Fine-tuning LLM for Multilingual Code Generation

Link to Google Doc to add comment and see latest version: click here

OVERVIEW

- **Very** Large code LLMs (> 1B params) often have good performance in a variety of languages (**Salesforce/codegen-2B-multi -> C, C++, Go, Java, JavaScript, and Python**)
- **Small** code LLMs (< 1B params) are usually trained to work on one programming language to have good performance (**Salesforce/codegen-350M-mono -> Python**)
- Small code LLMs can be extended to cover more languages by a variety of methods:
    a. Full Fine-tuning:
        - Pros: Best performance in fine-tuned model
        - Cons: High computational requirements, catastrophic forgetting
    b. LoRa Fine-tuning:
        - Pros: Efficacy, Scalable, No catastrophic forgetting
        - Cons: Lower performance than full fine-tuning
    c. Knowledge Distillation: tbd

GOALS

1. Optimally extend Small code LLMs to new programing language considering:
    a. Performance: in terms of Perplexity and HumanEval
    b. Efficiency: in terms of Memory and Time
2. Study the effect of different factors on the training, evaluation and inference of models:
    a. Training Data: Selection of github repos and how to filter them
    b. Training Strategies: learning rate, batch size, gradient accumulation
    c. Evaluation: Stride and stop tokens
    d. Inference: Sampling strategy and temperature
3. Qualitative Analysis of the results:
    a. Failure modes comparison in HumanEval between Python and Java (Exceptions, runtime errors and not passing test cases)
    b. Correlation between Perplexity and HumanEval

4. Further objectives:
    a. Relation between Different Programming Languages
    b. Knowledge Distillation

Timeline:

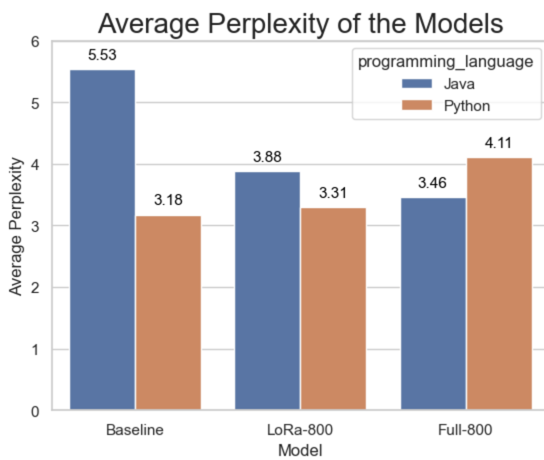
1. Read papers on Code LLM and select appropriate baseline -> **Salesforce/codegen-350M-mono**
2. Read papers on Code datasets and select perplexity evaluation data -> **bigcode/the-stack-dedup**
3. Read papers on evaluation strategy to implement HumanEavl in python and Java -> follow standards of **CodeX** and **Multipl-E**
4. Setting up Evaluation:
    - Done: Perplexity Evaluation on Google-Colab and HumanEval on Google-Colan and Virtual Containers (for Java)
    - Challenges: Perplexity Dataset, Humaneval Stop tokens for java
5. Generate Baseline Results using baseline model
6. Setting up Training and Evaluation environments:
    - Done: Configure Full & LoRa fine-tuning on Google-Colab, Track training progress on WandB, Save checkpoints on HuggingFace.
    - Challenges: Resuming Training reset some variables(LR, number of steps), Memory to 15GB and duration to 3 hours
7. Generate Results for the full tuned model and LoRa tuned model.
8. Edit the training setup (learning rate , gradient accumulation and batch size) based on the literature (**bigcode/starcoder**) and the results of the first finetuning
9. Train the model again and notice the effect of modification on two models:
    - Full fine tuning: *Significant improvement* in Perplexity and HumanEval for **Java** and *significant drop* for **Python**
    - LoRa fine tuning: *Significant improvement* in Perplexity and *Slight* in HumanEval for **Java** and *no change* for **Python**
10. Plan Next steps: further modifications, larger models, inference and distillation
11. Setting up live inference environment:
    - Done: Live inference configured on HuggingFace spaces to generate code from tuned models base on given prompt -> Code Inference
    - Challenges: CPU limitation

Results

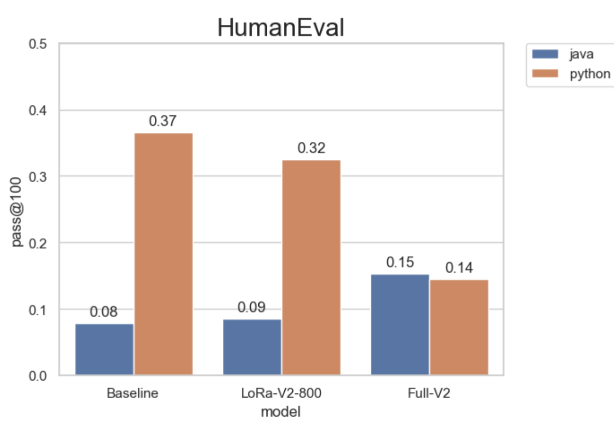
1. Comparisons

- Comparing the models in terms of:

    i. Perplexity: The Lower the better



    ii. HumanEval: The Higher the better



    iii. Average Memory Usage: given the same setup for all hyperparameters

    iv. Training Time