StarCoder: May the Source be With You!

Ramaseshan RAmachandran

March 12, 2025

- StarCoder[1], a large language model, is designed for code generation.
- It aims to boost developer productivity with powerful code completion and generation.
- The model is trained on a vast dataset of source code from various languages.
- It emphasizes open access and transparency, fostering community collaboration.

- **Multilingual Support:** Trained on a wide range of programming languages.
- **Contextual Understanding:** Generates accurate and relevant code by leveraging context (code snippets).
- **Fill-in-the-Middle (FiM) Capabilities:** Allows code completion within existing code blocks.
- **Open Access:** Weights and code are released for public use and research.
- **Large Context Window:** Ability to process larger code snippets for better understanding.
  - Utilizes a context window of 16,384 tokens
  - Employs a sliding window attention of 4,096 tokens
  - Allows the model to weigh the importance of each token in the input sequence.

- The Stack dataset contains 1 trillion tokens
- 80+ programming languages
- Data preprocessing and filtering done - all symbols $=, <, >$, etc. are considered as a token .
- Emphasizes responsible data usage and transparency.

Fine-Tuning - fine-tuned on 35B Python tokens

- Programming Language Selection Criteria: Languages with over 500 MB of data, top 50 ranking on GitHub 2.0 or TIOBE Index, and dialects of selected languages.
- Data exclusion criteria include configuration languages, unsupported languages, and data formats with limited volume.
- Visual inspection of randomly selected files ensures high-quality code and filters applications based on file extensions.
- Data filtering process involves XML, alpha, HTML, JSON, and YAML filters to remove irrelevant files.
- Implemented to exclude XML files, especially those with extensions like .sld, based on the presence of “<?xml version=“ within the first 100 characters.
- Alpha Filter identifies and removes files with a low percentage of alphabetic characters, especially in data files like MATLAB.

DATA SELECTION - SECTION 3 - II

- Converted Jupyter notebooks to scripts using Jupytext, identifying programming languages from metadata and guessing for unknown notebooks.
- Filtered and merged Markdown and code blocks in Jupyter notebooks to create a dataset of consecutive code-text pairs.
- Data source: 1.045 million structured Jupyter notebooks and natural language conversations from GitHub issues and pull requests.
- Data preprocessing removed auto-generated text, short messages, long comments, and bot-generated content.
- Conversations with two or more users or single-user conversations with less than 7,000 characters are included.
- Data source: Stack dataset, Git commits, and GitHub issues.
- Filtered out non-English issues, repositories with opted-out users, and deduplicated source code files.
- Data anonymization replaced usernames in conversations with participant counters - ip address, password, keys, id, email, user name, etc.

- Core Architecture: Decoder-Only Transformer - focus on predicting the next token in a sequence
- Decoder-only-models: Process the input sequence and generate the output sequence in a single pass
- Performance: The fine-tuning enhances capabilities, particularly in Python-specific tasks, improving overall code generation accuracy.

Hyperparameter	SantaCoder	
Hidden size:	2048	6144
Intermediate size	8192	24576
Max. position embeddings	2048	8192
Num. of attention heads	16	48
Num. of hidden layers	24	40
Attention	Multi-query	Multi-query
Num. of parameters	≈1.1B	≈5.5B

- Multi-query attention, an optimization technique, improves the efficiency of the attention mechanism in SantaCoder.
- This technique reduces memory footprint and speeds inference, enabling longer sequence handling.
- Fill-in-the-Middle (FiM) is crucial to SantaCoder's architecture. It lets the model generate code sequentially and fill missing code segments in existing blocks. This is valuable for code editing and refactoring.
- Optimized for Code Tasks: The architecture is optimized for code-related tasks, considering source code's structured nature and syntax importance.

- Trained on a GPU cluster with 512 A100 80 GB GPUs distributed across 64 nodes.
- Model parallelism used a 3D-parallel layout with tensor and pipeline parallelism, needing 16 GPUs per replica.
- Utilized 32-fold data parallelism with a micro-batch size of 1, accumulating for 16 steps to achieve a global batch size of 512.

- Code completion and generation tools.
- Assisting in code review and debugging.
- Automating repetitive coding tasks.
- Educational tool for learning programming.
- Research platform for further advancements in code LLMs.

CONCLUSION

- StarCoder represents a significant step towards powerful and accessible code generation models.
- Its open nature promotes collaboration and innovation in the field.
- ”May the source be with you!” emphasizes the importance of code accessibility and community.

REFERENCES I

[1] Raymond Li et al. *StarCoder: may the source be with you!* 2023. arXiv: 2305.06161 [cs.CL]. URL: https://arxiv.org/abs/2305.06161.