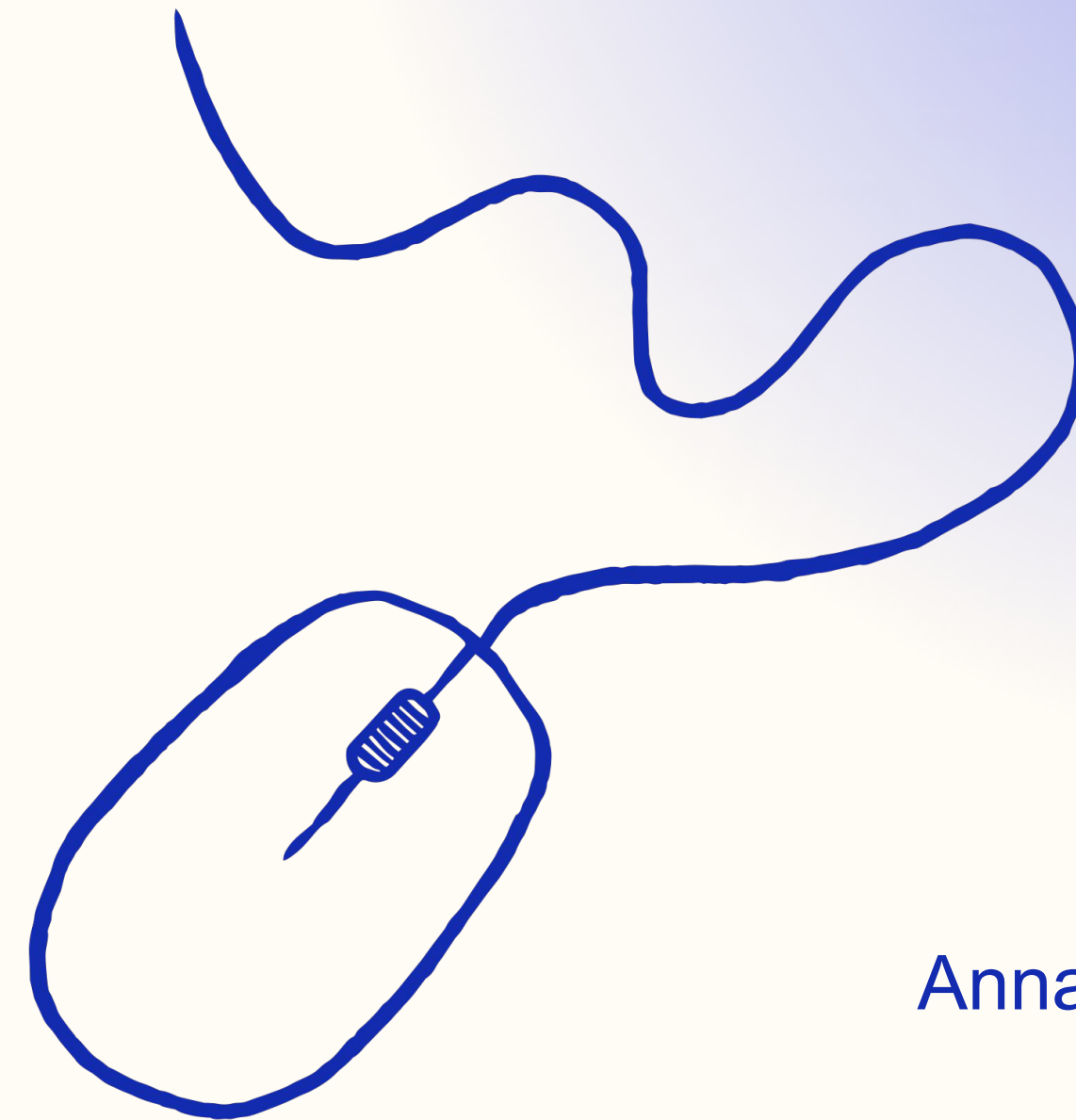


NEARCoder

Web3 Code LLM



Fine-tuning StarCoder2
for the NEAR Protocol

Julien Carbonnell
Anna-Valentina Hirsch
Kristian Boroz

01 - Introduction

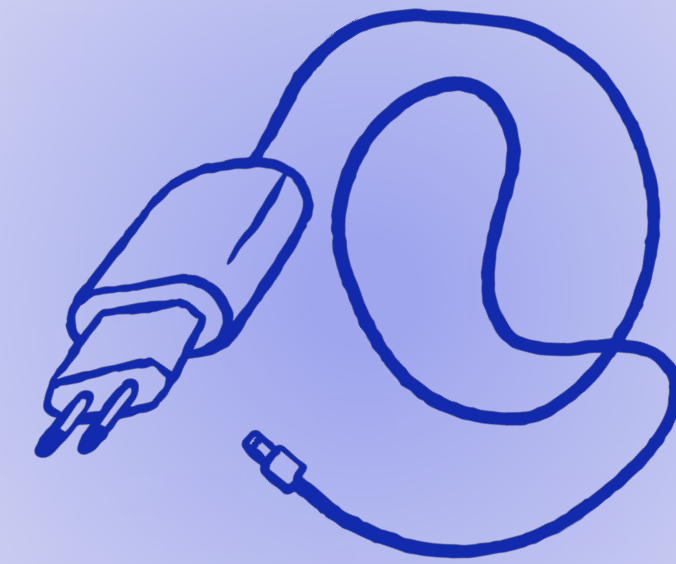
02 - Datasets

03 - Training

04 - Evaluation



Introduction - What is NEAR?



- Started in 2017 as an AI project by Illia Polosukhin, co-author of “Attention is All You Need”
- Near.ai aimed to teach machines how to code before facing the challenge to pay worldwide contributors.
- In 2018, Near creates the NEAR Protocol, a payment network on the blockchain, promoting self-sovereignty of users’ data.
- In May 2024, NEAR Protocol grew in a \$8.2B market capitalization, with millions of transactions every day, and announced the next big goal on their roadmap: **User-Owned AI**.



Introduction.

NEARCoder: help developers coding dApps on NEAR Protocol

- Web2 technologies are well covered (tutorials, forums)
- Web3 is still a nascent tech, with limited coding support:
 - Hard to hire seasoned developers
 - A powerful web3 coding assistant would be of significant utility in that context.

NEARCoder is built on top of StarCoder2 (SOTA April 24)
fine-tuned on the NEAR Protocol resources and GitHub repositories.

1. Continued Pre-Training

=> General knowledge on the Near Protocol

(blog articles, docs, papers...)

2. Structure-Aware Fine-tuning

=> dApps trees + readme files translated into user prompts.

3. Specialized Fine-Tuning

=> AST-segments of code files extracted from examples of NEAR dApps

Fine-tuning StarCoder2 on the Near Protocol



Datasets.



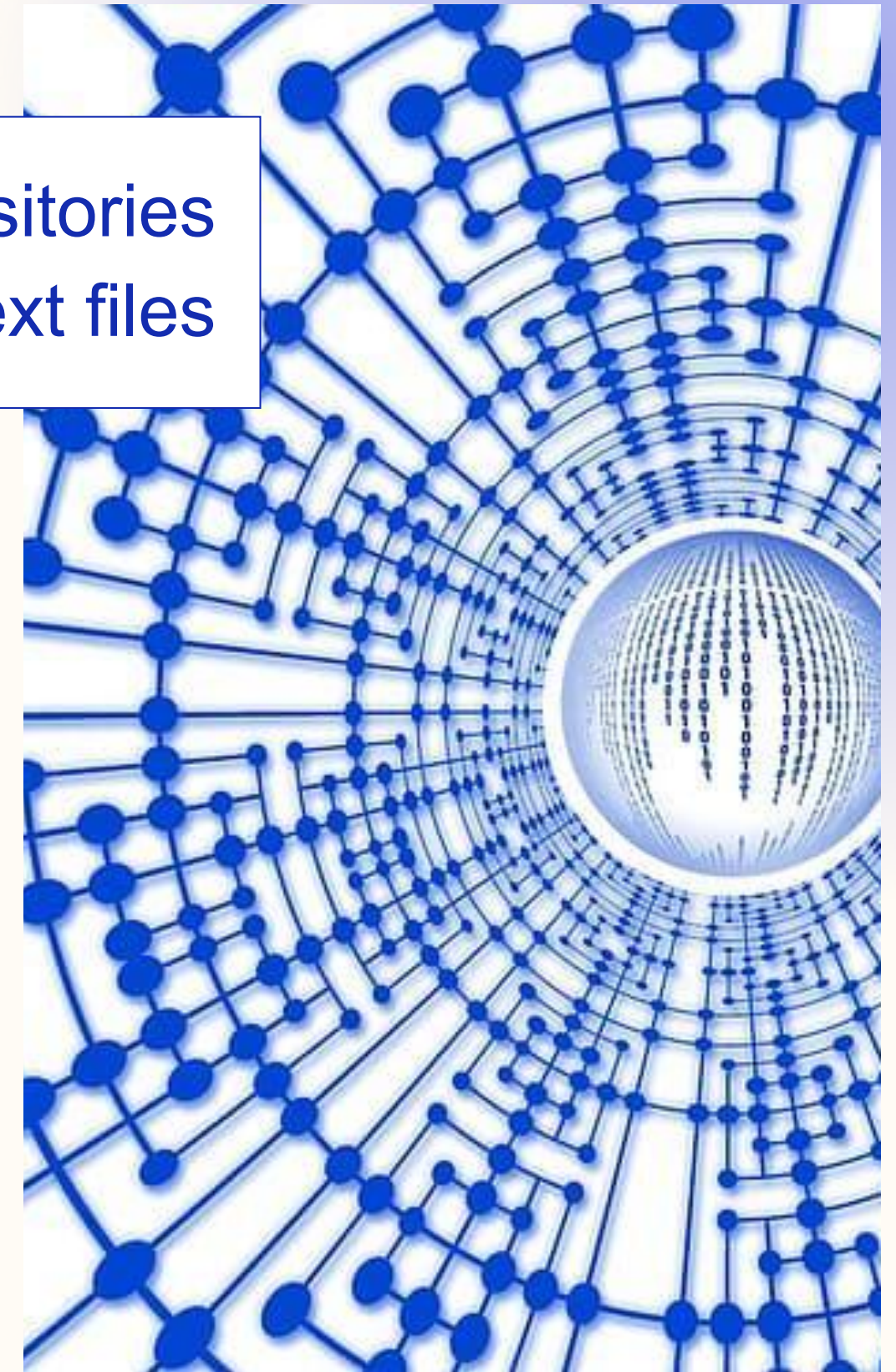
3,414 GitHub repositories
converted into text files

Data collection started with scraping examples of NEAR dApps listed at [Electric Capital](#). It took about a month to collect 3,414 repositories into text files, after cleaning duplicates and removing empty projects.

These text files of full repositories allowed the extraction of:

- **3,414 dApps trees** (structure-aware fine tuning)
- **23,166 dApps readme files** (translated into user prompts)
- **100,098 NEAR dApps code files** (segmented w/ Abstract Syntax Trees)

Complemented by **1,154 docs files** (general knowledge on NEAR)



Datasets.

Our datasets are open-sourced on Hugging Face for the sake of transparency and replicability.

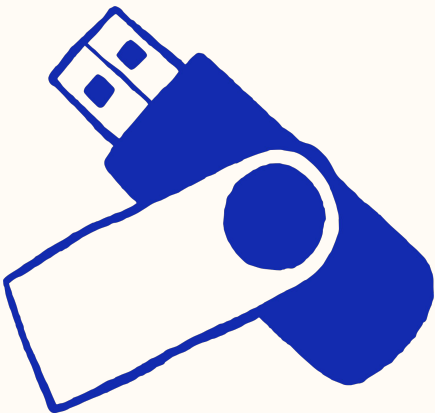
Our training data have been prepared and published into three independent datasets on Hugging Face:

- [preTrainingNEAR](#) for the Continued Pre-Training of [StarCoder2](#) on a general NEAR Protocol knowledge.
- [NEARdAppsPrompts](#) for the Structure-Aware Fine-Tuning of the [NEAR-preTrainedStarCoder2](#) with NEAR dApps trees and readme files converted into user prompts.
- [ASTCodeNEAR](#) for the Specialized Fine-Tuning of the [NEAR-structTunedStarCoder2](#) with AST-segmented code files of NEAR dApps, resulting in [NEARCoder](#).



Step 1: Continued Pre-Training.

TRAINING THE MODEL WITH GENERAL KNOWLEDGE ABOUT THE NEAR PROTOCOL.



DATA:	METHODS:
<ul style="list-style-type: none">● 1,154 official docs scraped from the internet:<ul style="list-style-type: none">○ nearDocs: 395 docs from Near Docs○ nearNode: 40 docs from Near Node Docs○ nearNEPs: 124 docs from Near Enhancement Protocol○ nearPapers: 3 papers from Near Papers○ nearWiki: 98 docs from Near Wiki	<ul style="list-style-type: none">● Supervised Fine Tuning (SFT) of StarCoder2● Hardware: NVIDIA A10G with 12 vCPU 46 GB RAM

Result: A new model “NEAR-preTrainedStarCoder2”

Step 2: Structure-Aware Fine Tuning.


TRAINING THE MODEL TO PRODUCE DIRECTORY STRUCTURES FROM PROJECT DESCRIPTIONS.

MOTIVATION:

- The first challenge is often to structure a project well
- NEAR dApps are often different from normal Apps

INITIAL DATASET:

- Repository trees + readme files extracted from 3414 dApps
 - 2.631 after cleaning



```
nearuko-nft-dapp/  
├── README.md  
├── package.json  
├── tsconfig.json  
├── contracts/  
│   ├── assembly/  
│   │   ├── __tests__/  
│   │   ├── models/  
│   │   ├── utils/  
│   │   ├── asconfig.json  
│   │   └── main.ts  
│   ├── build.sh  
│   └── deploy.sh  
├── frontend/  
│   ├── public/  
│   │   └── index.html  
│   └── src/  
│       ├── components/  
│       ├── services/  
│       └── index.tsx  
├── scripts/  
│   ├── build-contracts.js  
│   └── deploy-contracts.js  
├── tests/  
│   ├── integration/  
│   ├── unit/  
│   └── jest.config.js  
└── near-config/  
    ├── mainnet.json  
    ├── testnet.json  
    └── devnet.json
```


Step 2: Structure-Aware Fine Tuning.

TRAINING THE MODEL TO PRODUCE DIRECTORY STRUCTURES FROM PROJECT DESCRIPTIONS.



METHODS:

- Create a new labelled dataset for Direct Preference Optimization (DPO)

User Prompts:

- Data Extraction: Retrieve key details from README files
- Prompt Creation: Transform extracted details into user-friendly prompts.

Rejected Texts:

- Use NEAR-preTrainedStarCoder2 to generate trees based on the created prompts
- Assumption: These outputs are less accurate than the original trees.

Accepted Texts:

- Original Tree Structure

- Model Training with Direct Preference Optimization (DPO)

- More computationally efficient than reinforcement learning from human feedback (RLHF).

Using OpenAI API
Engine: GPT 4o

Result: A new model “NEAR-structTunedStarCoder2”

Step 2: Structure-Aware Fine Tuning.

Code example using OpenAI API

```
# Def params
engine = "gpt-4o"
max_output_tokens = 500
example = """Create a project structure for a NEAR DApp based on the following requirements:

1. The project should be related to the Nearuko NFT, which can be converted into a character in the Etheruko game.
2. Use necessary files typically required for a NEAR Protocol mainnet DApp.
3. Include all relevant dependencies and packages for an NFT project on the NEAR Protocol.
4. The main coding language should be TypeScript.
5. Ensure the project includes configurations, tests, and NEAR-compatible contract files.
6. Capture any common NEAR DApp structure conventions while setting up the project.

Provide a well-organized directory structure and file list based on these requirements."""

# Function to generate labels (prompts)
def generate_prompt(repoName, tree, readme, example):
    # Create a user prompt for a coding assistant
    prompt = (
        f"You are provided with a GitHub repository called\n{repoName}\n\n. This repository has the following directory structure\n"
        f"{tree}\n\n"
        f"The README file contains the following information\n{readme}\n\n"
        f"Step 1: Extract all the relevant information from the README file needed to predict the corresponding tree for a NEAR DApp, such as necessary files, dependencies, packages, and any particular coding languages or frameworks that should be used. "
        f"Step 2: Write a perfect user prompt asking a coding assistant to create a project stucture based only on the extracted information from the README file. Only return the user prompt from Step 2. Do not return any information about the tree or file names. Here is an example\n{example}\n\n"
    )

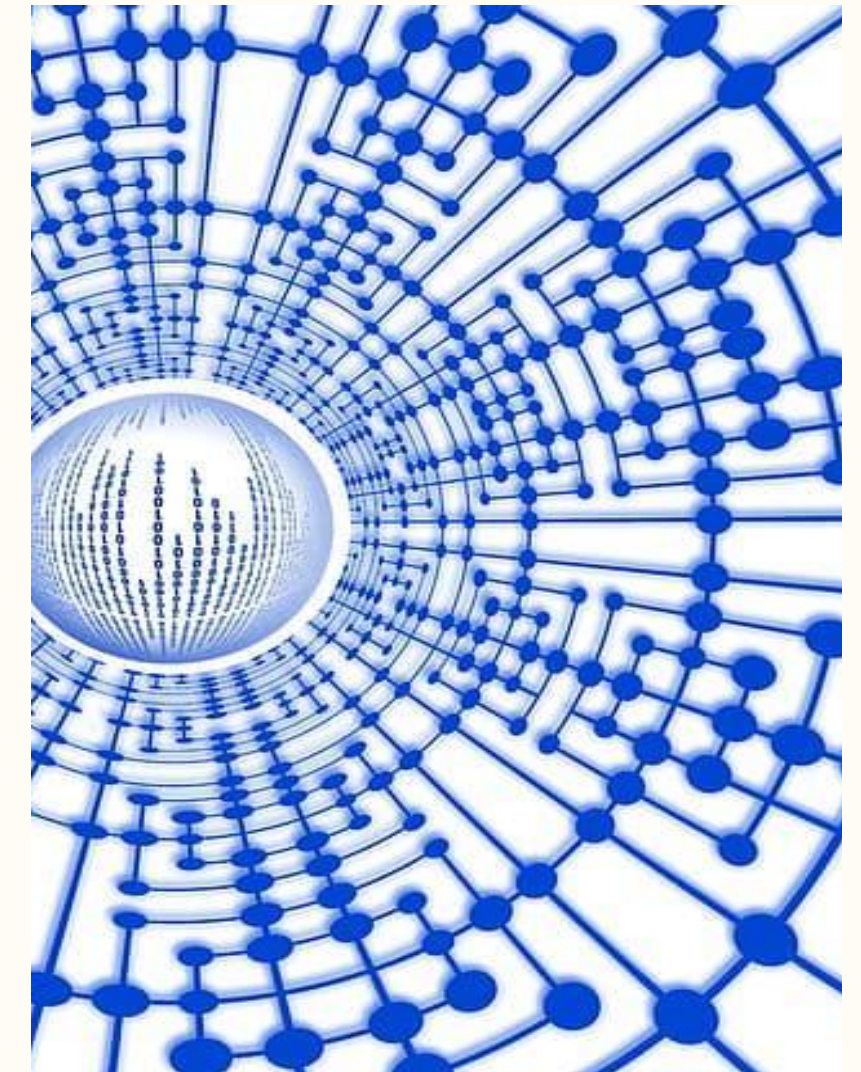
    response = client.chat.completions.create(
        model=engine,
        messages=[
            {"role": "system", "content": "You are a helpful assistant."},
            {"role": "user", "content": prompt}
        ],
        max_tokens=max_output_tokens
    )
    return response.choices[0].message.content
```

Step 3: Specialized Fine Tuning.

SFT of NEAR-structTunedStarCoder2 on AST-Segments with NVIDIA A10G.

DATA & METHODS:

- Fine-Tune the NEAR-structTunedStarCoder2 to actually produce code
- 100.098 code files (JS, RS, TS)
- Hardware: NVIDIA A10G with 12 vCPU 46 GB RAM



Result: The final model “NEARCoder”

Evaluation.

Evaluation Tasks:

- **HumanEval:** general code challenges in Javascript and Rust
- **NEAR Questions:** Specific NEAR-related questions.
- **dApp Structures:** Generating NEAR dApp trees with comments.
- **Basic NEAR dApp:** Building a simple NEAR dApp (e.g. an event calendar and a snake game).

Evaluation.

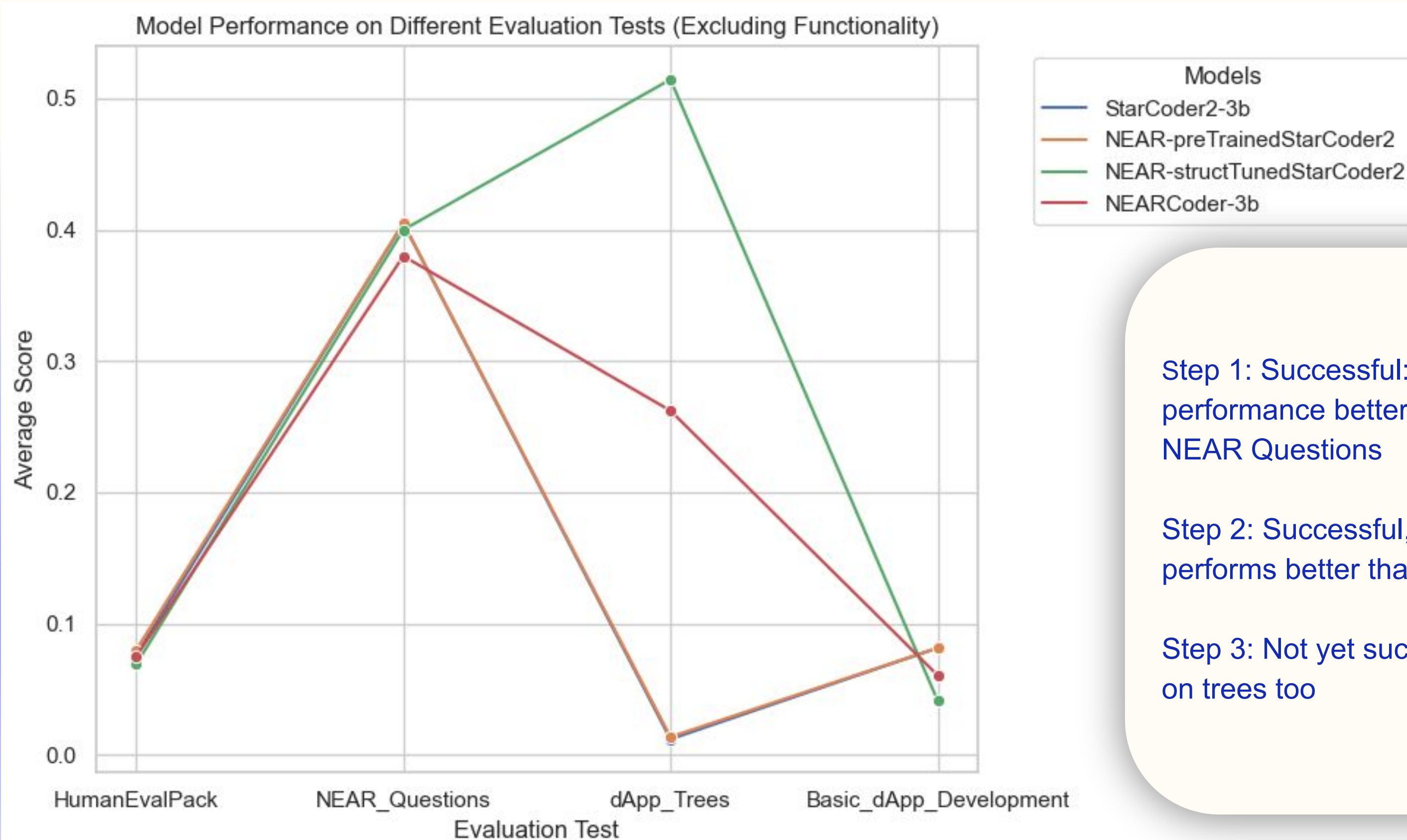
Evaluation Metrics:

- **HumanEval:** BLEU score.
- **NEAR Questions:** BLEU, ROUGE-1.
- **dApp Structures:** Structural similarity, comment quality.
- **Basic NEAR dApp:** Functionality, usability, and code quality.

Evaluation.

	model	humanEval_bleu	nearQuestions_bleu	\
0	StarCoder2-3b	0.076	0.53	
1	NEAR-preTrainedStarCoder2	0.080	0.52	
2	NEAR-structTunedStarCoder2	0.069	0.54	
3	NEARCoder-3b	0.075	0.49	
	nearQuestions_rouge1	dAppTrees_structure	dAppTrees_comment	\
0	0.28	0.0240	0.0	
1	0.29	0.0275	0.0	
2	0.26	0.0290	1.0	
3	0.27	0.0245	0.5	
	dAppGen_functionality	dAppGen_usability	dAppGen_quality	
0	0.0	0.08	0.0830	
1	0.0	0.08	0.0830	
2	0.0	0.00	0.0830	
3	0.0	0.00	0.1215	

Evaluation.



Step 1: Successful: NEARpreTrained performance better than Baseline model on NEAR Questions

Step 2: Successful, because NEARstruct performs better than Baseline on Trees

Step 3: Not yet successful, losing accuracy on trees too

Conclusion.

- **Skills unaffected by our training:**

General coding challenge

General NEAR knowledge ~

- **Skills improved by our training:**

Structure tuning improves dApp tree generation

- **Skills decreased by our training:**

AST-segments diminishes dApp development

Hypothesis for improvement:

- Further improve the AST-segments dataset
- AST-segments as pre-training, not fine tuning
- StarCoder2-15b-instruct in place of -3b
- Develop suitable evaluation metric for coding



Thanks

Julien Carbonnell
Anna-Valentina Hirsch
Kristian Boroz

