**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+](https://github.com/evalplus/humanevalplus_release) version 0.1.10; [MBPP+](https://github.com/evalplus/mbppplus_release) 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

**Autogenerated Code 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>

1. **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>

1. **LiveCodeBench Dataset:** holistic and contamination-free evaluation of coding capabilities of LLMs  
   Source: <https://huggingface.co/livecodebench>

**4. APPS (Automated Programming Progress Standard)**

* **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.

**5. 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.

**6. 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.

**7. MultiPL-E**

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

**Evaluation Code**

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

**Training Datasets**

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>

1. **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>

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

Description: A decontaminated version of [evol-codealpaca-v1](https://huggingface.co/datasets/theblackcat102/evol-codealpaca-v1). Decontamination was done in the same way as StarCoder ([bigcode decontamination process](https://github.com/bigcode-project/bigcode-dataset/tree/main/decontamination)). See [Magicoder paper](https://arxiv.org/abs/2312.02120).

Link: <https://huggingface.co/datasets/ise-uiuc/Magicoder-Evol-Instruct-110K>

1. **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>

1. **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>

1. **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/>

**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](https://huggingface.co/google/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]

**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>

**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](https://gradientflow.substack.com/p/agentic-ai-challenges-and-opportunities?isFreemail=true&post_id=144412548&publication_id=20983&r=wakvt&triedRedirect=true&triggerShare=true)
* [Advanced Llama 3 agent](https://www.linkedin.com/posts/langchain_creating-an-ai-agent-with-langgraph-llama-activity-7197629964672798720-7Rpr?utm_source=share&utm_medium=member_desktop) (post), [code](https://github.com/samwit/langchain-tutorials), [video](https://www.youtube.com/watch?v=lvQ96Ssesfk)
* [Agentic RAG](https://www.linkedin.com/posts/yujiantang_forget-rag-and-ai-agents-theres-a-new-cool-activity-7197325927742070784-AW2r?utm_source=share&utm_medium=member_desktop)
* 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**](https://github.com/WooooDyy/LLM-Agent-Paper-List)

**LLM Agent Papers**

[**https://github.com/AGI-Edgerunners/LLM-Agents-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)**

<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/>

**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