software Specifications? Yes, but not Quite.
Danning Xie¹, Byungwoo Yoo², Nan Jiang¹, Mijung Kim², Lin Tan¹, Xiangyu Zhang², Judy S Lee³

**Study Overview**

- **Motivation**
  - Software specifications are essential for ensuring the reliability of software systems.
  - Existing approaches on specifications extraction (from comments or documents) are domain-specific and semi-automatic.

- **Function signature**: `isNullOrNilEmpty(String string)`
  - JavaDoc comment: `@return true if the string is null or is an empty string`
  - Specification extracted by `Jdoctor`:
    ```java
    return null || string.isEmpty() 
    ```

- **Examining existing approaches on specifications extraction**
  - From comments or documents
  - Domain-specific and semi-automatic.

- **Benchmark model** — Starcoder
  - 15.5B, open-source, long input support (8,192 tokens)

- **Studied datasets and techniques**
  - `Jdoctor`: translates JavaDoc comments (`@param`, `@returns`, `@thros`) into specifications
  - `DocTer`: extracts DL-specific constraints (e.g., tensor shapes) from API documentation.

- **Random Few-Shot Learning**
  - `StarCoder`, with 10–60 of randomly selected examples, achieves comparable results with the SOTA specification extraction tools.

- **Semantic retrieval (SR) strategy** further improves Starcoder’s performance to outperforming SOTA approaches.

**RQ1 & 2: Specification Extraction Capability**

```
<table>
<thead>
<tr>
<th>Model</th>
<th>#samples</th>
<th>Open Source</th>
<th>K=10</th>
<th>20</th>
<th>40</th>
<th>60</th>
<th>Cost ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jdoctor</td>
<td>✓</td>
<td>✓</td>
<td>94.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>StarCoder</td>
<td>✓</td>
<td>✓</td>
<td>93.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>GPT-3</td>
<td>✓</td>
<td>✓</td>
<td>92.9</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>GPT-3.5</td>
<td>✓</td>
<td>✓</td>
<td>94.4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Bloom</td>
<td>✓</td>
<td>✓</td>
<td>83.5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>CodeGen</td>
<td>✓</td>
<td>✓</td>
<td>84.6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>CodeGen2</td>
<td>✓</td>
<td>✓</td>
<td>86.4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Incoder</td>
<td>✓</td>
<td>✓</td>
<td>88.4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>
```

---

**RQ3: Failure Root Cause Analysis**

- **Large Language Models**
  - We identify the root causes of the LLM by manually fixing them.

- **Ineffective prompts**: The examples selected in the prompts are not good enough. Fixed by manually selecting more relevant examples, or altering the order of examples.

- **Missing domain knowledge**: LLM is lack of context while some traditional methods are search-based.

- **Wrong focus**: The examples selected in the prompts are not good enough. Fixed by manually selecting more relevant examples, or altering the order of examples.

**Baseline Approaches**

- **Missing rule**: 78%
- **Incomplete Semantic Comprehension**: 13.5%
- **Incorrect Rule**: 8.5%

---

**RQ4: Model Comparison**

```
<table>
<thead>
<tr>
<th>Approach/Model (VR)</th>
<th>#samples</th>
<th>Open Source</th>
<th>K=10</th>
<th>20</th>
<th>40</th>
<th>60</th>
<th>Cost ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jdriver</td>
<td>✓</td>
<td>✓</td>
<td>94.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>StarCoder</td>
<td>✓</td>
<td>✓</td>
<td>93.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>GPT-3</td>
<td>✓</td>
<td>✓</td>
<td>92.9</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>GPT-3.5</td>
<td>✓</td>
<td>✓</td>
<td>94.4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Bloom</td>
<td>✓</td>
<td>✓</td>
<td>83.5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>CodeGen</td>
<td>✓</td>
<td>✓</td>
<td>84.6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>CodeGen2</td>
<td>✓</td>
<td>✓</td>
<td>86.4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Incoder</td>
<td>✓</td>
<td>✓</td>
<td>88.4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>
```

---

**Study Overview**

- **Software spec. techniques**
  - Benchmark model: Starcoder
    - 15.5B, open-source, long input support (8,192 tokens)

- **Random Few-Shot Learning**
  - `Random`: Randomly selecting K samples as the few-shots.
  - `Semantic Retrieval (SR)`: Applying a RoBERTa model as the semantic retrieval model to select the most semantically similar K samples as the few-shots.

- **Most LLMs achieve better or comparable performance as custom-built traditional specification extraction techniques.**

- **StarCoder**, an open-sourced model, is the most competitive model for extracting specifications, with its high performance, $0 cost, and long prompt support, facilitating its adaptability and customization.

- **StarCoder’s strong performance makes GPT3 Davinci less desirable given its size and cost. CodeGen and CodeGen2 are reasonable open-source alternatives.**