CS7470 Seminar in Programming Languages

Spring 2024

Organization

The goal of the seminar is to guide you through completing a project that uses a *large language model trained on code* (Code LLM) in a novel way. I strongly encourage you to work in a team and not try to tackle any research project alone. (This is a general recommendation, and not about this seminar.) We will conduct the seminar in three parts:

1. We will read papers to understand how Code LLMs are evaluated. Papers that evaluate benchmarks sometimes make overly general claims, such as "we evaluate reasoning" or "we evaluate programming". So, we will dig deeper and read the evaluation problems when necessary. Our objective will be to identify interesting gaps in evaluation.
2. We will build an evaluation set and benchmarking infrastructure for the new task that we identified in Part 1. The goal is to come up with problems that span a range of difficulties: from problems that are impossible for the most capable models, to those that are easy for small models. However, we have to convincingly argue that the hard problems are not impossible.
3. We will work on fine-tuning Code LLMs to do better on your task. We will build a fine-tuning set—either natural or synthetic—and use it to fine-tune and evaluate models of various sizes.

I hope that this three-part format will be a good fit for most projects. If you have something in mind that doesn't lend itself to this format, send me a written proposal and we can discuss it.

Time and Location

Tuesdays and Fridays, 1:35PM – 3:15PM. Hayden Hall 321

Prerequisites

There are no formal prerequisites. However, you must have experience doing research in either programming languages, software engineering, or machine learning. Undergraduate students require permission of the instructor. Graduate students were supposed to be able to enroll themselves, but need to ask for an override because Graduate Programming Languages was mistakenly added as a prerequisite.

Deadlines

See the schedule below for project writeup/presentation deadlines. There is reading assigned to most classes. Before each class you must:

1. Do the reading and be prepared for in-class discussion.
2. Submit a brief written reflection on each paper that you will share with the whole class. Being able to write quickly and fearlessly is an important skill.

Schedule

Notes:

1. I've included OpenReview links for many papers, where you can read the discussion between authors and reviewers. You do not have to read this discussion, but it can be helpful. However, *I*

strongly recommend you form your own opinion about the paper before reading the discussion on OpenReview.

2. I've ordered the reading for each day in a sequence that I think makes the most sense for reading. We'll follow this order for our in-class discussion.

Tuesday, Jan 9	**Introduction**
Friday, Jan 12	**Generating code from natural language** 1. Evaluating Large Language Models Trained on Code [arXiv] 2. MultiPL-E: A Scalable and Polyglot Approach to Benchmarking Neural Code Generation [IEEE] 3. Execution-based Evaluation for Open-Domain Code Generation [Open-Review]
Tuesday, Jan 16	**Generating tests from code (and natural language)** 1. An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation [IEEE] 2. Can Large Language Models Write Good Property-based Tests? [arXiv, demo]
Friday, Jan 19	**Finding and fixing bugs** 1. Generating High-Precision Feedback for Programming Syntax Errors using Large Language Models [arXiv] 2. Understanding the Effectiveness of Large Language Models in Detecting Security Vulnerabilities [arXiv] 3. Large Language Models are Few-Shot Testers: Exploring LLM-Based General Bug Reproduction [ACM]
Tuesday, Jan 23	**Instruction tuning** 1. Finetuned Language Models are Zero Shot Learners [OpenReview] 2. LIMA: Less Is More for Alignment [OpenReview] 3. OctoPack: Instruction Tuning Code Large Language Models [Open-Review, GitHub]
Friday, Jan 26	**In-class hacking**
Tuesday, Jan 30	**Editing code** 1. CoditT5: Pretraining for Source Code and Natural Language Editing [ACM] 2. InstructCoder: Empowering Language Models to Edit Code [OpenReview] 3. Can It Edit? Evaluating the Ability of Large Language Models to Follow Code Editing Instructions [arXiv]
Friday, Feb 2	**Training Code LLMs** 1. CodeParrot [original blog post, Hugging Face NLP course] 2. InCoder [OpenReview] 3. SantaCoder [arXiv][1] 4. StarCoder [OpenReview]

[1] The OpenReview page for SantaCoder is private by mistake. However, the paper was lightly reviewed for a work-shop and the reviews won't help our discussion.

Tuesday, Feb 6	**Project Proposals and Discussion** Your team should prepare a written proposal that discusses the following: 1. What is the task that you want to evaluate, and why does it matter? 2. What is the related work, and why is your project novel? 3. Is there a reasonable approach to automated evaluation? 4. Is there preliminary evidence that the task can be hard for state-of-the-art LLMs? 5. Do you have any ideas on how to build a fine-tuning set for the task? 6. How many problems does your evaluation set need to be reasonable?
Friday, Feb 9	**Self-instruction** 1. Self-Instruct: Aligning Language Models with Self-Generated Instructions [arXiv] 2. WizardCoder [OpenReview, arXiv] 3. Magicoder: Source Code Is All You Need [arXiv]
Tuesday, Feb 13	**Snow Day?**
Friday, Feb 16	**Not exactly self-instruction** 1. Self-alignment with instruction backtranslation [OpenReview] 2. Toolformer: Language Models Can Teach Themselves to Use Tools [Open-Review] 3. Knowledge Transfer from High-Resource to Low-Resource Programming Languages for Code LLMs [arXiv]
Tuesday, Feb 20	**Beyond pretraining and instruction finetuning** 1. Coarse-Tuning Models of Code with Reinforcement Learning Feed-back [OpenReview, arXiv] 2. Code Translation with Compiler Representations [OpenReview]
Friday, Feb 23	**Using Code LLMs for non-code tasks** 1. PAL: Program-aided language models [PMLR] 2. Symbolic planning and code generation for grounded dialogue [Open-Review]
Tuesday, Feb 27	**Class Cancelled (Arjun away)**
Friday, Mar 1	**Miscellaneous** 1. ADsafety: Type-based Verification of JavaScript Sandboxing [USE-NIX] 2. LEVER: Learning to Verify Code Generation with Execution [PMLR]
Tuesday, Mar 5	**Spring Break**
Friday, Mar 8	**Spring Break**

Tuesday, Mar 12	**Project Presentations: Benchmark Problems and Results** Your team should prepare a presentation on the following. 1. Example problems from your benchmark. Include both easy and hard problems. 2. How you're automating the evaluation. Discuss all manual steps. 3. How long it takes your benchmark to run. 4. Evaluation results on three model sizes. I recommend 1B, 3B, and 7B. You can use larger models if you're able to wrangle the resources to do so. **Maybe fine-tuning tutorial**
Friday, Mar 15	**Class cancelled for PhD visit day.** Go meet your prospective colleagues.
Tuesday, Mar 19	**Class Cancelled (Arjun away)**
Friday, Mar 22	**1. Two Late Presentations** **2. Fine-tuning tutorial**
Tuesday, Mar 26	**Early Work** 1. On the naturalness of software [IEEE] 2. Predicting Program Properties from "Big Code" [ACM]
Friday, Mar 29	**Early Work** 1. Code completion with statistical language models [ACM] 2. DeepBugs: a learning approach to name-based bug detection [ACM]
Tuesday, Apr 2	**Grab Bag 0** 1. **(Old Paper)** Automatic patch generation by learning correct code [ACM] 2. **(New Paper)** CruxEval: A Benchmark for Code Reasoning, Understanding, and Execution [Twitter]
Friday, Apr 5	**Grab Bag 1** 1. A structural model for contextual code changes [ACM] 2. Seq2Parse: neurosymbolic parse error repair [ACM] 3. Code2vec: learning distributed representations of code [ACM]
Tuesday, Apr 9	**Humans and Code LLMs** 1. Grounded Copilot: How Programmers Interact with Code-Generating Models [ACM] 2. How Beginning Programmers and Code LLMs (Mis)read Each Other [arXiv] 3. CodeCompose: A Large-Scale Industrial Deployment of AI-assisted Code Authoring [arXiv]
Friday, Apr 12	**Papers from Spring 2024** 1. NoFunEval: Funny how Code LMs Falter on Requirements Beyond Functional Correctness [Twitter] 2. OS-Copilot [Twitter] 3. Building LLMs for Code Repair [Blog Post]
Tuesday, Apr 16	**Last Class. Project Presentations: Finetuning Approach and Results** Your team should prepare a written proposal that discusses the following: 1. How will you build a fine-tuning dataset for your task? 2. How much data do you think you will need? This should be supported by evidence from the papers that you've read. 3. What is the minimum context length that you need for your task?

	How much time will it take to train and evaluate the models that you have in mind?
Friday, Apr 26	**Project Reports Due**

Recommended Work

What follows is not homework. But, if you don't have any experience working with LLMs, you will find them helpful to do.

Learn to Use Discovery

You should get an account on the Discovery Cluster if you haven't already. Your advisor should be your account sponsor, and I can sponsor if needed. You should get familiar with launching and monitoring jobs on GPU nodes.

We expect most of the work for the seminar to be doable on the 32GB V100 GPUs, which are abundant on Discovery, so you should get familiar with those. Discovery has A100s, but they are usually oversubscribed.

I often use the following workflow to use a node interactively. I create an sbatch script that runs sleep infinity on a GPU node. For example, the following script reserves a V100 GPU for two hours:

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --output=/dev/null
#SBATCH --mem=40G
#SBATCH --export=ALL
#SBATCH --cpus-per-task=6
#SBATCH --time=2:00:00
#SBATCH --job-name=v100_shell
#SBATCH --gres=gpu:v100-sxm2:1
#SBATCH --partition=gpu
sleep infinity
```

I launch the script with sbatch and then use squeue until the node is allocated:

```
squeue -u $USER
```

I run squeue repeatedly until the job is marked "RUNNING" on a particular node. Squeue gives you the node name, and you can SSH into the node. Once I'm on the node, I use tmux. (Without tmux or screen, any program that you're running will be killed if you disconnect.)

It is also possible to use Visual Studio Code with Discovery. See the file /home/a.guha/TIPS on how to do so.

LLM Inference

You should get used to running inference with an LLM. Hugging Face models have little example scripts that you can use to get started. For example, SantaCoder has several example scripts that should work. On a GPU, you may find that vLLM is significantly faster than Hugging Face Transformers and supports many contemporary LLM architectures.