**Code Generation Results: HumanEval, MBPP  
SLMs without fine-tuning**   
January 11, 2024

1. **Code used to generate the below summary**

Code: <https://github.com/agnedil/code-generation>

1. **Challenges**

* The Replicate API library would not work directly, so I had to use its version within another library – LangChain.
* I used the code from this repo for HumanEval evaluation of all models: <https://github.com/openai/human-eval>. The multiprocessing module executed with an error: "AttributeError: Can't pickle local object in Multiprocessing". **I had to modify the original code to fix it**.
* The final check of the code correctness in the original OpenAI code is done by combining the problem (otherwise called starter code or function docstring) and completion: **problem["prompt"] + completion** which means the completion shouldn’t contain the problem. Two issues: 1) incorrect indentation when joining the problem and completion – the code generates error when running, 2) some models may still misunderstand and include the problem into the completion (Llama 3 Instruct trained for chat, does it for 33% of cases). Therefore, I am using an additional prompt to ask LLMs to **include the problem definition (function docstring) into the completion**, and I exclude **problem["prompt"]** from the evaluated string. **I had to modify** [**the original code**](https://github.com/agnedil/Praxis) **to fix this**.
* **Summary of code modifications** (all in execution.py):
  + **Add class DillProcess** to fix the pickling issue (uses dill instead of pickle).
  + **Modify function check\_correctness()** to have an extra argument use\_prompt which controls the exclusion or addition of problem["prompt"] to the Python program to be run. The body of the function is modified to accommodate for the use of use\_prompt.
  + Modify exception handling to **add error tracebacks** (helps when the error message is empty).
* SLMs tend to output **additional explanations** and clarifications like: “Here is the requested code completion:” etc. which break the automatic code execution during the verification stage. Adding more specific instructions like: “Complete the following code. Output only the runnable code and nothing else:” would still lead to non-runnable content like triple backticks in the output. As a result – **I had to provide minimum help to some models** by removing the initial or trailing triple backticks or strings like “```python”. Or by adding “from typing import List” as this was removed in the process (when LLM forgets to include it into the repeated func definition)
* **General approach to evaluate all models**: create one extensive and comprehensive prompt for all models. If any model fails to fully understand it and outputs code with human phrases or non-runnable symbols, it should be considered the drawback of the model.

1. **Results**

* **Llama 3 8B** – promising results.
* Non-chat optimized model ”**meta/meta-llama-3-8b**” - several cases of hallucinations when functions are repeated and the code is incomplete in the end (stops at the middle of a function). See Appendix
* **Nous-hermes-2-solar-10.7b** – tries to explain the solution if no prompt is used (func docstring as prompt) – not runnable. 25.61% when using a prompt.
* **Gemma 7B** – incomprehensible output whether I include the prompt prefix or not.
* **Code Gemma 7b IT** (when asked to output the full func for HumanEval) – a) code generation template (per HG docs): unusable output – patchy pieces of code, sometimes 1 or 2 random lines, b) chat generation template: more usable output, but still a lot of errors in the first 5 problems: repeats def in the end, 2 out of 3 outputs were completions w/out func signature and docstring, 2 others contained extraneous text, e.g. the word “def” after the func was already provided, etc. Decided not to waste compute units – the leaderboard performance is still only 55%.
* **Phixtral** – generates code, but contains extraneous text (Here is a solution). If I do additional post-processing, this will be a disadvantage for other models. Also, it is very slow – up to 2 minutes per test case (5 hours for the entire run)
* **GPT-J-6B** – not fit for the task as the model is too weak, outputs hallucinations that remind of the expected output only very remotely (trained in 2021).
* **Yi-6B** is a bilingual (Chinese) model – pass@1 = 3% if not using prompt (function docstring as prompt), otherwise if using a prompt the model outputs some irrelevant snippets of code and. Asking to output the starter code concatenated with the completion doesn’t help – the output still includes the completion without the beginning in most cases (<https://huggingface.co/01-ai/Yi-6B> ).
* **Flan-T5** outputs complete nonsense that resembles code – completely not runnable.
* **Phi** – not designed for code completion. Outputs incomprehensible combinations of letters (“em”, “emlen”, “A”, “A.A.A.A.”, etc.) as generated code with or without a prompt (if with prompt, the model repeats the entire prompt before the incomprehensible output).
* **Phixtral-2x2\_8** – MoE of two Phi models (4.5B), follows the instruction much better than Phi, reached Pass@1 = ~15%. Still outputs irrelevant human-like output although asked specifically not to do that: e.g. here’s the code, here’s the concatenated code, etc. Also, it takes ~1 min per API call which is a lot, considering there are 500 data points in the MBPP dataset.
* **Qwen1.5-7b** (replicate.com) – demonstrated a good result on HumanEval Pass@1 at ~44%, but only 20% on MBPP. The main challenge with this model is that it takes 200-300 s per one API call - took 1 day to run MBPP on replicate. This is unacceptable for experiments with agents as I will have to make several API calls per one agent call + run this for all 500 MBPP data points again – will take more than a day per experiment.
* **Mistral 7B** provided the expected result. The model would strip any docstrings from the functions – only the definition def was left. I helped the model by removing triple backticks from start / end, “```python”, and adding “from typing import List” because the model would strip this import most of the times while the import is specific to the HumanEval dataset.
* **Codestral Mamba** – showed the best result on my leaderboard, followed by **Ministral 8B** and, surprisingly, **Ministral 3B**. The latter is the smallest model that I tried, but it outperformed many other models that are 2 to 2.5 times bigger, and even the models with 22B parameters (7+ times bigger). Other models from the Mistral family also showed good results which, on average, make this group of models as the leading one among all other models.
* **OpenCodeInterpreter-DS-6.7B** and **Artigenz-Coder-DS-6.7B** – when asked to output the entire function, keeps saying “Here is the completed function” (even if I ask not to do it in the prompt). Model needs extra help by getting the code placed between *```python* and *```*. **May be better at pure code completion**?
* **Mamba 2.8B (**replicate.com): if not using a prompt (func docstring as prompt) – the model tries to generate a completion, but then follows a paragraph of hallucinations that look like human free-form text with how-to questions about software development. When using a prompt – the model doesn’t even try to complete the code – it starts hallucinating right away (see saved file with examples).
* Gemma 7B, Gemma 2B, Flan-T5, Phi, Mamba 2.8B (replicate.com) – incoherent output.
* **Deepseek-Coder-6.7B-Instruct** – scored great on HumanEval, but *did only 1% on MBPP*, mainly because the model outputs unnecessary explanations, although it is explicitly asked not to do that. Example: “Sure, here is the Python function that calculates.” This is done for every data point. Somewhat similar numbers are for OpenCodeInterpreter-DS-6.7B. Reason is same: unnecessary clarifications when asked not to do it: “Here is the Python function that satisfies the given tests:” *Solution – maybe decrease temperature*?
* **Llama 3**.1 8B Instruct – released fall 2024. Inference takes an average of 2 minutes for Human Eval and 0.75 min for MBPP. Both tasks required 4 hours to finish running in Google Colab on an A100 GPU which is the best available. This is too long for subsequent experiments.

All models received slight help by stripping ``` backticks at edges including the ```python string + adding “from typing import List” which is often stripped by SLMs.

The pass@1 scores below are provided for my run (first number) and then for the result shown on the Big Code Leaderboard (second number), if it is available: <https://huggingface.co/spaces/bigcode/bigcode-models-leaderboard>

***Table 1. Prompt Asking to Return a Complete Function***

| **Model** | **Hosted By** | **Model Size** | **Human-Eval Full Func**  **Pass@1** (Me / Big Code) | **H-E Compl** | **MBPP** | **LBPP** | **Big Code Bench** | **Average** | **Temp / top\_p** |  | **Cost (USD)**  **Full Func** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Small Language Models (SLMs)** | | | | | | | | | | | |
| Nxcode-CQ-7B-orpo | Google Colab | 7.25B | 82.93 / 87.23 | 75.61% | 73% | 22.84% |  |  | 1.0 / 1.0 |  | $50/month |
| Codestral Mamba | mistral.ai | 7.3B | 75.61% / 75% | 60.37% | 39.4% |  |  |  | 0.7 / 1.0 |  | 0.02 |
| Ministral 8B | mistral.ai | 8B | 72.56% / 76.8% (instruct) | 71.34% | 56.2% |  |  |  | 0.3 / 1.0 |  | 0.01 |
| Deepseek-Coder-6.7B-Instruct | Google Colab |  | 65.24% / 80.22% | 70.73% | 1% | 0% (extra words!) |  |  | 1.0 / 1.0 |  | $50/m |
| Ministral 3B | mistral.ai | 3B | 64.63% / 77.4% (instruct) | 61.59% / 77.4% (instruct) | 51.8% |  |  |  | 0.3 / 1.0 |  | 0.01 |
| Mistral-Nemo-Instruct-2407 | mistral.ai | 12B | 58.54%/ 67% | 53.05% | 47.4% |  |  |  | 0.3 / 1.0 |  | 0.01 |
| Llama 3 8B | Replicate | 8B | 51.83% / 45.65% | API Error | API Error |  |  |  | 0.95 / 1 |  | 0.29 |
| Llama 3.1 8B Instruct | Google Colab | 8B | 65.9% / 72.6% | 55.47% | 56.8% |  |  |  |  |  |  |
| CodeQwen1.5-7B-Chat | Google Colab |  | 50% / 87.2% | 54.88% | 55.2% |  |  |  | 1.0 / 1.0 |  | $50/m |
| Qwen1.5-7b | replicate.com | 7B | 43.9% | 200 s per API call | 19.4% |  |  |  | 0.95 / 1 |  | 3.55 |
| OpenCodeInterpreter-DS-6.7B | Google Colab |  | 41% / 73.2% | 71.95 / 73.2% | 5.4% |  |  |  | 1.0 / 1.0 |  | $50/m |
| Mistral 7B | mistral.ai | 7B | 31.1% / 30.5% | 35.98% / 30.5% | 13.6% |  |  |  | 0.7 / 1.0 |  | 0.01 |
| Nous-hermes-2-solar-10.7b | replicate.com | 10.7B | 25.61% | 28.65% | 30.4% |  |  |  | 0.95 / 1 |  | 0.61 |
| Phixtral-2x2\_8 (4.5B) | replicate.com | 4.5B | 14.64% | 34.756% | 14.6% |  |  |  | 0.95 / 1 |  | 2.77 |
| Artigenz-Coder-DS-6.7B | Google Colab |  | 1.22% / 70.89% | 73.17% | 0.2% |  |  |  | 1.0 / 1.0 |  | $50/m. |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Slightly Bigger SLMs** | | | | | | | | | | | |
| Mistral-Small-2409 | mistral.ai | 22B | 70.73% / 80% | 64.63% | 60.2% |  |  |  | 0.7 / 1.0 |  | 0.03 |
| Codestral latest | mistral.ai | 22.2B | 26.83% / 81.1% | 64.63% | 37% |  |  |  | 0.7 / 1.0 |  | 0.15 |
| Mixtral-8x7B-v0.1 | mistral.ai | 12 active (47 total) | 16.46% / 40.2% | 33.54% | 0% |  |  |  | 0.7 / 1.0 |  | 0.05 |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Not Useful SLMs** | | | | | | | | | | | |
| Yi 6B | replicate.com | 6B | 3% | 3% | 0.2% |  |  |  | 0.95 / 1 |  | 0.44 |
| Code Gemma 7b IT | Google Colab | 7B | 0% (? model) | 27.44% | 51% (chat model) |  |  |  | 1.0 / 1.0 |  |  |
| Gemma 7B | replicate.com | 7B | 0 % | 0% | 0% |  |  | 0% | 0.95 / 1 |  | 0.05 |
| Gemma 2B | replicate.com | 2B | 0 % | 4.26% | 4.6% |  |  |  | 0.95 / 1 |  | 0.05 |
| Flan-T5 | replicate.com |  | 0% |  |  |  |  |  | 0.95 / 1 |  |  |
| Phi | replicate.com |  | 0% | Abracadabra (even if temp=0.25 |  |  |  |  | 0.95 / 1 |  |  |
| Mamba 2.8B | replicate.com | 2.8B | n/a | Abracadabra (even if temp=0.25 | 0% |  |  | 0% | 0.95 / 1 |  | 0.02 (20 calls) |
|  |  |  |  |  |  |  |  |  |  |  |  |

HuggingFace transformer models’ default temperature and top\_p parameters are explained here: <https://huggingface.co/docs/transformers/v4.22.2/en/main_classes/text_generation>

Usually they are 1.0 and 1.0, respectively, and can be checked by running model.config.temperature and model.config.top\_p.

1. **Conclusions**

* **Nxcode-CQ-7B-orpo** consistently stands out with **~83–87%** on HumanEval and **73%** on MBPP among 7–8B models in both **HumanEval** and **MBPP**.
* **Mistral/Ministral** families cluster around **65–76%** pass@1 on HumanEval, with MBPP typically in the **50–60%** range—respectable but trailing Nxcode-CQ.
* **Larger parameter counts** do **not** always guarantee higher pass@1!
* A handful of models do far worse, often failing to solve any tasks on one or both benchmarks.

**Top Performers (SLMs)**

* **Nxcode-CQ-7B-orpo**
  + **HumanEval**: ~83% pass@1
  + **MBPP**: 73%
  + **Comments**: Among the small language models (7B range), Nxcode-CQ stands out for having both high HumanEval pass@1 scores (in the low-to-mid 80s) and a strong MBPP score of **73%**—the best overall in the table among the smaller models.
* **Mistral-based Models** (e.g., *Ministral 8B, Mistral 3B*)
  + **Ministral 8B**: 72.56%–76.8% pass@1 on HumanEval, 56.2% on MBPP
  + **Ministral 3B**: ~65%–77% pass@1 on HumanEval, 51.8% on MBPP
  + **Comments**: These show decent HumanEval performance. Their MBPP scores (in the **50–56%** range) are below Nxcode-CQ but still mid-tier among smaller LLMs.
* **Codestral Mamba (7.3B)**
  + **HumanEval**: ~75.6% pass@1
  + **MBPP**: 39.4%
  + **Comments**: Reasonably strong on HumanEval, but its MBPP score is comparatively lower than Nxcode-CQ and most Mistral-based models.
* **CodeQwen1.5-7B-Chat**
  + **HumanEval**: 50%
  + **MBPP**: ~55.2%

**Mid-Performers and Edge Cases**

* **Deepseek-Coder-6.7B-Instruct**
  + **HumanEval**: 65%
  + **MBPP**: ~1%
  + **Comments**: Shows large discrepancy: decent HumanEval performance but very low MBPP score (~1%). Possibly an instruction-tuning or prompt-format mismatch issue.
* **OpenCodeInterpreter-DS-6.7B**
  + **HumanEval**: ~41%
  + **MBPP**: ~5%
  + **Comments**: Another big gap between HumanEval and MBPP performance.
* **Qwen1.5-7B**
  + **HumanEval**: ~44%
  + **MBPP**: 19.4% (or ~14% in another setup)
  + **Comments**: Notable for the discrepancy between multiple test runs (possibly prompt formatting or code-execution differences).

**Mid-Size Models (10–22B)**

* **Mistral-Small-2409 (22B)**
  + **HumanEval**: ~70.7%
  + **MBPP**: 60.2%
  + **Comments**: Solid across both benchmarks, on par or slightly above many 7–8B models.
* **Codestral Latest (22.2B)**
  + **HumanEval**: 26.8%
  + **MBPP**: 37%
  + **Comments**: Performance is mid-range.

**Very Low Performers**

Several models yield 0–5% pass@1 on HumanEval or MBPP, including:

* **Yi 6B** (3% on HumanEval, 0.2% on MBPP)
* **Gemma** variants (often 0% on HumanEval, near 0%–5% on MBPP)
* **Flan-T5**, **Phi** (0% on given tasks in these tests)
* **Mamba 2.8B** (no data or 0% MBPP)

1. **Past and Next steps**

* Table contains much more data now.
* Finished the **first HumanEval run**.
* **Conducted the MBPP run** for all models – CONSIDERABLE EFFORT as the dataset has 500 data points which means the code needs to be generated and verified 500 times.
* I went over all the models and **added their default temperature and top\_p values** to the table (considerable effort). TODO: For the best and worst performing models – conduct another run with a **different temperature and top\_p settings**. This is one of my research hypothesis.
* Conduct the **second HumanEval run** based on completion without including the function signature – I heard the performance may be different
* According to (Matton et al. 2024), **data leakage** in code generation occurs when popular evaluation benchmarks (like HumanEval and MBPP) appear in a model’s training data and, whether intentionally or unintentionally, compromise the validity of test scores. Therefore, it may be reasonable to **test the models (and agents) on a much more recent dataset**. One dataset is proposed in the paper: <https://huggingface.co/datasets/CohereForAI/lbpp>
* Select **a few candidates** performing well on all datasets and start conducting the SLM fine-tuning effort and agent building experiments.
* Finish the **Methodology** section

**References**

Matton A., Tom Sherborne, Dennis Aumiller, Elena Tommasone, Milad Alizadeh, Jingyi He, Raymond Ma, Maxime Voisin, Ellen Gilsenan-McMahon, Matthias Gallé. 2024. **On Leakage of Code Generation Evaluation Datasets**.