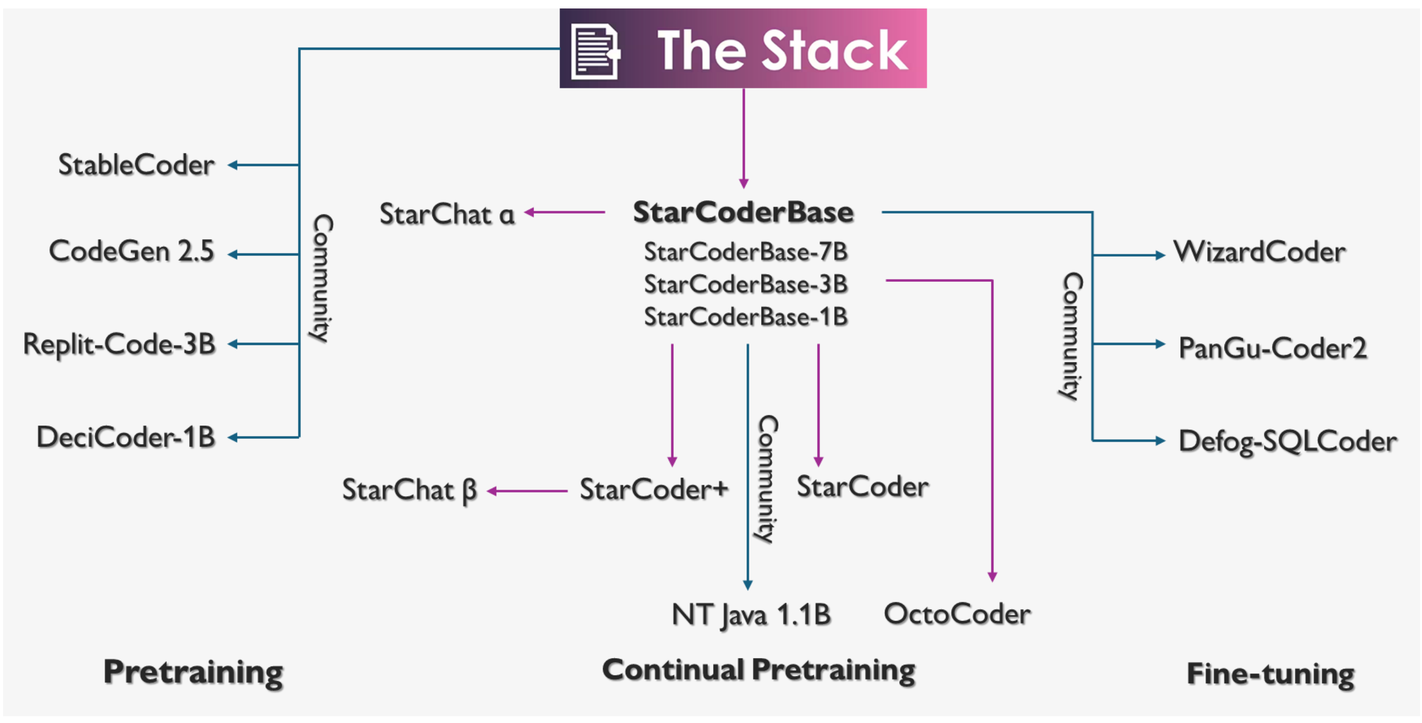
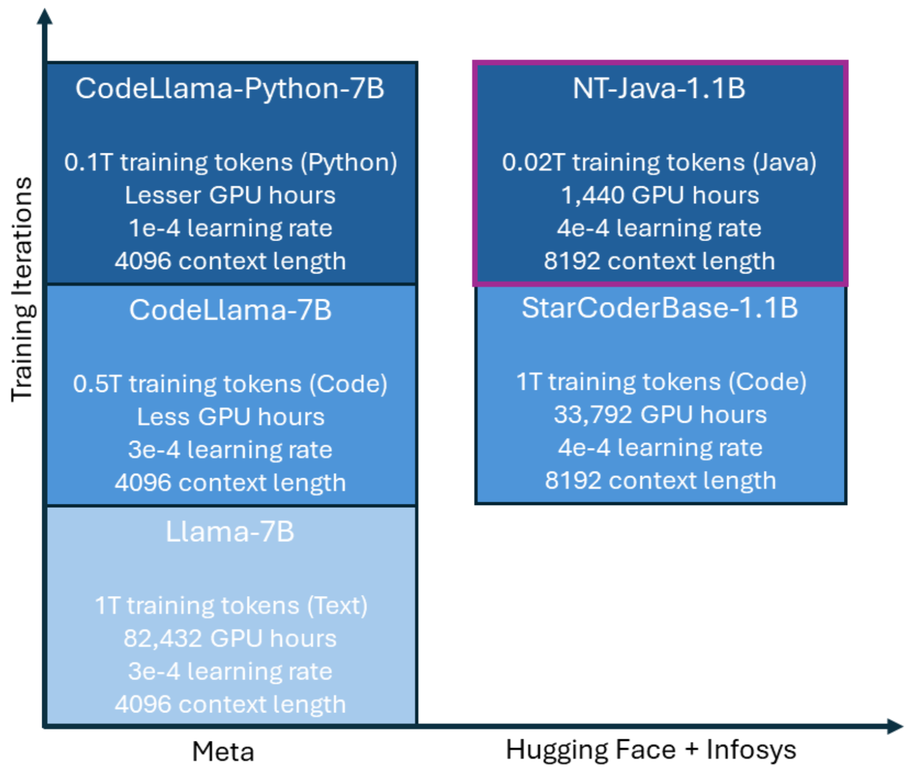
Narrow Transformer: Mono-lingual Code SLM for Desktop

Kamalkumar Rathinasamy, Balaji A J, Ankush Kumar, Gagan Gayari, Harshini K, Rajab Ali Mondal,
Sreenivasa Raghavan K S, Swayam Singh, Mohammed Rafee Tarafdar

NEURAL INFORMATION PROCESSING SYSTEMS





Motivation

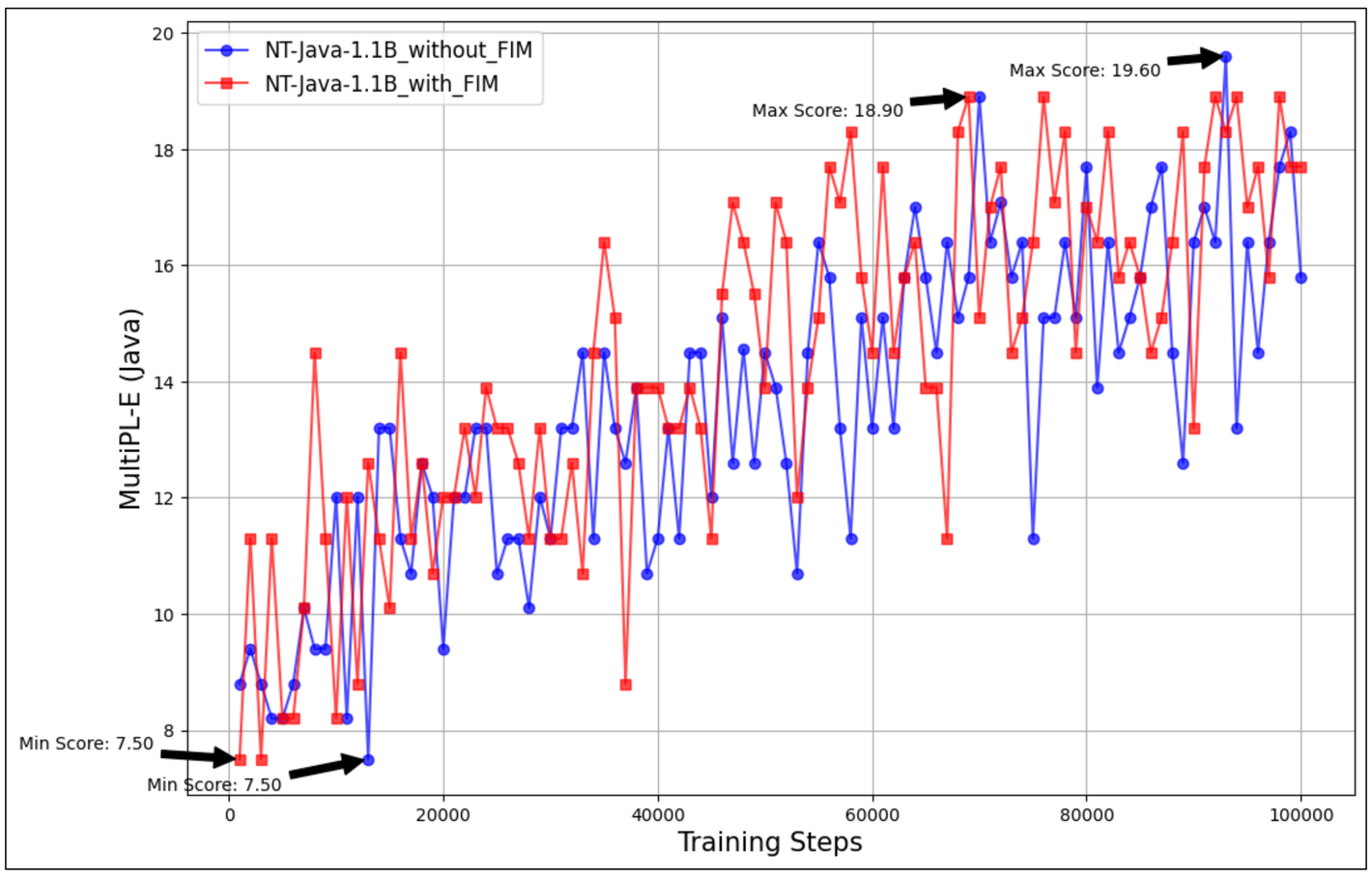
Generic multi-lingual code models boost productivity but the one-size-fits-all approach of these generic multi-lingual code models often falls short in meeting the nuanced requirements of project-level coding tasks in an enterprise. This led to the development of Narrow Transformers (NTs), specialized models optimized for specific programming languages. These NTs are designed to optimize performance for a specific programming language, balancing the trade-offs between model size, inferencing cost, and operational throughput. As demand for tailored solutions grows, we can expect a surge in NT development, providing the precision and efficiency required by enterprise projects.

Ablation on In-filling

Experiment 1: Next token prediction objective: Trained for 100,000 steps (5 epochs) with a batch size of 1 million tokens. Learning rate started at 4×10−4, decayed to 4×10−6 with 1,000 iterations of linear warmup. Training spanned 12 days with a global batch size of 180. Checkpoints saved every 1,000 steps.

Experiment 2: Fill-in-the-Middle (FIM): Repeated Experiment 1 with a 50% FIM rate. The FIM dataset was split into SPM (Suffix-Prefix-Middle) and PSM (Prefix-Suffix-Middle).

Observations: Without FIM, the model's infilling capability dropped significantly (FIM scores near zero). With FIM, there was a minor decrease in MultiPL-E metrics (about 0.7%), but the model maintained its infilling proficiency. Performance comparisons are shown in chart.



Model	FIM	HumanEval-FIM (Java)	MultiPL-E (Java)
NT-Java-1.1B (Experiment 1)	No	0.01	19.6
NT-Java-1.1B (Experiment 2)	Yes	0.67	18.9

Computation Capabilities

Our analysis indicates that our NT-Java quantized models achieve an optimal balance between accuracy and resource utilization, making them a suitable candidate for deployment in resource-constrained environments. For the computation of the MultiPL-E scores of the quantized variants, we employed the ‘load in 4-bit’ and ‘load in 8-bit’ parameters within the BigCode Eval Harness.

Model	Pass@1 (Java)	Size (GB)
StarCoderBase-1.1B	14.2	≈ 2.27
NT-Java-1.1B_Q4	**15.1**	**0.76**
NT-Java-1.1B_Q8	**17.7**	**1.23**
StarCoderBase-3B	19.25	≈ 6.1
NT-Java-1.1B	**20.2**	**2.27**

References

- Raymond Li et al. “StarCoder: may the source be with you!” In: CoRR abs/2305.06161 (2023).
- Loubna Ben Allal et al. “SantaCoder: don’t reach for the stars!” In: CoRR abs/2301.03988 (2023).
- Baptiste Rozi ` ere et al. “Code Llama: Open Foundation Models for Code”. In: CoRR abs/2308.12950 (2023).
- https://github.com/Infosys/Megatron-LM#nt-java-11b-extending-pretraining

Hugging Face

Paper