Although these works show a clear preference on large models, the potential of training smaller models with larger dataset remains under-explored. Instead of training compute-optimal language models, Touvron et al. (2023a) highlight the importance of the inference budget, instead of focusing solely on training compute-optimal language models. Inference-optimal language models aim for optimal performance within specific inference constraints This is achieved by training models with more tokens than what is recommended by the scaling law (Hoffmann et al., 2022). Touvron et al. (2023a) demonstrates that smaller models, when trained with more data, can match or even outperform their larger counterparts. Also, Thaddée (2023) suggest that existing scaling laws (Hoffmann et al., 2022) may not predict accurately in situations where smaller models are trained for longer periods.

Motivated by these new findings, this work focuses on exploring the behavior of smaller models when trained with a significantly larger number of tokens than what is suggested by the scaling law (Hoffmann et al., 2022). Specifically, we train a Transformer decoder-only model (Vaswani et al., 2017) with 1.1B parameters using approximately 3 trillion tokens. To our knowledge, this is the first attempt to train a model with 1B parameters using such a large amount of data. Following the same architecture and tokenizer as Llama 2 (Touvron et al., 2023b), we name our model TinyLlama. TinyLlama shows competitive performance compared to existing open-source language models of similar sizes. Specifically, TinyLlama surpasses both OPT-1.3B (Zhang et al., 2022) and Pythia-1.4B (Biderman et al., 2023) in various downstream tasks.

Our TinyLlama is open-source, aimed at improving accessibility for researchers in language model research. We believe its excellent performance and compact size make it an attractive platform for researchers and practitioners in language model research.

2 Pretraining

This section describes how we pre-trained TinyLlama. First, we introduce the details of the pre-training corpus and the data sampling method. Next, we elaborate on the model architecture and the hyperparameters used during pretraining.

2.1 Pre-training data

Our main objective is to make the pre-training process effective and reproducible. We adopt a mixture of natural language data and code data to pre-train TinyLlama, sourcing natural language data from SlimPajama (Soboleva et al., 2023) and code data from Starcoderdata (Li et al., 2023). We adopt Llama's tokenizer (Touvron et al., 2023a) to process the data.

SlimPajama This is a large open-source corpus created for training language models based on RedPajama (Together Computer, 2023). The original RedPajama corpus is an open-source research effort aimed at reproducing Llama's pretraining data (Touvron et al., 2023a) containing over 1.2 trillion tokens. The SlimPajama was derived by cleaning and deduplicating the original RedPajama.

Starcoderdata This dataset was collected to train StarCoder (Li et al., 2023), a powerful open-source large code language model. It comprises approximately 250 billion tokens across 86 programming languages. In addition to code, it also includes GitHub issues and text-code pairs that involve natural languages. To avoid data duplication, we remove the GitHub subset of the SlimPajama and only sample code data from the Starcoderdata.

After combining these two corpora, we have approximately 950 billion tokens for pre-training in total. TinyLlama is trained on these tokens for approximately three epochs, as observed by Muennighoff et al. (2023), where training on data repeated for up to four epochs results in minimal performance degradation compared to using unique data. During training, we sample the natural language data to achieve a ratio of around 7:3 between natural language data and code data.

2.2 Architecture

We adopt a similar model architecture to Llama 2 (Touvron et al., 2023b). We use a Transformer architecture based on Vaswani et al. (2017) with the following details: