Fine-tuning a Code LLM on Custom Code on a single GPU

Includes code:
https://huggingface.co/learn/cookbook/en/fine_tuning_code_llm_on_single_gpu
Fine-tuning datasets – see a separate file

Fine-tuning SLM for code gen
https://predibase.com/blog/fine-tune-a-code-generation-llm-with-llama-2-for-less-than-the-cost-of-a

https://medium.com/@liana.napalkova/fine-tuning-small-language-models-practical-recommendations-68f32b0535ca

Coding Agents

Coding agent using SLMs from Scratch: https://blog.gopenai.com/build-react-agents-using-slms-from-scratch-a0336e99ba7c
Run A Small Language Model (SLM) Local & Offline:
https://cobusgreyling.medium.com/run-a-small-language-model-slm-local-offline-1f62a6cbdaef
Small Language Model improves performance for code generation using AI:
https://www.linkedin.com/pulse/model-size-affects-performance-code-generation-using-ai-yerramsetti-62syc/
https://blog.fabrichq.ai/large-language-models-for-code-generation-f95f93fe7de4
https://deepsense.ai/coding-agents-in-large-language-models/
https://deepsense.ai/building-coding-agents/
Autogen for multi-agents: https://medium.com/@nageshmashette32/autogen-ai-agents-framework-3ee68bab6355

Using LLMs / SLMs for code generation

Best SLMs: https://slashdot.org/software/small-language-models/

Best LLMs for coding: https://www.techradar.com/computing/artificial-intelligence/best-large-language-models-llms-for-coding

LLMs for Code Generation: A summary of the research on quality:
https://www.sonarsource.com/learn/llm-code-generation/

LLMs for code generation: https://blog.fabrichq.ai/large-language-models-for-code-generation-f95f93fe7de4

Papers with Code: https://paperswithcode.com/task/code-generation
Includes:

- 20 code generation benchmark **LEADERBOARDS** with competitive results, e.g. HumanEval, MBPP, etc.
- Code generation **datasets**
- **Libraries**
- Dozens of **papers**

Mistral AI Introduces Ministral 3B and 8B language models for Edge Computing with strong multilingual and reasoning performance.

UC Berkley course on LLM agents:
https://www.linkedin.com/posts/areganti_if-youre-thinking-of-using-llm-agents-activity-7243808799487135744-YokP?utm_source=share&utm_medium=member_desktop

Awesome code generation:
https://github.com/LIANGQINGYUAN/awesome_codegeneration
Awesome cod LLMsL: https://github.com/codefuse-ai/Awesome-Code-LLM

Andrew Ng's post on code generation (find link on LinkedIn):
https://x.com/AndrewYNg/status/1803835964604977663

https://blog.kartones.net/post/code-generation-papers/

Datasets (used in Praxis proposal for datasets): https://github.com/mlabonne/llm-datasets

Benefits of Automatic Code Generation (for so what)
Parent link: https://github.com/resources/articles/ai/what-is-ai-code-generation
Child link 1: https://github.blog/news-insights/research/research-how-github-copilot-helps-improve-developer-productivity/ (paper: https://dl.acm.org/doi/pdf/10.1145/3520312.3534864)
Child link 2: https://github.blog/news-insights/research/research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/

McKinsey study: Unleashing developer productivity with generative AI – productivity gains of almost 100% in regular code generation.
https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/unleashing-developer-productivity-with-generative-ai
https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai

Quantifying GitHub Copilot's impact on developer productivity and happiness:
https://github.blog/news-insights/research/research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/

https://www.softxjournal.com/article/S2352-7110(24)00048-7/fulltext (downloaded pdf)

What is AI code generation?
https://github.com/resources/articles/ai/what-is-ai-code-generation

Automatic Code Generation for High-Performance Graph Algorithms
Publisher: IEEE: https://github.com/resources/articles/ai/what-is-ai-code-generation

Automatically generate code and documentation using OpenAI CODEX
https://shashankprasanna.com/automatically-generate-code-and-documentation-using-openai-codex/

Praxis ideas

Efficiency vs. Performance in Agent-Driven Code Generation:

- **Research Focus**: Investigate how agents using mostly proprietary large language models (LLMs) differ from those using open-source small language models (SLMs) in generating efficient code. Measure the trade-offs between the quality of the generated code, computational cost, and memory usage. Cost efficiency and broader availability of AI: SLMs can be deployed in resource-constrained environments (Google colab vs. a powerful expensive GPU server or even a GPU cluster).
- **Research Questions**: What are the differences in the quality and efficiency of the code generated by agents using LLMs vs. SLMs? How can an agent adaptively switch between LLMs and SLMs depending on the complexity of the task? How effective are SLM-based agents in resource-constrained environments compared to LLM-based agents in the cloud? What strategies can agents use to balance the trade-offs between resource constraints and code generation quality?

Fine-tuning Small Language Model on domain-specific data will noticeably increase the maintainability index compared to using an LLM without fine-tuning.
Fine-tuning of LLM
Maintainability index
Set up experiments with the two LLMs and compare the maintainability indeces.
Adjusting Small Language Model parameters, such as temperature and top-p, will noticeably improve the cyclomatic complexity of auto-generated code.
Temperature, top-p

Cyclomatic complexity
Set up experiments when LLM parameters are adjusted and not adjusted and compare the cyclomatic complexity.
Multi-agent collaboration will lead to a greater number of tests passed compared to the reflection agentic workflow
Reflection, multi-agent collaboration agentic workflows
Number of tests passed
Set up experiments with two agentic workflows and compare the number of tests passed.

DEFINITIONS

https://en.wikipedia.org/wiki/Automatic_programming

Automatic programming

In computer science, **automatic programming**[1] is a type of computer programming in which some mechanism generates a computer program to allow human programmers to write the code at a higher abstraction level.

Program synthesis is one type of automatic programming where a procedure is created from scratch, based on mathematical requirements.

Origin

Mildred Koss, an early UNIVAC programmer, explains: "Writing machine code involved several tedious steps—breaking down a process into discrete instructions, assigning specific memory locations to all the commands, and managing the I/O buffers. After following these steps to implement mathematical routines, a sub-routine library, and sorting programs, our task was to look at the larger programming process. We needed to understand how we might reuse tested code and have the machine help in programming. As we programmed, we examined the process and tried to think of ways to abstract these steps to incorporate them into higher-level language. This led to the development of interpreters, assemblers, compilers, and generators—programs designed to operate on or produce other programs, that is, *automatic programming*."[3]

Generative programming

Generative programming and the related term meta-programming[4] are concepts whereby programs can be written "to manufacture software components in an automated way"[5] just as automation has improved "production of traditional commodities such as garments, automobiles, chemicals, and electronics."[6][7]

The goal is to improve programmer productivity.[8] It is often related to code-reuse topics such as component-based software engineering.

Source-code generation

Source-code generation is the process of generating source code based on a description of the problem[9] or an ontological model such as a template and is accomplished with a programming tool such as a template processor or an integrated development environment (IDE). These tools allow the generation of source code through any of various means.

Modern programming languages are well supported by tools like Json4Swift (Swift) and Json2Kotlin (Kotlin).

Programs that could generate COBOL code include:

- the DYL250/DYL260/DYL270/DYL280 series[10]
- Business Controls Corporation's SB-5
- Peat Marwick Mitchell's PMM2170 application-program-generator package

These application generators supported COBOL inserts and overrides.

A macro processor, such as the C preprocessor, which replaces patterns in source code according to relatively simple rules, is a simple form of source-code generator. Source-to-source code generation tools also exist.[11][12]

Large language models such as ChatGPT are capable of generating a program's source code from a description of the program given in a natural language.[13]
Many relational database systems provide a function that will export the content of the database as SQL data definition queries, which may then be executed to re-import the tables and their data, or migrate them to another RDBMS.

Low-code applications

[edit]

Main article: Low-code development platforms

A low-code development platform (LCDP) is software that provides an environment programmers use to create application software through graphical user interfaces and configuration instead of traditional computer programming.

See also

https://github.com/resources/articles/ai/what-is-ai-code-generation - READ USEFUL