OpenAPI autocompletion with Large Language Models

Bohdan Petryshyn
Supervised by Assoc. Professor Mantas Lukoševičius

Terms - Code completion

- Symbol-level autocompletion vs Snippet-level autocompletion

```
import http.server
import socketserver
from http import HTTPStatus


class Handler(http.server.SimpleHTTPRequestHandler):
    def do_GET(self):
        self.send_response(HTTPStatus.OK)
        self.send_header('Content-type', 'text/html')
        self.end_headers()
        self.wfile.write(b'Hello, world!')

httpd = socketserver.TCPServer(('', 8022), Handler)
httpd.serve_forever()
```

Terms - OpenAPI specification

- Definition formats: JSON, YAML
- Definition versions: 2.0, 3.0, 3.1

```yaml
openapi: "3.0.0"
info:
  version: 1.0.0
  title: Swagger Petstore
  license:
    name: MIT
servers:
  - url: http://petstore.swagger.io/v1
paths:
  /pets:
    get:
      summary: List all pets
      operationId: listPets
      tags:
        - pets
      parameters:
        - name: limit…
      responses:
        '200':
          description: A paged array of pets
```

Problem

- Existing code generation solutions demonstrate good performance in common languages like Python. Their performance in more narrow-purpose languages like OpenAPI or GraphQL is less than satisfactory.
- OpenAPI generation can’t be evaluated using existing natural language or code generation methods. Developing an evaluation framework would stimulate improvement of solutions in this field.

Existing evaluation methods

- Machine translation metrics:
  - BLEU
  - METEOR
- Levenshtein distance
- Code generation metrics

Evaluation framework. Metrics

- Correctness - how often the generated OpenAPI definition is semantically identical to the original one.
- Validity - how often the generated OpenAPI definition is syntactically valid.
- Speed - how fast the solution generates the OpenAPI definition.

Evaluation framework. Implementation

- Automated evaluation framework
- 100 randomly selected 10-line snippets are masked from 10 definitions
- Semantic OpenAPI difference is considered during evaluation.
  - Invalid result == test failed
  - Significant difference == test failed
  - Insignificant difference:
    - Description changed
    - Example changed (structure is the same)

Evaluation framework - Definitions

- Criterias for definitions for the evaluation framework
  - Most recent definitions
  - Not found in The Stack dataset
  - Human-written
  - 3,000+ lines. Preferring 20,000+ lines
  - Different fields
  - Different companies

Solution prototype. Platforms and tools

- Foundational model - Code Llama from Meta
- Infrastructure platform - Hugging Face Inference Endpoints
- Client-side programming language - JavaScript

Results. Hardware used

Model	GPU	GPU RAM	CPU	RAM	Hourly price (as of the time of writing)
Code Llama 7B	1x Nvidia A10G	24 GB	6 vCPU	28 GB	1.3 USD
Code Llama 13B	4x Nvidia A10G	96 GB	46 vCPU	175 GB	7 USD

Results. GitHub Copilot

Results

- Correctness: **29%**
- Validity: **68%**
- Speed: **19** char/s

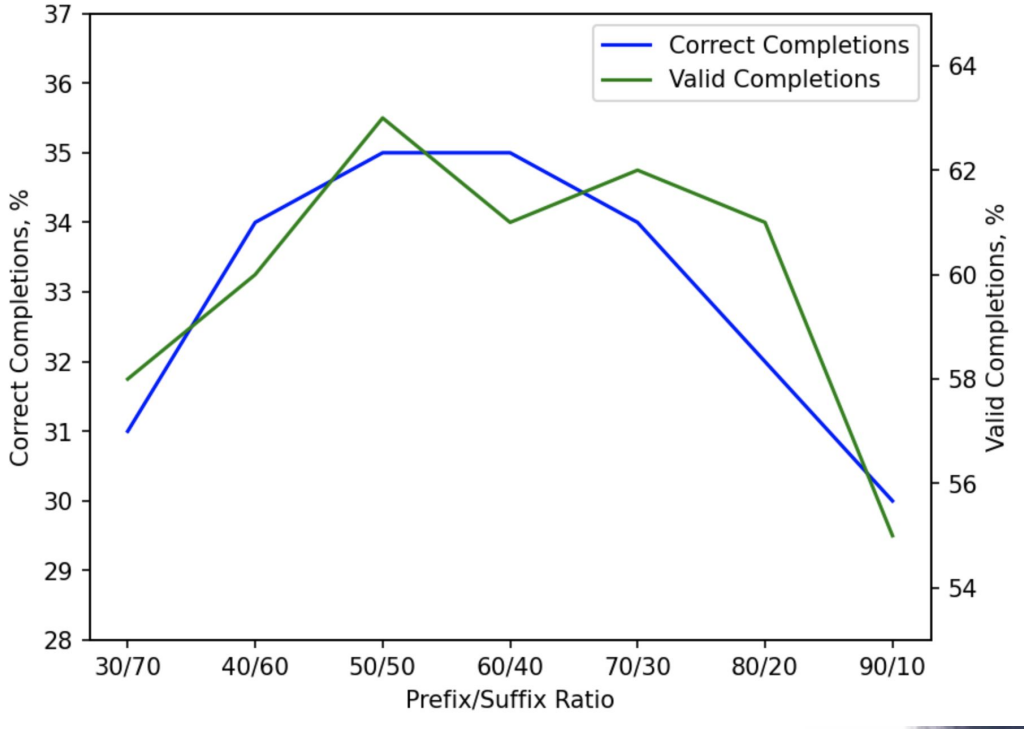
GitHub Copilot

Results. Optimal prefix-to-suffix ratio

Optimal ratio: 50/50

Results:

- Correctness: **36%**
- Validity: **61%**
- Speed: **81** char/s



Results. Optimal context size

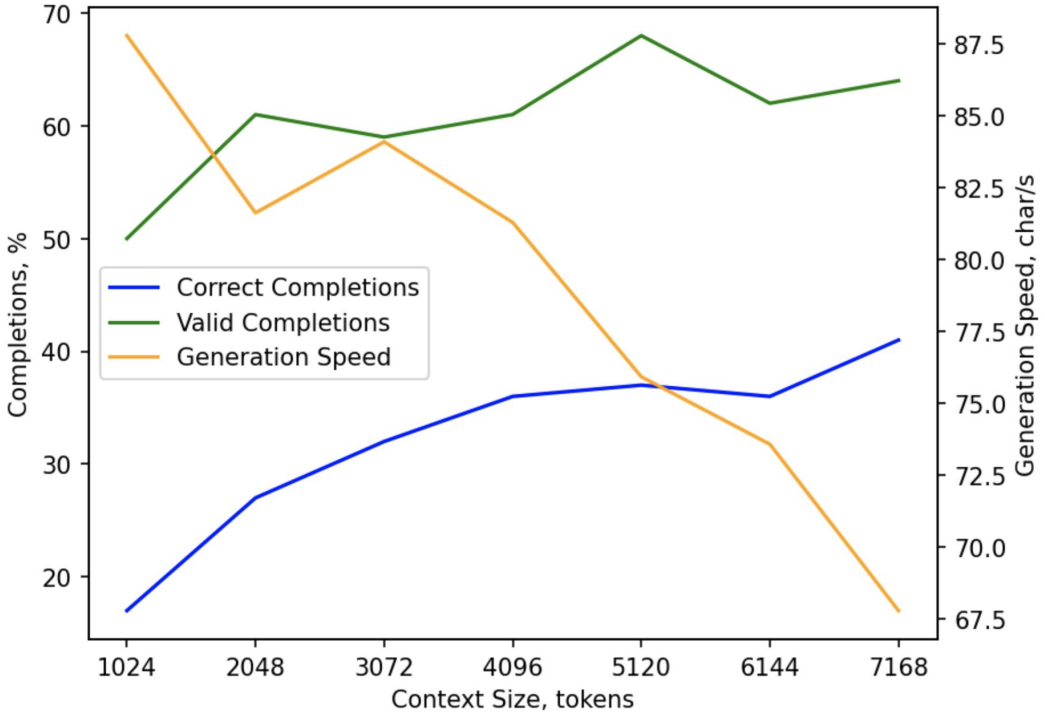
Optimal size: 4096 tokens

Results:

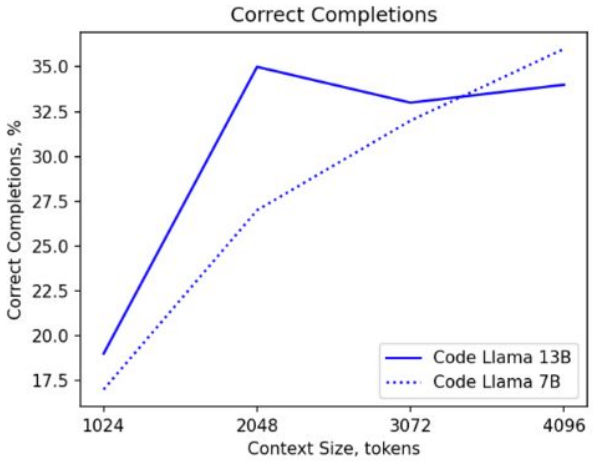
- Correctness: **36%**
- Validity: **61%**
- Speed: **81** char/s

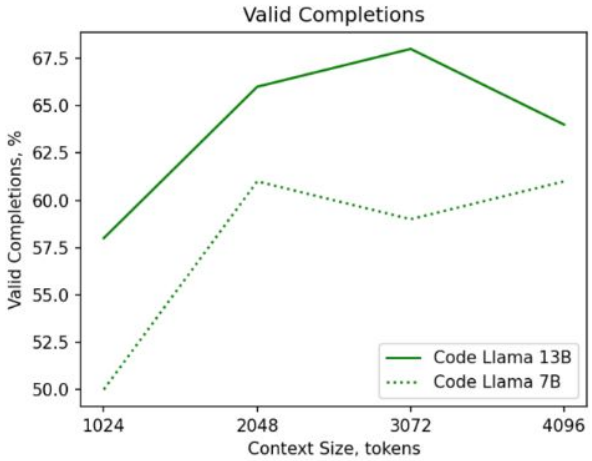
Best:

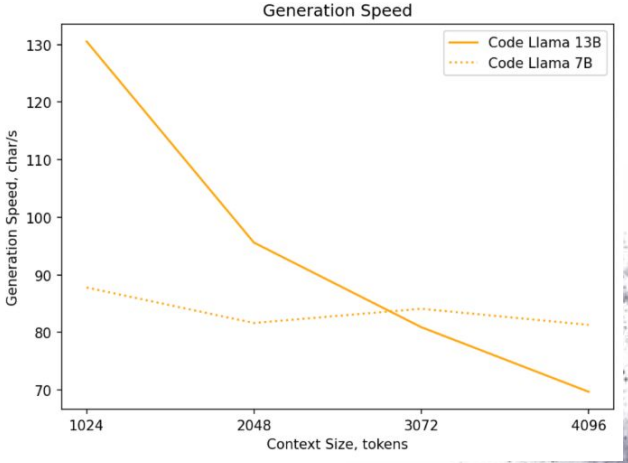
- Correctness: **41%**
- Validity: **64%**
- Speed: **67** char/s



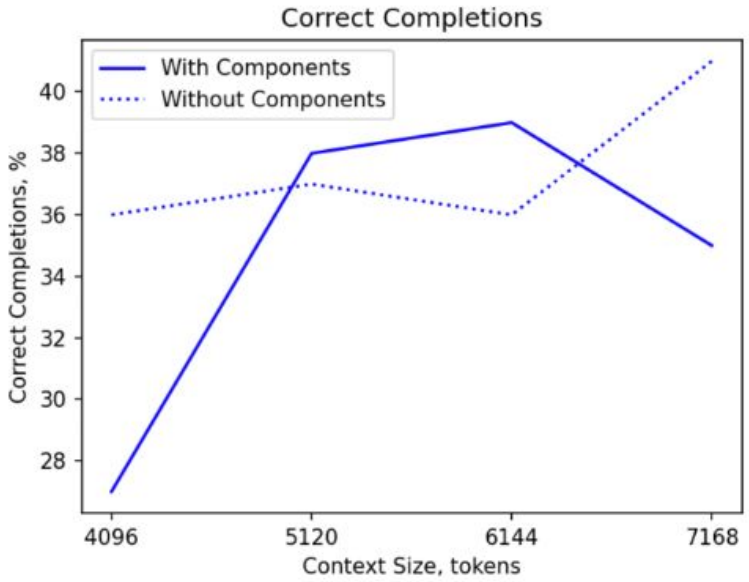
Results. Optimal model sie

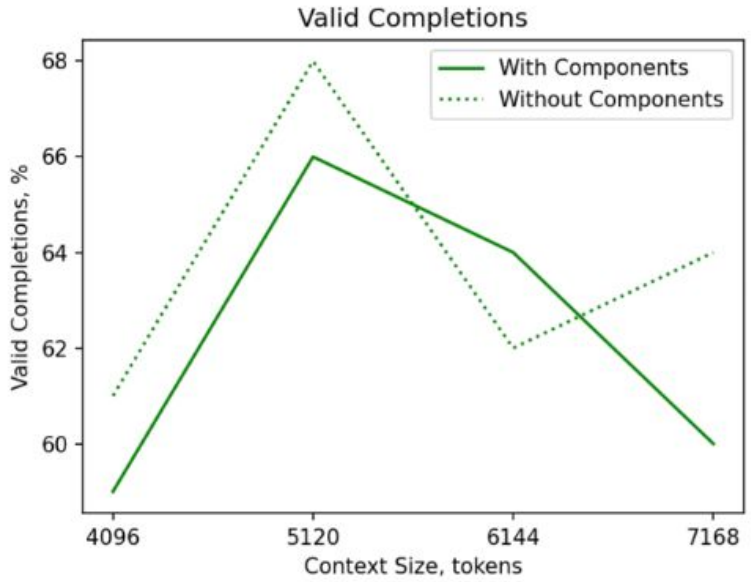






Results. OpenAPI metadata in prompt





Results. YAML vs JSON

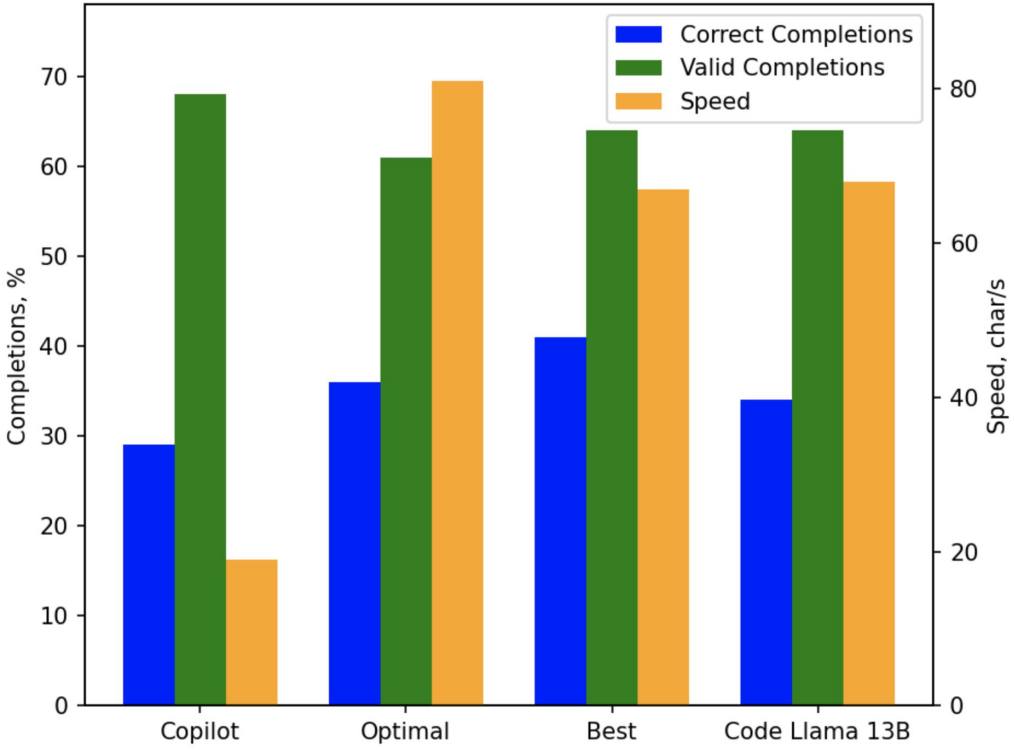
Results:

- Correctness: **27% (-8%)**
- Validity: **56% (-5%)**
- Speed: **72** char/s **(-11%)**

JSON vs YAML size:

- Characters: **+ 43.5%**
- Tokens: **51.5%**

Results. Comparison



Next steps

- Code Llama fine tuning
  - Most of the definitions were already used for training
  - Dataset size >>> Learned parameters size
- Build VSCode extension prototype
  - Would help to evaluate usability