Training a Subbillion LLM based on The Smol Training Playbook

Evaggelos Asitzoglou
School of Electrical and Computer Engineering
National Technical University of Athens
Athens, Greece
vaggelisima@yahoo.com

Panagiotis Avgerinos
School of Electrical and Computer Engineering
National Technical University of Athens
Athens, Greece
panikavger@gmail.com

Michail-Athanasios Peppas
School of Electrical and Computer Engineering
National Technical University of Athens
Athens, Greece
peppas.michael@gmail.com

Andreas Saounatsos
School of Electrical and Computer Engineering
National Technical University of Athens
Athens, Greece
asaounat@gmail.com

Giannis Chatzigeorgiadis
School of Electrical and Computer Engineering
National Technical University of Athens
Athens, Greece
yiannishatzigeorgiadis@gmail.com

***Abstract*—In this report, we detail the end-to-end development of our own 0.5B-parameter LLM, which we pretrained from scratch on 22 billion tokens. Although efficient small-scale models have become increasingly accessible, optimizing their training curriculum remains a challenge. Building upon the foundational methodologies of the Hugging Face SmolLM playbook, we introduce a custom three-stage pretraining pipeline. Specifically, we diverge from the baseline by modifying the depth-to-width ratio to favor a deeper, narrower architecture. Additionally, we allocate a larger fraction of our parameter budget to the embedding layer, drawing inspiration from recent advancements in small-scale model design. Following the pretraining phase, the model underwent Supervised Fine-Tuning (SFT) to align its output for instruction-following tasks. Our training loss curve and downstream benchmarks demonstrate that our custom pretraining curriculum and subsequent SFT pipeline produce strong performance on logical reasoning benchmarks, outperforming many older models trained on larger data volumes. Our training configurations, and final model weights are publicly available at https://github.com/ntua-el21026/NTUA-LLama_500M_Nanotron.git.**

I. Introduction

In recent years, large language models (LLMs) have demonstrated remarkable capabilities across a wide range of natural language processing and, more generally, pattern recognition tasks, enabled by their scalability in both parameters and training data. Modern models may contain hundreds of billions of parameters, allowing them to effectively capture highly complex statistical patterns. However, this scale comes at a cost: training such models requires tremendous computational resources and specialized hardware, resulting in significantly increased financial investment.

These limitations have motivated the adoption of smaller, more efficient language models designed for targeted use rather than universal generality, and trained on curated datasets to solve specific tasks not only more efficiently but also with lower infrastructure demands. Because they require fewer parameters and reduced compute, such models are faster to train, cheaper to deploy, and accessible to organizations without access to excessive resources. Furthermore, their compact memory footprint unlocks the potential for on-device, offline inference. This edge-deployment capability offers distinct advantages over massive cloud-based architectures, including guaranteed data privacy, zero network latency, and reliable functionality in environments with limited internet connectivity.

The Smol Training Playbook [1] emerges within this context as a practical methodology for building high-quality language models under realistic resource constraints. Rather than treating model development as a purely scaling-driven process, it frames training as a sequence of deliberate engineering decisions, including architectural selection, data curation, controlled ablations, evaluation design, supervised fine-tuning, and infrastructure optimization.

Inspired by this methodology, we followed a similar engineering process to build a 500M-parameter LLaMA-style model using the Nanotron [2] framework. The model was pretrained on a curated mixture of high-quality datasets, totaling approximately 22 billion tokens, enabling efficient experimentation under constrained compute while maintaining large-scale training characteristics. Crucially, we recognize that at the sub-billion parameter scale, attempting to achieve universal competence often results in diluted, uniformly mediocre performance. Therefore, rather than spreading our limited parameter budget thinly to maximize broad factual recall, we deliberately adopted a deep and narrow architectural topology to optimize for targeted proficiency in logical reasoning

II. Approach

Our core methodology is grounded in the standard decoder-only Transformer architecture [3], incorporating key structural optimizations and the tokenizer vocabulary introduced by the LLaMA 3 model family [4]. Because our training pipeline was heavily constrained by the available compute budget—specifically restricted to a single compute node on the Leonardo supercomputer with a strict wall-clock time limit—our approach necessitated highly deliberate engineering. Consequently, every architectural scaling decision and infrastructural configuration was rigorously optimized to maximize training throughput and resource efficiency.

A. Architecture

TABLE I: Architectural Comparison: GPT-2 (Medium) vs. Our Model

Hyperparameter	GPT-2 (Medium) [5]	Our Model
Estimated Parameters	∼345M	∼492M
Hidden Layers (Depth)	24	32
Hidden Dimension (Width)	1024	1024
Attention Architecture	MHA (16 Q, 16 KV)	GQA (16 Q, 4 KV)
Vocabulary Size	50,257	128,256
Context Window	1024	1024
Positional Embeddings	Learned Absolute	RoPE
Activation Function	GELU	SwiGLU (SiLU)
Normalization	LayerNorm	RMSNorm

The primary objective of this phase was to design a language model with approximately 500 million parameters capable of outperforming legacy architectures of similar scale, such as GPT-2 [5]. A detailed comparison of our architectural hyperparameters against the GPT-2 Medium baseline is provided in Table I. Drawing upon recent empirical studies on sub-billion parameter models, specifically MobileLLM [6], we adopted a "deep and narrow" topological strategy. We operated under the premise that while expanding the hidden dimension (width) primarily increases a model's capacity for broad factual memorization, increasing the number of transformer layers (depth) significantly enhances its ability to generalize and execute complex sequential logic. By sacrificing some encyclopedic knowledge capacity in favor of architectural depth, we engineered a model highly optimized for reasoning and mathematical problem-solving.

The exact parameter count of our model is approximately 492M. To maximize the efficiency of this strict parameter budget, we integrated several modern architectural mechanisms from the LLaMA family [4]:

- **Deep and Narrow Topology:** The network is configured with 32 hidden layers and a relatively constrained hidden dimension of 1024. This high depth-to-width ratio forces the model to learn more abstracted, generalizable representations early in the training process, a crucial quality for small-scale models competing against larger counterparts.
- **Expanded Vocabulary and Tied Embeddings:** We adopted the LLaMA-3 tokenizer, which features a massive vocabulary size of 128,256 tokens. While this drastically improves the tokenization efficiency for mathematical equations, code, and multilingual text, it introduces a significant "embedding tax" (consuming roughly 131M parameters). To mitigate this, we utilized weight tying between the input embedding and the final output unembedding layers, a technique proven to regularize the model and save parameters without degrading performance [7].
- **Grouped-Query Attention (GQA):** To optimize both training memory footprint and future inference speed, we implemented GQA [8]. The model utilizes 16 attention heads for queries and 4 heads for key-value pairs, substantially reducing the memory bandwidth required for the KV cache while maintaining performance comparable to standard Multi-Head Attention.
- **Modern Activation and Normalization:** We replaced standard Feed-Forward Networks with SwiGLU activations (using an intermediate size of 2816 and the SiLU function) to improve gradient flow and model capacity [9]. Furthermore, we employed Root Mean Square Normalization (RMSNorm) [10] to enhance training stability while reducing computational overhead compared to LayerNorm.
- **Rotary Positional Embeddings (RoPE):** To allow for better length extrapolation and robust relative context utilization, we applied RoPE [11] with a base theta of 10000.0, supporting a maximum context window of 1024 tokens.

B. Infrastructure and Training Setup

All pretraining and fine-tuning stages were executed on a single compute node within the Leonardo supercomputer, equipped with four NVIDIA A100 GPUs (60GB VRAM capacity each) and 32 CPU cores. To orchestrate the distributed training, we utilized the Nanotron framework [2], which provides optimized primitives for 3D parallelism.

Due to our deep 32-layer architecture and the significant memory overhead of the expanded vocabulary, memory management required rigorous optimization. Initial empirical tests using a micro-batch size of 32 resulted in immediate Out-Of-Memory (OOM) errors. Furthermore, due to framework-specific compatibility constraints between ZeRO redundancy optimizers [13] and our requirement for FP32 gradient accumulation, we opted to disable ZeRO entirely (ZeRO Stage 0). Instead, we relied on pure Data Parallelism (DP=4) combined with activation checkpointing [14] to fit the model within the 60GB VRAM limit.

To maintain a mathematically optimal global batch size without exceeding memory constraints, we reduced the micro-batch size to 4 sequences per GPU and applied 16 gradient accumulation steps. Across the 4 replica workers, this yielded an effective global batch size of 256 sequences. Data loading was parallelized using 8 dedicated CPU workers to prevent I/O bottlenecks, allowing us to maintain a stable hardware throughput of approximately 100,000 tokens per second across all training stages.

The network was optimized using a fused AdamW optimizer [15] with gradients accumulated in FP32 precision. Because our deep "narrow and long" topological design significantly increased the risk of exploding gradients—a common instability when training Transformers with high layer counts—gradient clipping was a major architectural concern. To mitigate this, gradients were strictly clipped at a maximum norm of 1.0 throughout all phases. While specific hyperparameters such as peak learning rate and warmup steps were dynamically adjusted per training stage (detailed in subsequent sections), our general strategy relied on a cosine decay schedule. Crucially, to support our continuous multi-stage curriculum, the learning rate was never allowed to decay to zero; a minimum threshold was always enforced to preserve model plasticity for learning new data distributions.

III. Phase 1: Pre-training

A. Pre-training Data

Selecting appropriate data sources is crucial for a small-scale, general-purpose LLM. We train on English-only text using a three-part corpus spanning web/language data, math reasoning data, and code data. In this setting, the dataset mixture often affects downstream performance more than minor architecture or optimizer tweaks: with limited capacity, noisy sources can waste the token budget and degrade representations, instruction following, and reasoning. Because corpus composition directly shapes the model's skills (e.g., exposure to step-by-step explanations, code patterns, and Q&A interactions), we emphasize high-signal text and a stage-wise curriculum that builds language competence first and then gradually introduces math and code while mitigating catastrophic forgetting.

Web / language data. For the language backbone, we used a mixture of *Cosmopedia*, *SlimPajama*, *FineWeb-Edu*, and *Stack Exchange*. Aggregated across stages, the overall web-mixture proportions were: 52% *Cosmopedia*, 25% *SlimPajama*, 17% *FineWeb-Edu*, and 6% *Stack Exchange* (note that each stage uses different internal ratios; these percentages refer to the total web mixture over the full run).

Cosmopedia is a large synthetic corpus (e.g., textbooks, blog posts, stories, WikiHow-style articles) generated by Mixtral-8x7B-Instruct-v0.1, reported to contain >30M files and ~25B tokens [12], [16]. *SlimPajama* is a cleaned and deduplicated multi-source web dataset derived from RedPajama, used to inject broad and diverse web coverage [17]. *FineWeb-Edu* is a high-quality educational subset of FineWeb, accompanied by quality fields such as `score` (0–5) and `int_score`; in our pipeline we retained only the top-quality portion with `int_score` $\in \{4, 5\}$ to emphasize explanatory writing [18], [19]. Finally, *Stack Exchange* provides large-scale Q&A-style discourse from multiple communities and is distributed through periodic public data dumps, which we use to strengthen instruction-like interactions [20], [21].

Math data. For mathematical reasoning, we used *OpenMathReasoning* exclusively. This dataset is designed for math reasoning training and contains hundreds of thousands of math problems (e.g., ~306K unique problems as reported on its dataset card) [22], [23]. Given the relatively small fraction of math tokens in our overall budget, we expect the model to handle *basic* math reliably, but not consistently solve complex problems.

Code data. For code, we used the *StarCoder* training dataset (*bigcode/starcoderdata*) and restricted programming content to *Python* only. The StarCoder dataset contains permissively licensed code across many languages and is described at large scale in the dataset card and accompanying paper [25], [26]. As with math, the code subset we used is relatively small, but proved sufficient for simple programming tasks.

TABLE II: Aggregated web/language mixture (over all stages).

Dataset	Share
Cosmopedia	52%
SlimPajama	25%
FineWeb-Edu	17%
Stack Exchange	6%

B. Training stages

Overview. We adopt a three-stage pre-training curriculum that progressively shifts the data distribution from broad web text to a more instruction- and reasoning-oriented mixture. Across the full pre-training run, the aggregated token split between web/math/code is 84/8/8, while each stage uses a distinct modality ratio (Table III). This design aims to (i) build a strong linguistic foundation, (ii) introduce math and code in a controlled manner, and (iii) prepare the model for SFT by emphasizing instruction-like interactions.

Table IV summarizes the modality ratios and learning-rate schedules used in each stage.

Stage 1: Web-only foundation (language competence). Stage 1 uses only web/language datasets (no math/code), accounting for more than half of the total training tokens. The objective is to learn robust token-level statistics, grammar, and general world knowledge from high-signal explanatory text, while maintaining breadth via diverse web sources. In practice, Stage 1 stabilizes optimization and reduces the risk of the model overfitting early to narrow domains (e.g., code) before it has acquired general language competence. We train Stage 1 with a peak learning rate of $\eta_{\max} = 3 \times 10^{-4}$, warmup of 100 steps, and a minimum learning rate $\eta_{\min} = 5 \times 10^{-5}$ under a decayed schedule.

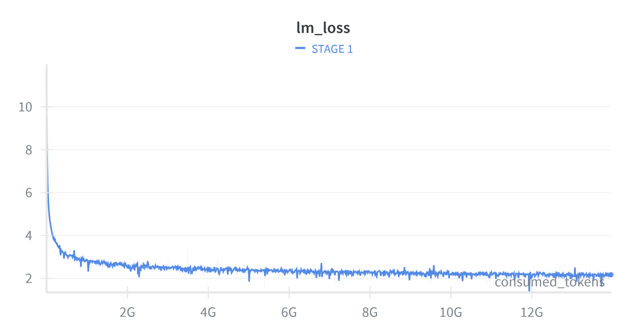


Fig. 1: LM training loss (`lm_loss`) during Stage 1 (web-only foundation).

Stage 2: Controlled introduction of reasoning modalities (math & code). Stage 2 introduces both math and code at equal weight (25% math, 25% code), while retaining 50% web text. This stage targets reasoning and syntax-heavy patterns without sacrificing generalization. Keeping half of the tokens as web text acts as an "anchor" distribution, preventing the model from drifting too far toward domain-specific features and helping maintain fluency as new modalities are learned. Stage 2 uses $\eta_{\max} = 1.5 \times 10^{-4}$ with 900 warmup steps and $\eta_{\min} = 1.5 \times 10^{-5}$.

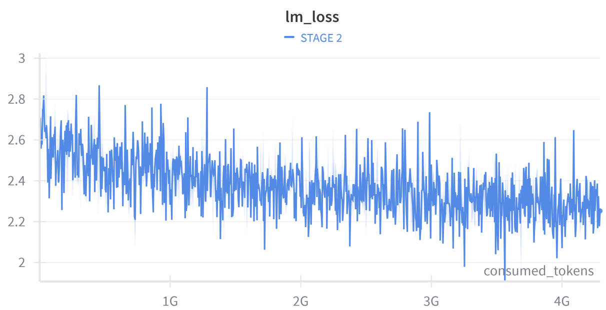


Fig. 2: LM training loss (`lm_loss`) during Stage 2 (mixed modality training).

Stage 3: SFT-oriented warmup (instruction-like discourse). Stage 3 shifts the mixture toward SFT readiness by increasing exposure to *Stack Exchange* within the web portion and using a lighter but non-trivial amount of math and code (16% each, 68% web). The goal is to nudge the model toward question-answer formats, concise explanations, and helpful conversational structure, while still preserving broad linguistic coverage. This stage serves as a bridge between pure pre-training and instruction tuning. Stage 3 uses a lower peak learning rate $\eta_{\max} = 5 \times 10^{-5}$ with 1000 warmup steps and $\eta_{\min} = 5 \times 10^{-6}$.

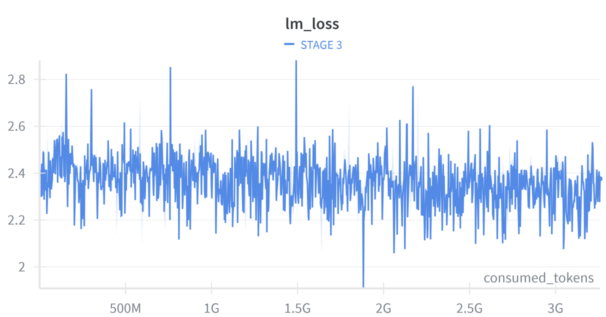


Fig. 3: LM training loss (`lm_loss`) during Stage 3 (SFT-oriented warmup).

TABLE III: Stage-wise modality composition.

Stage	Web	Math	Code
Stage 1 (foundation)	100%	0%	0%
Stage 2 (mixed)	50%	25%	25%
Stage 3 (SFT warmup)	68%	16%	16%

TABLE IV: Learning-rate schedule per stage.

Stage	$\eta_{\max}$	Warmup steps	$\eta_{\min}$
Stage 1	3.0×10^{-4}	100	5.0×10^{-5}
Stage 2	1.5×10^{-4}	900	1.5×10^{-5}
Stage 3	5.0×10^{-5}	1000	5.0×10^{-6}

TABLE V: Qualitative objective of each stage.

Stage	Primary objective
Stage 1	Build broad language competence and stabilize optimization on web text.
Stage 2	Introduce math/code patterns while preserving fluency via a web “anchor” distribution.
Stage 3	Increase instruction-like discourse exposure to bridge pre-training and instruction tuning.

Optimization stability (gradient norm). Across stages, we observed the gradient norm gradually decreasing and eventually stabilizing around a steady value. Aside from a small number of transient spikes, training remained stable; these spikes were effectively suppressed by gradient clipping, preventing destabilizing parameter updates.

Mitigating forgetting across stages. To reduce catastrophic forgetting, we continue sampling from earlier sources as the curriculum progresses. Specifically, Stage 2 includes 16% of the Stage 1 mixture, and Stage 3 retains 0.8% from Stage 1 datasets. This lightweight replay preserves earlier linguistic breadth while leaving sufficient capacity for learning the newer stage distributions.

Dataset emphasis and rationale. Within the web portion, we prioritize *Cosmopedia* for clarity and an explanatory style [12], [16], while *SlimPajama* provides broad and more natural web coverage [17]. *FineWeb-Edu* is filtered to high-quality slices (highest `int_score` levels) and is emphasized in later stages to strengthen educational, step-by-step exposition [18], [19]. Finally, *Stack Exchange* is increased toward the end of training to better align the model with instruction-like Q&A interactions prior to SFT [20], [21]. The aggregated web mixture over all stages is summarized in Table II.

C. Evaluation

Architecture and objective. We evaluate pre-training with a three-tier protocol. Tier 1 tracks training health (loss stability, gradient behavior, throughput, NaN/Inf checks). Tier 2 measures token-level modeling quality on targeted web/code/math slices, based on perplexity (PPL). Tier 3 measures accuracy in downstream reasoning and QA behavior with conditional-likelihood MCQ scoring. This design follows the SmolLM-style development and keeps evaluation reproducible with offline cached inputs [1], [51], [52].

Dataset decision. Web slices (C4, FineWeb-Edu, Wikipedia, WikiText) test broad linguistic robustness and explanatory style. Code slices (CodeParrot, Stack-v2-Edu, StarCoder2-extras, OpenCodeReasoning) test syntax consistency and code-solution structure. Math slices (FineMath 3+/4+, InfiWebMath 3+/4+, OpenMathReasoning, OpenMathInstruct-1) test symbolic and multi-step mathematical text. We expect PPL of all slices to drop, with the model excelling at code and math. Tier-3 tasks cover commonsense, science QA, physical reasoning, and broad knowledge [18], [22], [31]–[50].

The results. Modeling quality (PPL) improves consistently with tokens, and the strongest gains are concentrated in the intended specialization domains (code and math). Reasoning/QA macro behavior is flatter and non-monotonic, indicating that token-level modeling gains do not uniformly transfer to all QA categories. The combined plots make this trade-off explicit: by the end of pre-training, code/math remain comparatively strong, while broad web-style modeling and broad-knowledge/harder science QA remain the main bottlenecks.

The aggregate trend confirms strong specialization in token-level modeling quality across stages. The reasoning/QA aggregate remains stable from Stage 1 to Stage 2, then decreases at Stage 3 end. This indicates that distribution-quality gains are robust, while downstream reasoning gains are bounded and task-sensitive.

TABLE VI: Tier-2 slice values at final checkpoint of each stage. "–" means not in that stage suite.

Slice	Stage 1 final	Stage 2 final	Stage 3 final
general_web	32.07	34.19	34.30
web_fineweb_edu_sample10bt	26.49	22.73	22.46
wiki_en_20231101	20.24	24.10	24.25
books_newslike	46.00	42.19	42.46
code	12.84	6.76	6.40
code_stackv2_edu_filtered	–	6.05	5.45
code_starcoder2data_extras_lhq	–	10.61	10.34
code_opencodereasoning	–	–	8.72
math_finemath3plus	12.81	8.60	8.44
math_finemath4plus	–	9.38	9.17
math_infiwebmath3plus	18.52	11.42	11.20
math_infiwebmath4plus	–	9.53	9.38
math_openmathreasoning	–	–	6.44
math_openmathinstruct1	–	–	7.01

Slice-level results match the intended objective of prioritizing reasoning/code/math capability. Code and math slices show the strongest improvements and reach low final error values, especially on high-quality and reasoning-oriented subsets. Broad web-style slices remain comparatively harder, consistent with the deliberate capacity trade-off toward specialized competence rather than broad factual coverage.

Task-level behavior remains heterogeneous. While several tasks demonstrate strong and stable performance (e.g., PIQA, SciQ, COPA), the newly introduced harder evaluation components expose persistent weaknesses in broad-knowledge and hard science-style reasoning.

Despite these domain-specific bottlenecks, contextualizing the results against external baselines reveals the remarkable token efficiency of our training curriculum. As shown in Table VII, our ∼492M parameter model—trained on merely 22 billion tokens—matches or outperforms legacy models exposed to vastly larger corpora. For instance, it achieves an ARC-Challenge accuracy of 0.25 and an OpenBookQA score of 0.31, outperforming both GPT-2 Medium [5] (∼10B tokens, lacking reasoning focus) and Pythia-410M [53] (300B tokens). Ultimately, the pre-training curriculum achieves its primary goal: validating that a deep-and-narrow architecture combined with a curated, math-and-code-heavy mixture can yield highly competitive logical reasoning capabilities.

IV. Phase 2: Supervised Fine-Tuning (SFT)

Supervised Fine-Tuning was performed on the pretrained model from stage 3, to align it with instruction-following and assistant-style dialogue behavior [24]. The SFT framework of nanotron was used, using the same run scrips for the case that the dataset is of sft type and sftdataloading is enabled. The training objective is next-token cross-entropy but applied to chat/instruction-formatted data. Additionaly, with prompt masking, so the loss is driven by the assistant response tokens rather than the user/system prompt. Primarly, this prevents the model from learning to reproduce the prompt formatting instead of learning to answer.

A. SFT Data

The dataset used for SFT training is smol-smoltalk, an explicitly curated subset of SmolTalk dataset adapted especially for "smol" models (less than 1B parameters) [27]. It is also the dataset used in the SmolLM2 training recipes, making it a well-documented default for models in the sub-billion range, whereas many alternative open instruction datasets are either noisier, less consistently formatted, or not designed with small model alignment in mind [28].

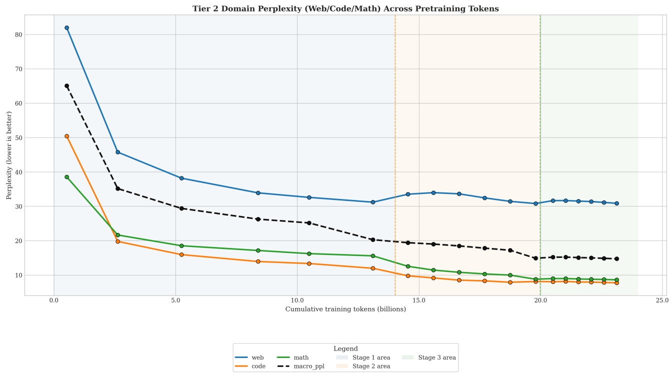
B. SFT Training

For the main SFT training the same tokenizer family as pretraining Llama-3 was used, to keep the token-ID mapping consistent with the model's learned embeddings. Changing tokenizer post-pretraining typically requires resizing/reinitializing embeddings and can degrade quality. Moreover, the standardized way to serialize a list of chat messages into model-ready text is using a chat template.

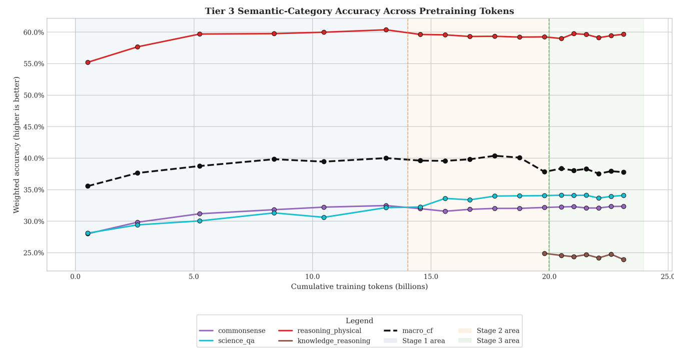
The selected tokenizer did not include a usable chat template so prompts were not being formatted consistently for chat-style SFT and inference. A custom tokenizer directory was created that embeds a Llama-3-style Jinja chat_template as documented in Llama 3 prompt format. The template renders messages with <|begin_of_text|>, role headers (system/user/assistant), and <|eot_id|> end-of-turn markers [30].

The training done was split into tokens per training step as 16,384 tokens/step. Therefore for 5000 steps the the total number of SFT tokens used is 81.92M. At the relative scale, the tokens for SFT should be some small percentage of the pretrain tokens, for this case as having 22B total pretraining tokens, this percentage is 0.37.

The weights were initialized from the Stage-3 pretrained checkpoint, but since at this stage an SFT warmup took place, the pretrained representation was preserved while the optimization dynamics were restarted for the new instruction/chat objective. Optimization used AdamW with `betas=(0.9,0.95)`, `eps=1e-8`, gradient clipping at `1.0`, and `accumulate_grad_in_fp32: true`. The learning-rate (LR) schedule used a peak LR of `1e-5`, with a



(a) Tier-2 domain trends (web/code/math) with overlaid macro PPL (dashed black), vs cumulative tokens.



(b) Tier-3 semantic-category trends with overlaid macro CF (dashed black), vs cumulative tokens.

Fig. 4: Pre-training evaluation trends with macro context embedded in each detailed plot.

TABLE VII: Tier-3 reasoning/QA task values at final checkpoint of each stage, compared against external baselines. "–" means not evaluated.

Task	**Stage 1**	**Stage 2**	**Stage 3**	**GPT-2 (345M [5]**	**Pythia-160M [53]**	**Pythia-410M [53]**
`arc_easy`	0.28	0.27	0.28	–	0.43	0.52
`arc_challenge`	–	–	**0.25**	–	0.22	0.24
`openbookqa`	0.28	0.31	**0.31**	0.18	0.26	0.29
`qasc`	0.13	0.15	0.14	–	0.17	0.22
`sciq`	0.54	0.59	0.60	0.74	0.74	0.79
`commonsense_qa`	0.23	0.23	0.24	–	0.21	0.23
`hellaswag`	0.32	0.32	0.32	0.29	0.30	0.40
`copa`	0.64	0.60	0.60	–	0.53	0.58
`piqa`	0.67	0.65	0.66	0.63	0.59	0.67
`winogrande`	0.51	0.51	0.51	0.50	0.50	0.54
`mmlu`	–	–	0.24	–	0.26	0.29

linear warmup of `200` steps, followed by cosine decay over `5000` steps.

In sft a lower learning rate helps to avoid overwriting pretrained knowledge and reduces the risk of catastrophic forgetting, as in this part of training the already-trained network is updated on a narrower, behavior-focused dataset [29].

C. Evaluation

Architecture and objective. SFT is evaluated with a bridge-plus-checkpoints protocol in *token space*. The pre-SFT bridge is the end of pre-training at **23.13B cumulative pre-training tokens** SFT evaluations are reported at **8.19M**, **24.58M**, **57.34M**, and **81.92M** SFT tokens. We keep the same modeling and reasoning suites for comparability and add an SFT-native prompt track for instruction behavior.

Methodology for the SFT-native prompt suite. We use a fixed 50-prompt set grouped into eight categories: `format_following`, `structured_output`, `code_generation`, `instruction_priority`, `instruction_following`, `reasoning_short`, `safety_behavior`, and `chat_cleanliness`. For each response, we apply deterministic checks: instruction match, strict all-check pass, format validity, chat cleanliness, and type-specific checks. All prompts and checks are identical across token checkpoints.

Manual Vibe Tests. Some vibe tests done manually as following:
Prompt: Who is the current President of the United States?
Completion: The current President of the United States, also known as the President of the Republic of the United States (U.S.) or simply "the President" in the modern sense.

Prompt: You have a 4-hour study session. Create a plan with 6 time blocks. Include: (a) 2 short breaks, (b) one block for "active recall", (c) one block for "practice problems".Explain why the order is optimal in 3 sentences
Completion: Plan your next steps on a daily basis. I am here to help you stay focused and make progress.
To optimize learning, we must divide our sessions into two groups of three or four:
1-30 minutes each day: 15 minute intervals that include 20 minutes for review during class time.
Every day: 30-45-minute blocks.
If possible, complete at least every other day - 60-90 minutes after classes where you can go over previous week's notes from previous days so far

Prompt: for x complex number, calculate the roots of equation 10x^2+3x+2 = 0
Compltetion: To find the roots of the equation 10x^2+3x+2 = 0, we can use the quadratic formula. The quadratic formula is x = (-b ± (b^2 - 4ac) 2a, where a = 10, b = 3, c = 2, and a =

2. The roots of the equation are x = 0, 1, 1, and 2.

From the qualitative “vibe tests,” the model shows weak factual reliability and poor instruction adherence on knowledge-based queries. On structured planning tasks, the model partially captures the intent but fails to satisfy key constraints. For short, well-defined reasoning problems, the model attempts to apply standard solution procedures (e.g., invoking the quadratic formula), which indicates some retained “tool-use” patterns.

However, execution accuracy is inconsistent. Overall, the model appears more capable of initiating appropriate reasoning steps in small contexts than of producing correct, verifiable results or faithfully meeting multi-part constraints.

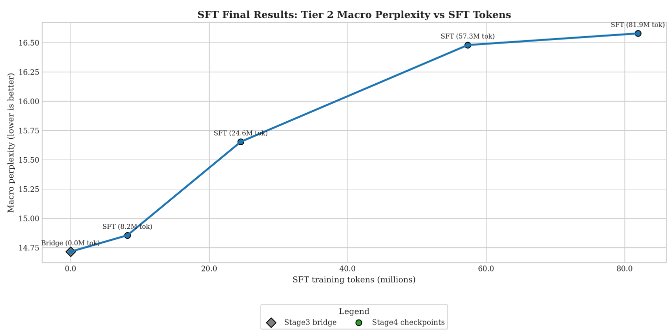


Fig. 5: Modeling macro across SFT tokens.

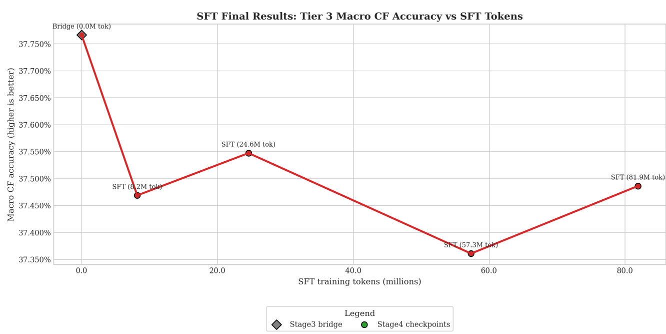


Fig. 6: Reasoning/QA macro across SFT tokens.

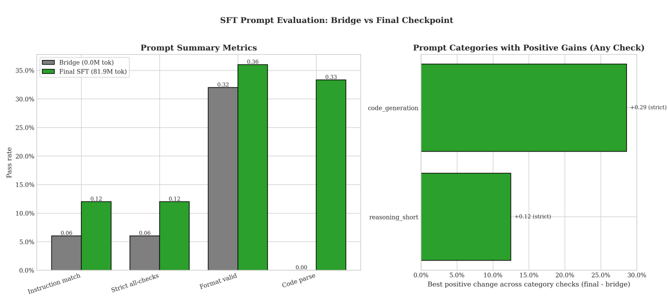


Fig. 7: SFT-native prompt behavior at strong categories.

The results. SFT produces immediate gains in prompt-facing behavior, while modeling quality degrades at higher SFT token counts. Strict instruction-compliance improves early and stays above bridge, but this comes with progressively higher modeling error at later checkpoints. Reasoning/QA macro stays near bridge, without clear monotonic improvement. Category gains are concentrated in `code_generation` and `reasoning_short`, indicating targeted adaptation rather than broad uplift across prompt families. Overall, the pattern is alignment-dominant: stronger output control, but limited transfer to general downstream reasoning benchmarks.

TABLE VIII: Prompt-summary metrics: base pre-training (0.00M SFT tokens) vs final SFT (81.92M SFT tokens).

Metric	Bridge	Final SFT	Δ
Instruction match rate	0.06	0.12	+0.06
All-check pass rate	0.06	0.12	+0.06
Format-valid rate	0.32	0.36	+0.04
Code-parse rate	0.00	0.33	+0.33

Where the model remains weak. Strict structured output (JSON validity) and strict instruction-priority/following remain weak. Several broad QA categories also stay flat. At higher SFT-token checkpoints, modeling error increases versus bridge, indicating drift.

SFT is a *partial success* with a clear trade-off: it improves instruction-format behavior and code-response validity, but does not uniformly improve broad QA and can regress base modeling quality late in training. A likely reason is objective mismatch: pre-training is code/math-focused, while part of SFT evaluation targets broader instruction and general knowledge. In practice, earlier checkpoints favor capability preservation; later checkpoints favor stricter format compliance.

V. Limitations and Future Work

While our deep-and-narrow architectural approach yielded strong reasoning performance, the model’s development was inherently constrained by several practical factors. Primarily, the project was governed by a strict temporal deadline, limiting the total allowable training time to merely a few days. This restrictive wall-clock constraint precluded extended pretraining runs, exhaustive hyperparameter sweeps, or the exploration of alternative data curricula. Furthermore, although our Leonardo HPC allocation provided substantial compute capacity, the restriction to a single 4-GPU node limited our ability to fully leverage advanced 3D parallelism techniques—such as Pipeline or Tensor Parallelism—confining our infrastructure to pure Data Parallelism with gradient accumulation.

Addressing these constraints forms the foundation of our future work. First, we intend to explore context window extension. Although the current model was constrained to a 1024-token sequence length during training to maximize batch throughput under our memory limits, applying techniques such as RoPE scaling and continuous pretraining on long-document corpora would allow the network to fully utilize its theoretical 2048-token capacity.

More fundamentally, we intend to shift our training paradigm from building a capable generalist to engineering a dedicated specialist. Rather than simply enriching our current data mixture with additional mathematical and algorithmic content, future iterations will aggressively prune broad-domain web text to actively reduce the model's generalized knowledge base. By intentionally sacrificing universal linguistic competence and broad factual recall, we aim to dedicate the entirety of the model's constrained parameter budget exclusively to domain-specific logical reasoning and mathematical execution.

Finally, the highly optimized $\sim$492M parameter footprint of our model makes it an ideal candidate for edge computing. Future work will focus on applying post-training quantization (such as 4-bit or 8-bit weight downcasting) and developing a lightweight, local user interface (UI). This will facilitate seamless, offline deployment on consumer hardware and mobile devices, guaranteeing data privacy and zero-latency inference without reliance on cloud infrastructure.

VI. Conclusion

In this report, we described the development of a 0.5B-parameter LLaMA-style model trained under realistic compute constraints, following the engineering perspective promoted by the Smol Training Playbook. Our main takeaway is that, at the sub-billion scale, performance is shaped less by sheer scale and more by deliberate design choices: we adopt a deep-and-narrow topology to prioritize targeted proficiency in logical reasoning, and we demonstrate that a carefully engineered small model can exhibit strong behavior despite limited capacity. We hope that this work helps make high-quality LLM development more reproducible and accessible, and we plan to extend it with broader evaluation, longer-context experiments, and improved post-training alignment.

Acknowledgment

We acknowledge the EuroHPC Joint Undertaking for awarding this project access to the Leonardo supercomputer and its associated resources, hosted by CINECA in Italy.

References

[1] HuggingFaceTB, "The Smol Training Playbook: The Secrets to Building World-Class LLMs," *Hugging Face Spaces*, 2025. [Online]. Available: https://huggingface.co/spaces/HuggingFaceTB/smol-playbook-toc. Accessed: 19-Feb-2026.

[2] Hugging Face, "nanotron: Minimalistic large language model 3D-parallelism training," *GitHub repository*, 2025. [Online]. Available: https://github.com/huggingface/nanotron. Accessed: 19-Feb-2026.

[3] A. Vaswani *et al.*, "Attention is All You Need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[4] A. Dubey *et al.*, "The Llama 3 Herd of Models," *arXiv preprint arXiv:2407.22706*, 2024. [Online]. Available: https://arxiv.org/abs/2407.22706. Accessed: 19-Feb-2026.

[5] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language Models are Unsupervised Multitask Learners," *OpenAI blog*, vol. 1, no. 8, 2019.

[6] Z. Liu *et al.*, "MobileLLM: Optimizing Sub-billion Parameter Language Models for On-Device Use Cases," in *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

[7] O. Press and L. Wolf, "Using the Output Embedding to Improve Language Models," *arXiv preprint arXiv:1608.05859*, 2016. [Online]. Available: https://arxiv.org/abs/1608.05859. Accessed: 19-Feb-2026.

[8] J. Ainslie *et al.*, "GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints," *arXiv preprint arXiv:2305.13245*, 2023. [Online]. Available: https://arxiv.org/abs/2305.13245. Accessed: 19-Feb-2026.

[9] N. Shazeer, "GLU Variants Improve Transformer," *arXiv preprint arXiv:2002.05202*, 2020. [Online]. Available: https://arxiv.org/abs/2002.05202. Accessed: 19-Feb-2026.

[10] B. Zhang and R. Sennrich, "Root Mean Square Layer Normalization," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[11] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu, "RoFormer: Enhanced Transformer with Rotary Position Embedding," *arXiv preprint arXiv:2104.09864*, 2021. [Online]. Available: https://arxiv.org/abs/2104.09864. Accessed: 19-Feb-2026.

[12] HuggingFaceTB, "Cosmopedia," *Hugging Face Datasets*, 2024. [Online]. Available: https://huggingface.co/datasets/HuggingFaceTB/cosmopedia. Accessed: 19-Feb-2026.

[13] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "ZeRO: Memory Optimizations Toward Training Trillion Parameter Models," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC20)*, 2020.

[14] T. Chen, B. Xu, C. Zhang, and C. Guestrin, "Training Deep Nets with Sublinear Memory Cost," *arXiv preprint arXiv:1604.06174*, 2016. [Online]. Available: https://arxiv.org/abs/1604.06174.

[15] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization," in *International Conference on Learning Representations (ICLR)*, 2019.

[16] Hugging Face, "How to Create Large-scale Synthetic Data for Pre-training (Cosmopedia)," 2024. [Online]. Available: https://huggingface.co/blog/cosmopedia. Accessed: 19-Feb-2026.

[17] Cerebras Systems, "SlimPajama: A 627B token, cleaned and deduplicated version of RedPajama," 2023. [Online]. Available: https://www.cerebras.ai/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama. Accessed: 19-Feb-2026.

[18] HuggingFaceFW, "fineweb-edu," *Hugging Face Datasets*, 2024. [Online]. Available: https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu. Accessed: 19-Feb-2026.

[19] G. Penedo, H. Kydlíček, L. Ben Allal, A. Lozhkov, M. Mitchell, C. Raffel, L. von Werra, and T. Wolf, "The FineWeb Datasets: Decanting the Web for the Finest Text Data at Scale," *arXiv preprint arXiv:2406.17557*, 2024. [Online]. Available: https://arxiv.org/abs/2406.17557. Accessed: 19-Feb-2026.

[20] Stack Exchange, "Stack Exchange Data Dump - 2023-03-08," *Internet Archive*, 2023. [Online]. Available: https://archive.org/details/stack-exchange-data-dump-2023-03-08. Accessed: 19-Feb-2026.

[21] Stack Exchange, "Announcing a change to the data-dump process," *Meta Stack Exchange*, 2024. [Online]. Available: https://meta.stackexchange.com/questions/401324/announcing-a-change-to-the-data-dump-process. Accessed: 19-Feb-2026.

[22] NVIDIA, "OpenMathReasoning," *Hugging Face Datasets*, 2025. [Online]. Available: https://huggingface.co/datasets/nvidia/OpenMathReasoning. Accessed: 19-Feb-2026.

[23] I. Moshkov, D. Hanley, I. Sorokin, S. Toshniwal, C. Henkel, B. Schifferer, W. Du, and I. Gitman, "AIMO-2 Winning Solution: Building State-of-the-Art Mathematical Reasoning Models with OpenMathReasoning dataset," *arXiv preprint arXiv:2504.16891*, 2025. [Online]. Available: https://arxiv.org/abs/2504.16891. Accessed: 19-Feb-2026.

[24] L. Ouyang *et al.*, "Training language models to follow instructions with human feedback," in *Advances in Neural Information Processing Systems*, vol. 35, pp. 27730–27744, 2022.

[25] BigCode, "StarCoder Training Dataset (starcoderdata)," *Hugging Face Datasets*, 2023. [Online]. Available: https://huggingface.co/datasets/bigcode/starcoderdata. Accessed: 19-Feb-2026.

[26] R. Li *et al.*, "StarCoder: may the source be with you!" *arXiv preprint arXiv:2305.06161*, 2023. [Online]. Available: https://arxiv.org/abs/2305.06161. Accessed: 19-Feb-2026.

[27] HuggingFaceTB, "HuggingFaceTB/smol-smoltalk," Hugging Face Datasets, 2025. [Online]. Available: https://huggingface.co/datasets/HuggingFaceTB/smol-smoltalk. Accessed: 19-Feb-2026.

[28] L. Ben Allal *et al.*, "SmolLM2: When Smol Goes Big – Data-Centric Training of a Small Language Model," *arXiv preprint arXiv:2502.02737*, 2025. [Online]. Available: https://arxiv.org/abs/2502.02737. Accessed: 19-Feb-2026.

[29] W. Huang, A. Cheng, and Y. Wang, "Mitigating Catastrophic Forgetting in Large Language Models with Forgetting-aware Pruning," in *Proc. of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Suzhou, China, Nov. 2025, pp. 21842–21856, doi: 10.18653/v1/2025.emnlp-main.1108.

[30] Meta, "Llama 3 — Model Cards and Prompt formats (Prompt format and special tokens)," Meta Llama Documentation, n.d. [Online]. Available: https://www.llama.com/docs/model-cards-and-prompt-formats/meta-llama-3/. Accessed: 19-Feb-2026.

[31] C. Raffel *et al.*, "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," *JMLR*, vol. 21, no. 140, pp. 1–67, 2020.

[32] Hugging Face, "codeparrot/codeparrot-clean," *Hugging Face Datasets*, 2021. [Online]. Available: https://huggingface.co/datasets/codeparrot/codeparrot-clean. Accessed: 19-Feb-2026.

[33] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer Sentinel Mixture Models," in *Proc. ICLR*, 2017.

[34] HuggingFaceTB, "finemath," *Hugging Face Datasets*, 2024. [Online]. Available: https://huggingface.co/datasets/HuggingFaceTB/finemath. Accessed: 19-Feb-2026.

[35] Wikimedia Foundation, "wikimedia/wikipedia," *Hugging Face Datasets*, 2023. [Online]. Available: https://huggingface.co/datasets/wikimedia/wikipedia. Accessed: 19-Feb-2026.

[36] common-pile, "stackv2_edu_filtered," *Hugging Face Datasets*, 2024. [Online]. Available: https://huggingface.co/datasets/common-pile/stackv2_edu_filtered. Accessed: 19-Feb-2026.

[37] BigCode, "starcoder2data-extras," *Hugging Face Datasets*, 2024. [Online]. Available: https://huggingface.co/datasets/bigcode/starcoder2data-extras. Accessed: 19-Feb-2026.

[38] NVIDIA, "OpenMathInstruct-1," *Hugging Face Datasets*, 2025. [Online]. Available: https://huggingface.co/datasets/nvidia/OpenMathInstruct-1. Accessed: 19-Feb-2026.

[39] NVIDIA, "OpenCodeReasoning," *Hugging Face Datasets*, 2025. [Online]. Available: https://huggingface.co/datasets/nvidia/OpenCodeReasoning. Accessed: 19-Feb-2026.

[40] P. Clark *et al.*, "Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge," *arXiv preprint arXiv:1803.05457*, 2018.

[41] T. Mihaylov, P. Clark, T. Khot, and A. Sabharwal, "Can a Suit of Armor Conduct Electricity? A New Dataset for Open Book Question Answering," in *Proc. EMNLP*, 2018.

[42] P. Khot *et al.*, "QASC: A Dataset for Question Answering via Sentence Composition," in *Proc. AAAI*, 2020.

[43] T. M. A. Welbl, N. F. Liu, and M. Gardner, "Crowdsourcing Multiple Choice Science Questions," in *Proc. Workshop on Noisy User-generated Text (W-NUT)*, 2017.

[44] A. Talmor, J. Herzig, N. Lourie, and J. Berant, "CommonsenseQA: A Question Answering Challenge Targeting Commonsense Knowledge," in *Proc. NAACL-HLT*, 2019.

[45] R. Zellers *et al.*, "HellaSwag: Can a Machine Really Finish Your Sentence?" in *Proc. ACL*, 2019.

[46] M. Sap *et al.*, "Social IQa: Commonsense Reasoning about Social Interactions," in *Proc. EMNLP-IJCNLP*, 2019.

[47] A. Wang *et al.*, "SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[48] Y. Bisk *et al.*, "PIQA: Reasoning about Physical Commonsense in Natural Language," in *Proc. AAAI*, 2020.

[49] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi, "WinoGrande: An Adversarial Winograd Schema Challenge at Scale," in *Proc. AAAI*, 2020.

[50] D. Hendrycks *et al.*, "Measuring Massive Multitask Language Understanding," in *Proc. ICLR*, 2021.

[51] HuggingFaceTB, "SmolLM3-3B," *Hugging Face Models*, 2025. [Online]. Available: https://huggingface.co/HuggingFaceTB/SmolLM3-3B. Accessed: 19-Feb-2026.

[52] L. Ben Allal, A. Lozhkov, G. Penedo, T. Wolf, and L. von Werra, "SmolLM-Corpus," *Hugging Face Datasets*, 2024. [Online]. Available: https://huggingface.co/datasets/HuggingFaceTB/smollm-corpus. Accessed: 19-Feb-2026.

[53] S. Biderman *et al.*, "Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling," in *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023.