1. LLM Evaluation

Leaderboards

For benchmarking the performance of various LLMs and SLMs on a specific task (code generation in our case).

- **MBPP Leaderboard with ALL models (including OpenAI):** https://paperswithcode.com/sota/code-generation-on-mbpp .
- **Big Code Models Leaderboard** evaluates on HumanEval (Python) + irrelevant for me MultiPL-E (C++, Java, and JavaScript): https://huggingface.co/spaces/bigcode/bigcode-models-leaderboard
- **EvalPlus** evaluates using HumanEval+ version 0.1.10; MBPP+ version 0.2.0. Models are ranked according to pass@1 using greedy decoding: https://evalplus.github.io/leaderboard.html
- **LiveCodeBench** - holistic and contamination-free evaluation of coding capabilities of LLMs: https://livecodebench.github.io/leaderboard.html. Optional – to estimate the usefulness for the Praxis.
- **CanAICode results**: https://huggingface.co/spaces/mike-ravkine/can-ai-code-results .
- **Awesome Code LLM:** https://github.com/huybery/Awesome-Code-LLM - **HumanEval & MBPP for ALL models**.
- **SEAL Leaderboard** – evaluates multiple coding languages: https://scale.com/leaderboard/coding. Optional – to estimate the usefulness for the Praxis
- **Vellum LLM Leaderboard** evaluates on HumanEval among other metrics: https://www.vellum.ai/llm-leaderboard. Optional – to estimate the usefulness for the Praxis

Code Generation Evaluation Datasets

Used to evaluate the quality of the generated code

1. **HumanEval:** popular benchmark for evaluating code generation models - contains programming problems with corresponding unit tests that can be used to verify the correctness of generated solutions.
   Description: The HumanEval dataset is a popular benchmark for evaluating code generation models. It contains programming problems with corresponding unit tests that can be used to verify the correctness of generated solutions.
   Unit Tests: Each problem in the HumanEval dataset is accompanied by unit tests that are used to check the functional correctness of the generated code.
   Usage: This dataset is widely used in the evaluation of large language models like OpenAI's Codex, which powers GitHub Copilot
   Source: https://github.com/openai/human-eval/tree/master/data

2. **MBPP (Mostly Basic Python Problems):** designed to evaluate code generation models on Python programming tasks. It consists of a large number of Python problems with a set of unit tests that assess the correctness of the generated Python code.
   Description: The MBPP dataset is designed to evaluate code generation models on Python programming tasks. It consists of a large number of Python problems with varying levels of difficulty.
   Unit Tests: Each problem in the MBPP dataset is accompanied by a set of unit tests that assess the correctness of the generated Python code.
   Usage: This dataset is particularly useful for evaluating models that generate Python code, ensuring they produce functionally correct solutions.
   Source: https://github.com/google-research/google-research/tree/master/mbpp

More Recent

3. LBPP
https://huggingface.co/datasets/CohereForAI/lbpp

4. BigCodeBench (2024)
https://huggingface.co/blog/leaderboard-bigcodebench
Several complexity levels, Source: started w/ODEX, then used GPT-4 + human validation to improve the dataset, verified by humans who solved these problems)
Blog: https://huggingface.co/blog/leaderboard-bigcodebench
Github: https://github.com/bigcode-project/bigcodebench
HF: https://huggingface.co/datasets/bigcode/bigcodebench
Leaderboard: https://huggingface.co/spaces/bigcode/bigcodebench-leaderboard

5. LiveCodeBench (2023-2024)
Holistic and **Contamination Free** Evaluation of LLMs for Code. LiveCodeBench collects new problems over time from contests across three competition platforms, namely LeetCode, AtCoder, and CodeForces. Currently, LiveCodeBench hosts four hundred high-quality coding problems that were published between May 2023 and May 2024.
Github: https://github.com/LiveCodeBench/livecodebench.github.io
HF: https://huggingface.co/livecodebench - see code generation and code generation lite
Paper: https://arxiv.org/abs/2403.07974
Leaderboard and code: https://livecodebench.github.io/ (large LLMs prevail)

6. TACO (Topics in Algorithmic COde generation) (2023)
HF: https://huggingface.co/datasets/BAAI/TACO
Github: https://github.com/flagopen/taco
Paper: https://arxiv.org/pdf/2312.14852
Other: https://paperswithcode.com/dataset/taco-topics-in-algorithmic-code-generation

Data source: coding challenges from Leetcode, HackerRank, HackerEarth, CodeChef, CodeForces, GeeksforGeeks, etc. We also integrated existing datasets, including APPS, CodeContest, and Description2code

Other Datasets

7. APPS (Automated Programming Progress Standard) – old, 4 years ago

- **Description:** The APPS dataset consists of a large set of programming problems, ranging from simple to complex, designed to evaluate code generation models. It covers multiple difficulty levels, from introductory to competitive programming challenges.
- **Unit Tests:** APPS includes test cases that serve as unit tests for the generated code. These tests ensure that the code meets the problem requirements and handles edge cases.
- **Usage:** This dataset is used to benchmark the performance of models across a wide range of programming tasks.
- https://github.com/hendrycks/apps

8. SPoC (Student Programming Contest)

- **Description:** SPoC is a dataset derived from student submissions in programming contests. It contains problems along with multiple solutions, including some that have been tested against unit tests.
- **Unit Tests:** While not all problems in SPoC come with unit tests, a significant portion does, allowing for the evaluation of code correctness.
- **Usage:** SPoC is often used to evaluate the ability of code generation models to handle real-world student programming problems.

9. CodeContest

- **Description:** CodeContest is a dataset comprising competitive programming problems, similar to those found on platforms like Codeforces and LeetCode.
- **Unit Tests:** Problems in this dataset come with comprehensive test cases, which act as unit tests to verify the correctness of the generated code.
- **Usage:** It is used to benchmark the performance of code generation models on competitive programming tasks.

10. MultiPL-E

https://huggingface.co/datasets/nuprl/MultiPL-E = HumanEval + MBPP translated into other programming languages

11. ODEX - https://arxiv.org/pdf/2212.10481 ,
https://github.com/zorazrw/odex?tab=readme-ov-file

Evaluation Code

- **My code** - Testing LLMs on the Code Generation Task: https://github.com/agnedil/Praxis
- **EvalPlus** evaluation code: https://github.com/evalplus/evalplus/

- **LiveCodeBench** evaluation code: https://github.com/LiveCodeBench/LiveCodeBench

2. LLM Fine-Tuning

Datasets to Fine-Tune LLMs on Code Generation

Used to fine-tune SLMs to improve their code generation capabilities

1. **Tested-143k-Python-Alpaca**

Description: Python dataset with 143,327 examples of code that passed automatic tests to ensure high quality.
Link: https://huggingface.co/datasets/Vezora/Tested-143k-Python-Alpaca

2. **CodeFeedback-Filtered-Instruction**

Description: a curated collection of code instruction queries extracted from open-source code instruction tuning datasets. It significantly advances code generation capabilities by integrating execution and iterative refinement functionalities.
Link: https://huggingface.co/datasets/m-a-p/CodeFeedback-Filtered-Instruction

3. **Magicoder-Evol-Instruct-110K**

Description: A decontaminated version of evol-codealpaca-v1. Decontamination was done in the same way as StarCoder (bigcode decontamination process). See Magicoder paper.
Link: https://huggingface.co/datasets/ise-uiuc/Magicoder-Evol-Instruct-110K

4. **Python-code-dataset-500k**

Description: a summary and reformat pulled from GitHub code. 500K examples to be cleaned first. Cleaning can be done using an SLM.
Link: https://huggingface.co/datasets/jtatman/python-code-dataset-500k

5. **Just-write-the-code-Python-GenAI-143k**

Description: The entire dataset of 230k examples of AI and Machine Learning python code retrieved from public repositories on GitHub. It is a prototype and needs to be cleaned.
Link: https://huggingface.co/datasets/guidevit/Just-write-the-code-Python-GenAI-143k and https://huggingface.co/datasets/guidevit/Just-write-the-code-Python-GenAI-230k

6. **Tiny codes**

1.6 M short and clear code snippets that can help LLM models learn how to reason with both natural and programming languages.
Link: https://www.sonarsource.com/learn/llm-code-generation/

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

3. Small Language Models

In addition to the ones whose results are submitted.

1. **DeepSeek-Coder-V2** – among the best small language models for code generation. Description and usage examples: https://github.com/deepseek-ai/DeepSeek-Coder-V2.
2. CodeGemma-7B-it (HumanEval – 61%)
3. **SmolLM**

Code

- Main repo that I am using: https://github.com/openai/human-eval
- May be helpful: https://github.com/abacaj/code-eval/tree/main
- Leaderboard: https://huggingface.co/spaces/bigcode/bigcode-models-leaderboard
- Pass @k explained: https://deepgram.com/learn/humaneval-llm-benchmark
- Replicate LLMs: https://replicate.com/pricing + deploy your own model.

```
# pip install -q datasets
from datasets import load_dataset
# Languages: "python", "js", "java", "go", "cpp", "rust"
ds = load_dataset("bigcode/humanevalpack", "python")["test"]
ds[0]
```

4. Agents and Advanced LLM Techniques

ADVANCED LLM TECHNIQUES

LLM agents and multi-agent systems: https://www.linkedin.com/posts/areganti_if-you-want-to-learn-how-to-build-llm-activity-7247795075475193857-b2iR/
OpenAI lightweight agentic library SWARM: https://www.linkedin.com/posts/philipp-schmid-a6a2bb196_this-came-unexpected-openai-released-swarm-activity-7250841965519368192-oJ35/
https://github.com/andrewyng/aisuite - aisuite makes it easy for developers to use multiple LLM through a standardized interface
Install LLMs locally: https://www.iphones.ru/iNotes/kak-ustanovit-neyroset-tipa-chatgpt-pryamo-na-mac-i-polzovatsya-ey-bez-interneta-besplatno

Hugging Face Releases SmolTools: A Collection of Lightweight AI-Powered Tools Built with LLaMA.cpp and Small Language Models:

https://www.marktechpost.com/2024/11/06/hugging-face-releases-smoltools-a-collection-of-lightweight-ai-powered-tools-built-with-llama-cpp-and-small-language-models/

LLM on CPU: https://www.linkedin.com/posts/gabriele-venturi_ai-localllm-1bit-ugcPost-7252959485252554752-8dnk/

DeepLearningAI: GenAI for SWE

Skill Certificate by DeepLearningAI
https://www.coursera.org/professional-certificates/generative-ai-for-software-development

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

AGENTS

Some useful info about SWE agents:
https://arjunbansal.substack.com/p/how-do-i-evaluate-llm-coding-agents

FREE ONLINE COURSES TO BUILD AGENTS:
https://www.linkedin.com/posts/areganti_agents-tools-llms-activity-7206120480821379072-cZuL/
Qwen agent: https://github.com/QwenLM/Qwen-Agent

Recommended agentic library: https://www.linkedin.com/posts/mehdiallahyari_multi-agent-rag-system-activity-7273554473375940608-mHhf?utm_source=share&utm_medium=member_desktop
Another one talked about:
https://www.linkedin.com/feed/update/urn:li:activity:7273016722708643841?utm_source=share&utm_medium=member_desktop
https://ai.pydantic.dev/#instrumentation-with-pydantic-logfire
Open-source agents for developers: https://www.all-hands.dev/

Agents Links

- Agentic AI: Challenges and Opportunities
- Advanced Llama 3 agent (post), code, video
- Agentic RAG
- Autodoc: https://github.com/context-labs/autodoc
- Internet of Agents: Weaving a Web of Heterogeneous Agents for Collaborative Intelligence - https://arxiv.org/abs/2407.07061v2
- Automating Leetcode solutions with Claude 3.5: https://www.reddit.com/r/leetcode/comments/1ex7a1k/i_automated_leetcode_using_claudes_35_sonnet_api/

Agent links that can't be downloaded

LLM Agent Paper List (Dozens!)
https://github.com/WooooDyy/LLM-Agent-Paper-List

LLM Agent Papers
https://github.com/AGI-Edgerunners/LLM-Agents-Papers

LLMs for education (article)
https://www.packtpub.com/article-hub/large-language-models-llms-in-education

Benefits of LLMs in education
https://publish.illinois.edu/teaching-learninghub-byjen/benefits-of-llms-in-education/

AppAgent: Multimodal Agents as Smartphone Users
https://appagent-official.github.io/ (this is a general description - paper downloaded too)

Autonomous Agents and Multi-Agent Systems
https://link.springer.com/journal/10458

Exclusive: Google finds AI agents pose fresh ethical challenges:
https://www.axios.com/2024/04/19/ai-agents-assistants-ethics-alignment-google

Computational Agents Exhibit Believable Humanlike Behavior (Stanford)
https://hai.stanford.edu/news/computational-agents-exhibit-believable-humanlike-behavior

Introduction to Agents: Understanding Agent Environment in AI:
https://www.kdnuggets.com/2022/05/understanding-agent-environment-ai.html

Multi-Agent Simulation
https://fablestudio.github.io/showrunner-agents/

A generalist AI agent for 3D virtual environments
https://deepmind.google/discover/blog/sima-generalist-ai-agent-for-3d-virtual-environments/

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.

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/

Most Effective LLM Agent Architectures

By Aishwarya Naresh Reganti (https://www.linkedin.com/posts/areganti_what-are-the-most-effective-llm-agent-activity-7277888363607539713-HxCz?utm_source=share&utm_medium=member_desktop)

Here are the key design patterns:
⛳ Delegation: Reduce latency (but not cost) by running agents in parallel.
⛳ Parallelization: Use cheaper, faster models for cost and speed gains
⛳ Specialization: A generalist agent orchestrates while specialists execute tasks. For example, the main agent employs a specifically prompted (or fine-tuned) medical model for health queries or a legal model for legal questions.
⛳ Debate: Multiple agents with different roles engage in discussion to reach better decisions
⛳Tool Suite Experts: When using hundreds or thousands of tools, specialize agents in specific tool subsets.

Parallelization
Reduce latency (but not cost) by running agents in parallel.
For example, 100 sub-agents can each read a different chapter of a book and return key passages.

Delegation
Use cheaper, faster models for cost and speed gains.
For example, the best performing model can delegate to a cheap/fast model to read a book

and return relevant passages. This works well if the task description and result are more compact than the full context.

Specialization
A generalist agent orchestrates while specialists execute tasks.
For example, the main agent employs a specifically prompted (or fine-tuned) medical model for health queries or a legal model for legal questions.

Debate
Multiple agents with different roles engage in discussion to reach better decisions.
For example, a software engineer proposes code, a security engineer reviews it, a product manager provides the user's perspective, and a final agent synthesizes the information and makes a decision.

Tool Suite Experts
When using hundreds or thousands of tools, specialize agents in specific tool subsets.
Each specialist (the same model with different tools) manages a specific toolset. The orchestrator then directs tasks to the appropriate specialist, keeping the orchestrator's prompt concise.

The design patterns are adapted from Anthropic's X post on agent and tool use architectures in practice.
I've modified it to be more general and not specific to Anthropic models.
Here's the link: https://x.com/alexalbert__/status/1796211979121840409

5. Agent Implementation

LangChain Reflexion:

- https://medium.com/aimonks/reflection-agents-with-langgraph-agentic-llm-based-applications-87e43c27adc7 (A LOT MORE DETAILS)
- https://blog.langchain.dev/reflection-agents/
- FIND OTHER EXAMPLES

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