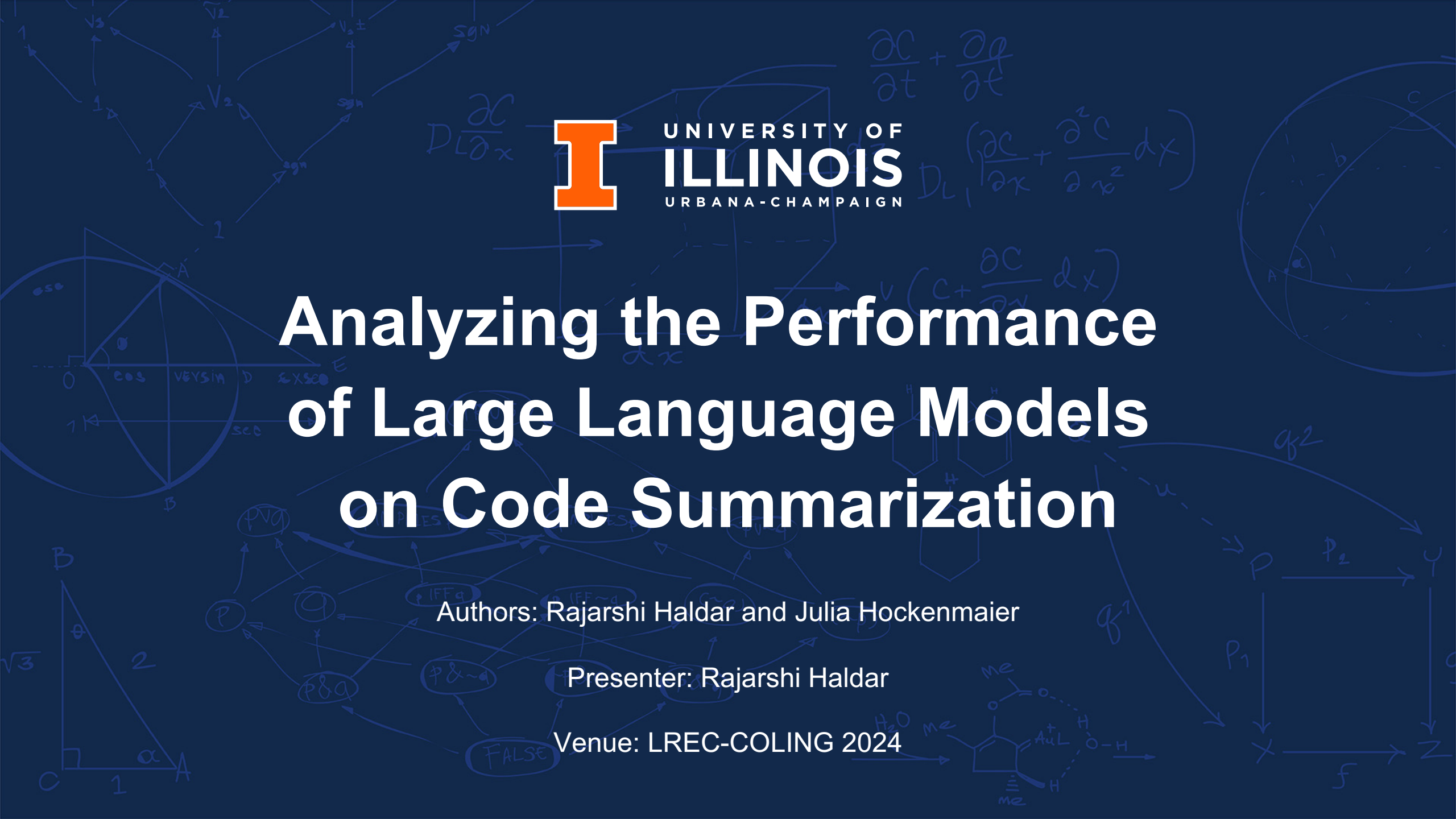


UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Analyzing the Performance of Large Language Models on Code Summarization

Authors: Rajarshi Haldar and Julia Hockenmaier

Presenter: Rajarshi Haldar

Venue: LREC-COLING 2024

INTRODUCTION

Task: Code Summarization

Translate code snippets into automatically generated natural language summaries that describe what the code does

```
def has_task(self, task_instance):
      if task_instance.key in self.queued_tasks or
task_instance.key in self.running:
            return True
```

Checks if a task is either queued or running in this executor

Motivation – Do LLMs understand code?

Answer – Yes, according to scoring on benchmarks

- Examples: CodeSearchNet, CodeXGLUE

Do these scores actually tell us

- Anything about an **LLM's** understanding of code?
- Whether these **datasets** challenging enough to require deep understanding?
- What commonly used **metrics** like BLEU-4 are measuring?

Research Questions

1. Do LLMs generate summaries by **copying tokens** directly from the code?
   - We compute the distribution of copying extent from code to description

2. Do LLMs depend on **function names** describing the semantics of the code?
   - We obfuscate function names of the code

3. Do LLMs understand the **syntactic structure** and **logic** of the code?
   - We remove the syntactic structure of the code

Research Question 1

Do LLMs generate summaries by copying tokens directly from the code?

Do LLMs Copy Tokens From Code?

Code usually contains meaningful names

- for **functions** and other **identifiers**

One simple strategy –

- **copy** most English-sounding tokens from code and tie them together to form a description
- would NOT require deep **understanding** of the semantics of the code structure

This would be effective

- so long as the generation is **fluent**

Question: Are LLMs using this strategy?

Problem: How do we **measure** extent of copying?

Do LLMs Copy Tokens From Code?

p_{copy} –

percentage of tokens in description that are also in the code

Do LLMs Copy Tokens From Code?

Code

```
def invoke_lambda( self, payload):
        awslambda_conn = self.get_conn()
        response = awslambda_conn.invoke(
                FunctionName = self.function_name,
                InvocationType = self.invocation_type,
                LogType = self.log_type,
                Payload = payload,
                Qualifier = self.qualifier)
                return response
```

Tokenized Code

```
def invoke _ lambda self payload aw sl ambda _ conn = self get _
conn response = aw sl ambda _ conn invoke function name = self
function _ name invocation type = self inv ocation _ type log
type = self log _ type payload = payload qualifier = self
qualifier return response
```

Description

Invoke Lambda Function

Do LLMs Copy Tokens From Code?

Code

```
def invoke_lambda( self, payload):
        awslambda_conn = self.get_conn()
        response = awslambda_conn.invoke(
                FunctionName = self.function_name,
                InvocationType = self.invocation_type,
                LogType = self.log_type,
                Payload = payload,
                Qualifier = self.qualifier)
                return response
```

Tokenized Code

```
def invoke _ lambda self payload aw sl ambda _ conn = self get _
conn response = aw sl ambda _ conn invoke function name = self
function _ name invocation type = self inv ocation _ type log
type = self log _ type payload = payload qualifier = self
qualifier return response
```

Description

Invoke Lambda Function

Do LLMs Copy Tokens From Code?

Code

```
def invoke_lambda( self, payload):
        awslambda_conn = self.get_conn()
        response = awslambda_conn.invoke(
                FunctionName = self.function_name,
                InvocationType = self.invocation_type,
                LogType = self.log_type,
                Payload = payload,
                Qualifier = self.qualifier)
                return response
```

Tokenized Code

```
def invoke _ lambda self payload aw sl ambda _ conn = self get _
conn response = aw sl ambda _ conn invoke function name = self
function _ name invocation type = self inv ocation _ type log
type = self log _ type payload = payload qualifier = self
qualifier return response
```

Description

Invoke Lambda Function

Do LLMs Copy Tokens From Code?

Code

```
def invoke_lambda( self, payload):
        awslambda_conn = self.get_conn()
        response = awslambda_conn.invoke(
                FunctionName = self.function_name,
                InvocationType = self.invocation_type,
                LogType = self.log_type,
                Payload = payload,
                Qualifier = self.qualifier)
                return response
```

Tokenized Code

```
def invoke _ lambda self payload aw sl ambda _ conn = self get _
conn response = aw sl ambda _ conn invoke function name = self
function _ name invocation type = self inv ocation _ type log
type = self log _ type payload = payload qualifier = self
qualifier return response
```

Description

Invoke Lambda Function

Do LLMs Copy Tokens From Code?

Code

```
def invoke_lambda( self, payload):
        awslambda_conn = self.get_conn()
        response = awslambda_conn.invoke(
                FunctionName = self.function_name,
                InvocationType = self.invocation_type,
                LogType = self.log_type,
                Payload = payload,
                Qualifier = self.qualifier)
                return response
```

Tokenized Code

```
def invoke _ lambda self payload aw sl ambda _ conn = self get _
conn response = aw sl ambda _ conn invoke function name = self
function _ name invocation type = self inv ocation _ type log
type = self log _ type payload = payload qualifier = self
qualifier return response
```

p_{copy} = 100

Description

Invoke Lambda Function

Do LLMs Copy Tokens From Code?

Code

```
def resetdb():
        from airflow import models
        from alembic.migration import MigrationContext
        log.info("Dropping tables that exist")
        models.base.Base.metadata.drop_all(settings.engine)
        mc = MigrationContext.configure(settings.engine)
        if mc._version.exists(settings.engine):
                mc._version.drop(settings.engine)
        from flask_appbuilder.models.sqla import Base
        Base.metadata.drop_all(settings.engine)
        initdb()
```

Tokenized Code

```
def reset db from air flow import models from a le mb ic
migration import migration context log info " drop ping tables
that exist " models base base metadata drop _ all settings
engine mc = migration context configure settings engine if mc _
version exists settings engine mc _ version drop settings engine
from flask _ app builder models sql a import base base metadata
drop _ all settings engine init db
```

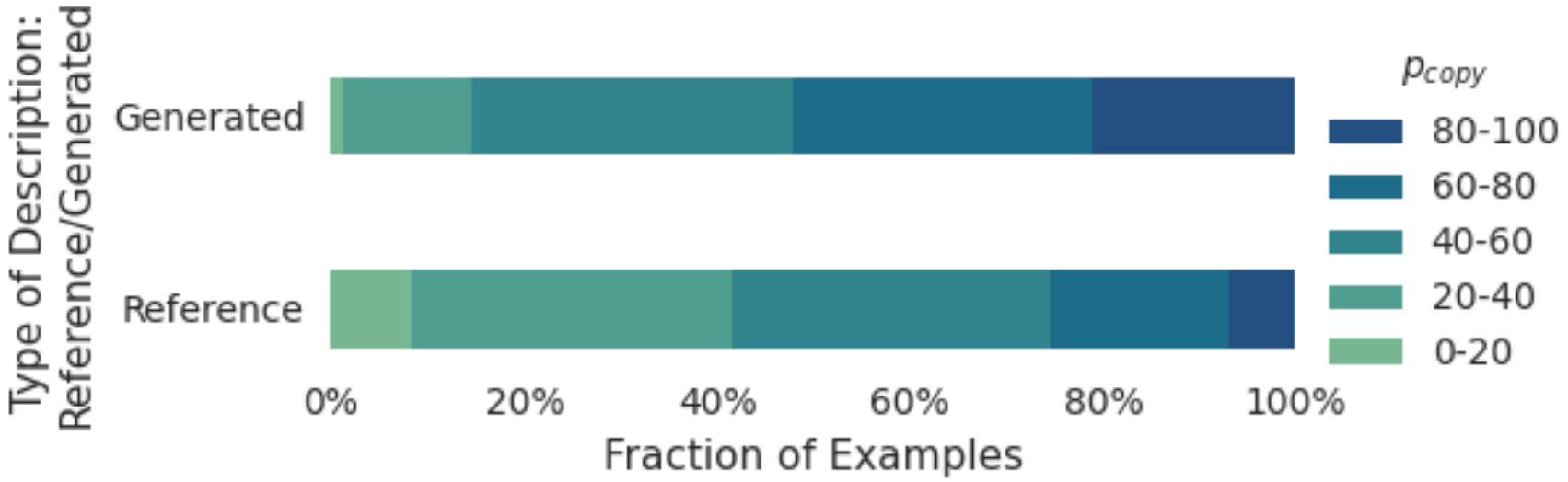
$p_{copy} = 0$

Description

Clear out the database

How high is p_{copy} in reference descriptions in the dataset?

Copying occurs in reference descriptions with an average pcopy of 45.51



How high is p_{copy} in reference descriptions in the dataset?

Copying occurs in reference descriptions with an average pcopy of 45.51

```
def resetdb():
        from airflow import models
        from alembic.migration import MigrationContext
        log.info("Dropping tables that exist")
        models.base.Base.metadata.drop_all(settings.engine)
        mc = MigrationContext.configure(settings.engine)
        if mc._version.exists(settings.engine):
                mc._version.drop(settings.engine)
        from flask_appbuilder.models.sqla import Base
        Base.metadata.drop_all(settings.engine)
        initdb()
```

Description: Clear out the database

```
def invoke_lambda( self, payload):
        awslambda_conn = self.get_conn()
        response = awslambda_conn.invoke(FunctionName = self.function_name,
                InvocationType = self.invocation_type,
                LogType = self.log_type,
                Payload = payload,
                Qualifier = self.qualifier)
                return response
```

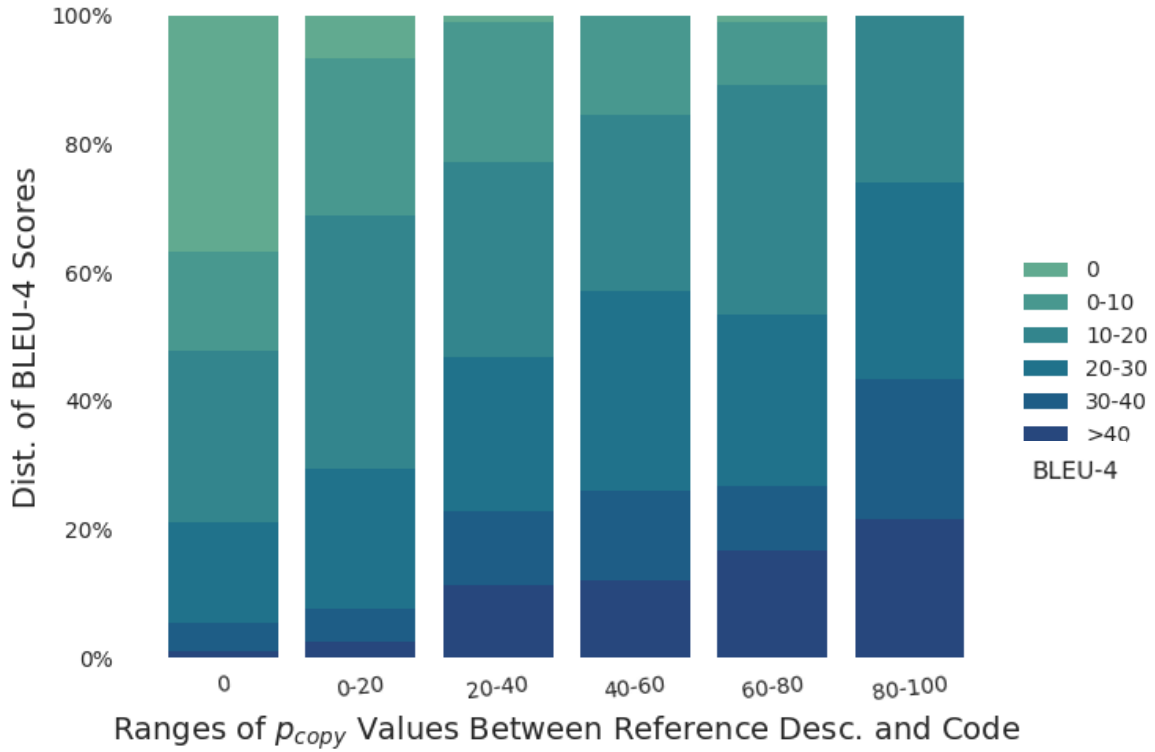
Description: Invoke Lambda Function

Question: How does this affect descriptions generated by an LLM?

We look at the distribution of BLEU scores of generations from CodeT5

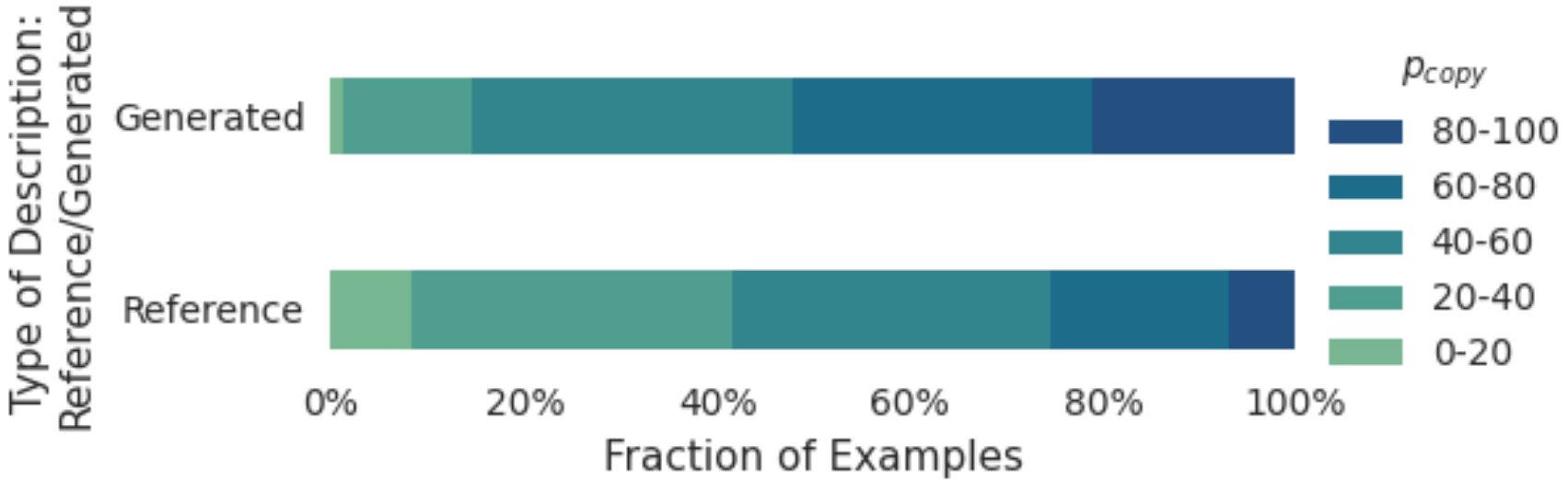
Are examples with higher p_{copy} easier to score high on?

LLMs score higher for examples with more tokens copied from the code



How high is p_{copy} in descriptions generated by an LLM?

Copying is much higher in LLM-generated descriptions



Experiments

Changing Function Names

Original Function Names

```
def get_vid_from_url(url):
    vid = match1(
        url, 'https?://www.mgtv.com/(?:b|l)/\d+/(\d+).html')
    if not vid:
        vid = match1(
            url, 'https?://www.mgtv.com/hz/bdpz/\d+/(\d+).html')
return vid
```

Obfuscated Function Names

```
def hfu_wje_gspn_vsm(url):
    vid = match1(
        url, 'https?://www.mgtv.com/(?:b|l)/\d+/(\d+).html')
    if not vid:
        vid = match1(
            url, 'https?://www.mgtv.com/hz/bdpz/\d+/(\d+).html')
return vid
```

Adversarial Function Names

```
def train(url):
    vid = match1(
        url, 'https?://www.mgtv.com/(?:b|l)/\d+/(\d+).html')
    if not vid:
        vid = match1(
            url, 'https?://www.mgtv.com/hz/bdpz/\d+/(\d+).html')
return vid
```

Changing Code Structure

Original Code

```
def get_vid_from_url(url):
    vid = match1(
        url, 'https?://www.mgtv.com/(?:b|l)/\d+/(\d+).html')
    if not vid:
        vid = match1(
           url, 'https?://www.mgtv.com/hz/bdpz/\d+/(\d+).html')
return vid
```

No Code Structure

```
get_vid_from_url url
vid match1
url 'https?://www.mgtv.com/(?:b|l)/\d+/(\d+).html'
vid
vid match1
url 'https?://www.mgtv.com/hz/bdpz/\d+/(\d+).html'
vid
```

No Function Body

```
def get_vid_from_url(url)
```

Experiments – Data

CodeXGLUE –

- A benchmark for several code-NL tasks including **code summarization**
- Consists of **code snippets** paired with **English descriptions** that are scraped from public open-source GitHub repositories
- Has examples from Go, Java, JavaScript, PHP, Python, and Ruby

We use the Python examples from CodeXGLUE, which contain 251,820 training, 13,914 dev, and 14,918 testing data points.

CodeXGLUE

```
def has_task(self, task_instance):
      if task_instance.key in self.queued_tasks or
task_instance.key in self.running:
            return True
```

Code

Description

Checks if a task is either queued or running in this executor

Experiments – Models

CodeT5

- An encoder-decoder model pretrained using objectives like identifier tagging and masked identifier prediction to be proficient at code understanding
- One off-the-shelf and one fine-tuned on each specific variant

PaLM 2

- an LLM made by Google AI for reasoning tasks, question answering, classification, translation, and code summarization
- We run it off-the-shelf with prompting giving 10 examples of code-description pairs

Llama 2

- An LLM designed by Meta AI that can also be used for sequence-to-sequence tasks, including an instruct model that can be asked to perform any task with a prompt
- We run two models – with 7B and 70B parameters
- We run them off-the-shelf with prompting giving 10 examples of code-description pairs

Metrics

We use **BLEU** and **BERTScore**

- Compare generated against human-written reference summaries
- Indicate F-scores (based on precision and recall)
- Range from 0 (complete mismatch) to 100 (perfect match)

BLEU

- Most commonly used code summarization metric
- Based on hard ***n*-gram matches**, up to *n=4*

BERTScore

- More recent Natural Language Generation metric
- Based on soft **token similarities** using embeddings computed by a BERT model

Code Summarization Scores (BLEU-4)

Scores drop for transformed variants

Variant	CodeT5	CodeT5 (FT)	PaLM 2	Llama 2 (7b)	Llama 2 (70b)
Original Function Names	17.66	17.66	19.23	22.36	22.41
Obfuscated Function Names	11.59	14.80	18.72	20.09	21.25
Adversarial Function Names	11.34	13.12	15.69	19.53	21.23
No Code Structure	13.96	16.57	11.92	15.11	18.42
No Function Body	13.94	15.27	11.90	14.93	18.16

Code Summarization Scores (BLEU-4)

Fine-tuning improves CodeT5's performance

Variant	CodeT5	CodeT5 (FT)	PaLM 2	Llama 2 (7b)	Llama 2 (70b)
Original Function Names	17.66	17.66	19.23	22.36	22.41
Obfuscated Function Names	11.59	14.80	18.72	20.09	21.25
Adversarial Function Names	11.34	13.12	15.69	19.53	21.23
No Code Structure	13.96	16.57	11.92	15.11	18.42
No Function Body	13.94	15.27	11.90	14.93	18.16

Code Summarization Scores (BLEU-4)

Fine-tuning improves CodeT5's performance

- Especially on Obfuscated Function Names and No Code Structure

Variant	CodeT5	CodeT5 (FT)	PaLM 2	Llama 2 (7b)	Llama 2 (70b)
Original Function Names	17.66	17.66	19.23	22.36	22.41
Obfuscated Function Names	**11.59**	**14.80**	18.72	20.09	21.25
Adversarial Function Names	11.34	13.12	15.69	19.53	21.23
No Code Structure	**13.96**	**16.57**	11.92	15.11	18.42
No Function Body	13.94	15.27	11.90	14.93	18.16

Code Summarization Scores (BLEU-4)

Typically, larger models perform better

Variant	CodeT5	CodeT5 (FT)	PaLM 2	Llama 2 (7b)	Llama 2 (70b)
Original Function Names	17.66	17.66	19.23	22.36	22.41
Obfuscated Function Names	11.59	14.80	18.72	20.09	21.25
Adversarial Function Names	11.34	13.12	15.69	19.53	21.23
No Code Structure	13.96	16.57	11.92	15.11	18.42
No Function Body	13.94	15.27	11.90	14.93	18.16

Code Summarization Scores (BLEU-4)

Changing function names drops scores for transformed variants

- Adversarial Function Names shows bigger drop than Obfuscated Function Names

Variant	CodeT5	CodeT5 (FT)	PaLM 2	Llama 2 (7b)	Llama 2 (70b)
Original Function Names	17.66	17.66	19.23	22.36	22.41
Obfuscated Function Names	11.59	14.80	18.72	20.09	21.25
Adversarial Function Names	11.34	13.12	15.69	19.53	21.23
No Code Structure	13.96	16.57	11.92	15.11	18.42
No Function Body	13.94	15.27	11.90	14.93	18.16

General Takeaways

Removing code structure shows lower drop on CodeT5 than PaLM 2 and Llama 2

- This shows larger models rely more on the syntactic structure of the code

Variant	CodeT5	CodeT5 (FT)	PaLM 2	Llama 2 (7b)	Llama 2 (70b)
Original Function Names	17.66	17.66	19.23	22.36	22.41
Obfuscated Function Names	11.59	14.80	18.72	20.09	21.25
Adversarial Function Names	11.34	13.12	15.69	19.53	21.23
No Code Structure	13.96	16.57	11.92	15.11	18.42
No Function Body	13.94	15.27	11.90	14.93	18.16

Code Summarization Scores (BLEU-4)

Llama 2 (7b) is competitive with Llama 2 (70b) on the original data

Variant	CodeT5	CodeT5 (FT)	PaLM 2	Llama 2 (7b)	Llama 2 (70b)
Original Function Names	17.66	17.66	19.23	22.36	22.41
Obfuscated Function Names	11.59	14.80	18.72	20.09	21.25
Adversarial Function Names	11.34	13.12	15.69	19.53	21.23
No Code Structure	13.96	16.57	11.92	15.11	18.42
No Function Body	13.94	15.27	11.90	14.93	18.16

Code Summarization Scores (BLEU-4)

Llama 2 (7b) is competitive with Llama 2 (70b) on the original data

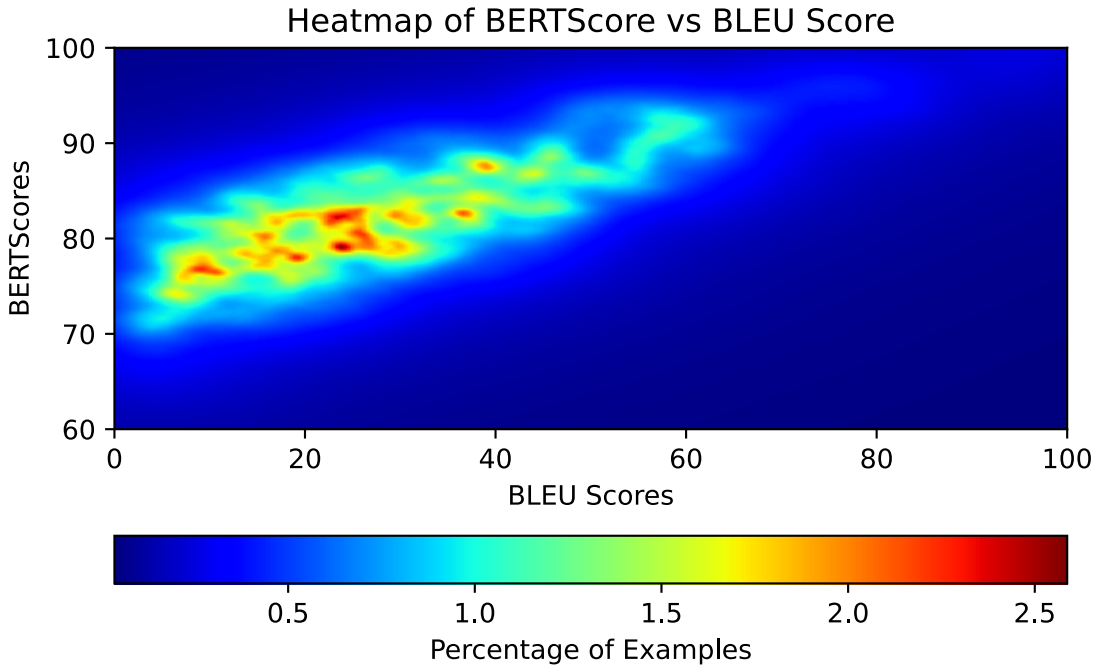
- However, the smaller model performs much worse on the transformed data

Variant	CodeT5	CodeT5 (FT)	PaLM 2	Llama 2 (7b)	Llama 2 (70b)
Original Function Names	17.66	17.66	19.23	22.36	22.41
Obfuscated Function Names	11.59	14.80	18.72	20.09	21.25
Adversarial Function Names	11.34	13.12	15.69	19.53	21.23
No Code Structure	13.96	16.57	11.92	15.11	18.42
No Function Body	13.94	15.27	11.90	14.93	18.16

Does Using a Different Metric Matter?

BLEU and BERTScore are highly correlated

- However, BERTScores are much higher than BLEU



Variant	CodeT5	CodeT5 (FT)	PaLM 2	Llama 2 (7b)	Llama 2 (70b)
Original Function Names	83.95	83.95	84.26	84.34	86.95
Obfuscated Function Names	78.57	81.25	82.28	82.52	84.41
Adversarial Function Names	78.53	80.01	80.81	80.84	84.30
No Code Structure	81.32	82.86	79.36	79.66	83.25
No Function Body	81.14	82.40	79.14	79.72	83.14

Conclusion

We presented a series of experiments to gain a deeper insight into what makes current LLMs effective at code summarization

- LLMs
  - copy tokens from the code
  - rely a lot on function names
  - rely less on the code structure
- Relying on token overlap is viable on standard benchmarks for these tasks
  - Because the corresponding descriptions in the datasets often have high token overlap
  - Because the popular metrics used like BLEU and BERTScore measure token similarity
- But, high scores on these tasks are not necessarily indicative of understanding

Thank You!

Reach out to me at rhaldar2@Illinois.edu

This work is supported by Agriculture and Food Research Initiative (AFRI) grant no. 2020-67021-32799/project accession no.1024178 from the USDA National Institute of Food and Agriculture.