HW2: Code Generation and Prompting

CS598LMZ Spring 2025

1 Goal

In this assignment you will use code models for program synthesis. The goal is to test how code LLMs can solve programming problems.

1.1 HumanEval benchmark

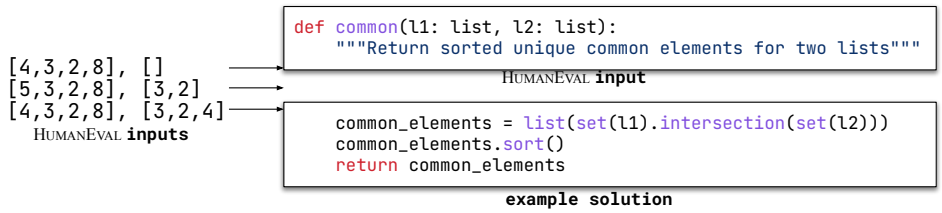

Figure 1: HumanEval example

Program synthesis, to some, is considered as the *holy grail* of computer science. The ability to synthesis/generate arbitrary code according to user specification is highly sought after as it can be extremely helpful for developers in their daily workflow. To evaluate program synthesis abilities, especially for LLM-based approaches, researchers have turned to benchmarks such as HumanEval.

HumanEval contains 164 unique hand-crafted problems where the input is a function header + docstring and the task is to generate a correct code snippet to complete the described task. Figure 1 shows an example task in HumanEval. We observe the problem `common` is to compute the common elements from two lists as input. Figure 1 also shows an example solution which can successfully solve the task. In HumanEval, the user specification is described in terms of the docstrings in natural language and LLMs are tasked to turn this natural language specification into executable code.

Unlike the edit distance metric you computed in the previous homework, program synthesis should require *functional correctness* evaluation. In HumanEval, we evaluate this functional correctness through the use of test cases by executing the LLM generated code snippets on specific test inputs (see example in Figure 1) and checking whether the output matches with the groundtruth solution output. Due to the probabilistic nature of LLM generation, a popular metric of pass@k is used to evaluate program synthesis ability, where it symbolizes the ability for an LLM to solve a problem in k tries. pass@k can be formalized as:

$$\text{pass@}k = \mathbb{E}_{problems}[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}] \tag{1}$$

where n is the number of samples we take from LLM per each problem, and c is the number of correct samples that pass all tests (usually $n >> k$ in order to obtain an accurate estimation). The unbiased variant of pass@k is widely used, to our knowledge, first popularized by the Codex paper[1].

In the homework, to reduce the amount of time required to generate LLM solutions, we will simplify by using the greedy output (i.e., temperature = 0). As such, the pass@k devolves into a simple accuracy metric, also referred to as greedy pass@1. You may understand this as generating only one solution (the greedy one) per problem without needing to sample k solutions per problem.

[1] https://arxiv.org/pdf/2107.03374.pdf

EvalPlus. In this homework, we will make use of the recent EvalPlus work which provides an framework to easily obtain and test HumanEval code generations. Additionally, EvalPlus also provides more robust testcases since the original HumanEval contains many problems without sufficient amount of test cases to cover corner case behaviors. If you are interested in EvalPlus, feel free to check out the repository[2] and paper[3]. As well as stay toon for later lectures where EvalPlus will be covered with a presentation on April 15th (Code Benchmarking).

2 Instructions

To complete this assignment, following the steps described below to complete all tasks.

2.1 Program Synthesis

- First use the Jupyter Notebook file provided in the homework assignment GitHub: https://github.com/uiuc-cs598lmz-s25/hw2/blob/main/code_generation.ipynb and follow the initial setup steps. Ensure you have changed the hardware accelerator to GPU if you are using Google Colab. Note: if you are using Colab for the first time please read through the Colab FAQ and/or follow some initial guides.
- Next, load the model we will be using for this assignment - CodeGen (`codegen-2B-mono`) and the corresponding tokenizer. This is a decoder-only model designed for autoregressive code generation (especially for python). CodeGen has already been covered in the class on Feb 18th (Decoder-only Models), please see the paper for more details about the model. In the notebook, we have provided the function `load_base_model` to do that. Feel free to also play around with the base model to understand the usage and the limitation, especially when given code as input.
- In this homework, we will be using HumanEval, which is described in Section 1.1. Specifically we will use the EvalPlus library to load the dataset. Please take a moment and play around with the raw dataset to understand the different fields in the raw dataset.
- Implement the `program_synthesis` function to perform code generation on the HumanEval dataset using the CodeGen model. In this homework assignment, we aim to auto-complete the function body using the specifications specified in the docstring in each of the problems in HumanEval. Note, your `program_synthesis` should return the complete function (including docstring, function header + body) for each of the problem with temperature = 0 (i.e., greedy sampling).
- Evaluate the solutions generated in the previous step again using the EvalPlus library. We have included the command in the notebook for evaluation. Note: EvalPlus returns both `Base` score and `Base + Extra` score, please log both in the report. Furthermore, please use `check_which_failed` to see a list of problems which the code synthesis process did not generate the correct solution. Please pick at least one of these incorrect solutions and describe what exactly was the problem in the report. Hint: you may check the `canonical_solution` field in the dataset to see the groundtruth solution for each problem as a reference.

2.2 Improve Program Synthesis

- Now, we aim to improve the previous synthesis ability of the model through some prompting/few-shot strategies.
- Implement the new inference function of `program_synthesis_improved` similarly to what you did for the previous code synthesis function. However, in this case please improve of the previous score by modifying the inputs from just being the raw function header + docstring by using different strategies. These strategies may be discussed previously in class, here are some example ways you may want to try, but feel free to implement your own approach:

[2] https://github.com/evalplus/evalplus
[3] https://openreview.net/pdf?id=1qvx610Cu7

- **Few-shot learning:** provide a few small examples of similar problems being solved to the model before giving the actual problem.
- **Test-case demonstration:** obtain some example tests (Hint: you can check the `base_input` field of the dataset to see what are the inputs used to test the solution and expected output by using `canonical_solution` field).
- **Prompt engineering:** you may also experiment with various different hand-written prompts to add to the beginning of the input to elicit better performance from the model (e.g., *"Lets think step-by-step first"*).

Please also use greedy decoding for this generation step.

- Evaluate the new solutions generated using the EvalPlus library and report the new performance. Please also select an example problem that the new code synthesis strategy was able to solve but the previous naive code synthesis approach did not and discuss potential reason why in the report.
- Note: Please do not worry about the exact amount of improvement (there is no defined target score, for the base CodeGen generation, you can expect a performance of around 20% for HumanEval base score). The goal is experiment with the model and get familiar with them so they may be useful for your own project/research later on.

2.3 Report

- Log both `Base` score and `Base + Extra` for both base and improved code synthesis runs.
- Provide one example failed solution of the base program synthesis run and describe why it is incorrect.
- Describe what your strategies are for the improved program synthesis function.
- Provide one example correct solution of the improved program synthesis run that was previously incorrect in the base program synthesis run and describe what changed.

3 Deliverables & Grading

You are expected to upload a zip file. The name of the zip file should be your NetID. The zip file should contain only the following:

- (15pt = 10pt: base synthesis + 5pt: improved synthesis) `code.py` including the complete implementation (Note: if working on Colab you can save the notebook as a python file)
- `codegen_results/` folder containing all the solutions generated for the base run as well as the `eval_results.json` file. Note: if you did all steps correct, this folder, all files inside should already be generated.
- `codegen_results_improved/` folder containing all the solutions generated for the improved run as well as the `eval_results.json` file. Note: if you did all steps correct, this folder, all files inside should already be generated.
- (5pt) `report.md` including two parts in the following order:

  1) Results of both the base run and new improved run on HumanEval computed using EvalPlus.

  `Base Run: {Base score}, {Base + Extra score}`

  `Improved Run: {Base score}, {Base + Extra score}`

  2) Paragraphs to answer the corresponding questions (see Section 2.3)

Warning: Not following the above format will result in deduction in marks.

4 Resources

- Note this project can be completed on Google Colab, see this for more detail: https://research.google.com/colaboratory/faq.html However you are more than welcome to use any GPUs you have access to
- Please read CodeGen paper for more detail: https://arxiv.org/pdf/2203.13474.pdf
- Pleas read EvalPlus paper for additional detail about the benchmark: https://openreview.net/pdf?id=1qvx610Cu7