**Code Generation Results - HumanEval  
SLMs without fine-tuning**   
December 7, 2024

Code: <https://github.com/agnedil/Praxis>

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) – 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.
* Qwen1.5-14b – demonstrated a good result of Pass@1 at ~44%.
* **Mamba 2.8B**: 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).
* **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**?

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**  **Pass@1** (Me / Big Code) | **MBPP** | **Average** | **Comments** | **Cost (USD)** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Small Language Models (SLMs)** | | | | | | | |
| Nxcode-CQ-7B-orpo | Google Colab | 7.25B | 82.93 / 87.23 |  |  |  | $50/month |
| Codestral Mamba | misral.ai | 7.3B | 75.61% / 75% |  |  |  | 0.02 |
| Ministral 8B | misral.ai | 8B | 72.56% / 76.8% (instruct) |  |  |  | 0.01 |
| Deepseek-Coder-6.7B-Instruct | Google Colab |  | 65.24% / 80.22% |  |  |  | $50/m |
| Ministral 3B | misral.ai | 3B | 64.63% / 77.4% (instruct) |  |  |  | 0.01 |
| Mistral-Nemo-Instruct-2407 | mistral.ai | 12B | 58.54%/ 67% |  |  |  | 0.01 |
| Llama 3 8B | replicate.com | 8B | 51.5% / 45.65% |  |  |  | 0.29 |
| CodeQwen1.5-7B-Chat | Google Colab |  | 50% / 87.2% |  |  |  | $50/m |
| Qwen1.5-14b | replicate.com | 14B | 43.9% |  |  | 200-300 s per one API call. | 3.55 |
| OpenCodeInterpreter-DS-6.7B | Google Colab |  | 41% / 73.2% |  |  |  | $50/m |
| Mistral 7B | mistral.ai | 7B | 31.1% / 30.5% |  |  |  | 0.01 |
| Nous-hermes-2-solar-10.7b | replicate.com | 10.7B | 25.61% |  |  |  | 0.61 |
| Phixtral-2x2\_8 (4.5B) | replicate.com | 4.5B | 14.64% |  |  | ~1 min per API call | 2.77 |
| Artigenz-Coder-DS-6.7B | Google Colab |  | 1.22% / 70.89% |  |  |  | $50/m. |
|  |  |  |  |  |  |  |  |
| **Slightly Bigger SLMs** | | | | | | | |
| Mistral-Small-2409 | misral.ai | 22B | 70.73% / 80% |  |  |  | 0.03 |
| Codestral latest | misral.ai | 22.2B | 26.83% / 81.1% |  |  |  | 0.15 |
| Mixtral-8x7B-v0.1 | misral.ai | 12 active (47 total) | 16.46% / 40.2% |  |  |  | 0.05 |
|  |  |  |  |  |  |  |  |
| **Not Useful SLMs** | | | | | | | |
| Yi 6B | replicate.com | 6B | 3% |  |  | Function def + doc string as prompt | 0.44 |
| Code Gemma 7b IT | Google Colab | 7B | 0% |  |  | Unusable output (only completion OR 1 or 2 random lines) |  |
| Gemma 7B | replicate.com | 7B | 0 % |  |  | Incoherent output | 0.05 |
| Gemma 2B | replicate.com | 2B | 0 % |  |  | Incoherent output | 0.05 |
| Flan-T5 | replicate.com |  | 0% |  |  | Incoherent output |  |
| Phi | replicate.com |  | 0% |  |  | Incoherent output |  |
| Mamba 2.8B | replicate.com | 2.8B | n/a |  |  | Incoherent output | 0.02 (20 calls) |
|  |  |  |  |  |  |  |  |

1. **Conclusions**

Even though many more models are hosted on Replicate.com than on Mistral.ai, including Lamma 3 8B, the Mistral family of models showed much better overall results while some models from replicate.com showed a 0% result.

**Codestral Mamba** showed the best result of 75.61% (75% on the Big Code Leaderboard), followed by **Ministral 8B** with 72.56% and **Ministral 3B** with 64.63%.

The best model hosted on Replicate.com is Llama 3 8B with a Pass@1 score only 51.5% which is slightly higher than its Big Code score.

Some models in my experiments scored lower than on the Big Code Leaderboard. For example, the smallest Ministral 3B model has the highest score there (77.4%) among the models that I tried, but it’s only third in my results. This probably means that if I find a better prompt for this model, there is a room for the results to improve.

Next steps:

* Trying the best SLMs from the Big Code Leaderboard, namely [Nxcode-CQ-7B-orpo](https://huggingface.co/NTQAI/Nxcode-CQ-7B-orpo), [CodeQwen1.5-7B-Chat](https://huggingface.co/Qwen/CodeQwen1.5-7B-Chat), [DeepSeek-Coder-7b-instruct](https://huggingface.co/deepseek-ai/deepseek-coder-6.7b-instruct), [OpenCodeInterpreter-DS-6.7B](https://huggingface.co/m-a-p/OpenCodeInterpreter-DS-6.7B), [Artigenz-Coder-DS-6.7B](https://huggingface.co/Artigenz/Artigenz-Coder-DS-6.7B). This can be done only using the transformers library in a Jupyter notebook in Google Collab or other GPU-enabled environment as these models are not hosted anywhere.
* Then, take only the best performing models (>50%) and run them against the MBPP dataset to get another metric for comparison.
* After which I will select a few candidates performing well on both datasets for the SLM fine-tuning effort.