

# The Joy and Pain of Training an LLM from Scratch

## A Technical Report on the Development of the Zagreus and Nesso Model Families

mii-llm Community  
*Made in Italy – Large Language Model*  
<https://mii-llm.ai>

### I. MOTIVATION: THE VISION OF SOVEREIGN EDGE INTELLIGENCE

Training a fully functional modern neural network, specifically a Large Language Model (LLM), from first principles has been a foundational ambition since the inception of our community [mii-llm](#) that stands for Made in Italy - Large Language Model. In the current landscape, the convergence of distributed computing power and accessible knowledge has never been more potent; consequently, constructing an intelligent machine stands as one of the most exciting tasks a group of machine learning specialists can undertake.

This vision materialized when Antonio Baldassarra (CEO of Seeweb) and Marco Cristofanilli (Head of AI at Seeweb) commissioned us to develop a Small Language Model (SLM) from scratch utilizing the Seeweb infrastructure. Seeweb, a cloud provider with a strategic focus on AI, granted us access to a cluster of on-demand nodes comprising a total of 64 NVIDIA A100 GPUs.

Our primary objective was to experiment and deliver a state-of-the-art SLM with approximately 500 million parameters, built from the ground up and optimized for edge use cases within the Italian language ecosystem. We hypothesize that, in the coming years, intelligent devices—and virtually any hardware equipped with a chip—will be enhanced by neural architectures with embedded reasoning and language capabilities. Small, efficient models will be key to enabling automation at the edge. To address this need, we created Zagreus, arguably one of the few high-performing small language models dedicated to the Italian language.

In the spirit of open and reproducible research, we are releasing the full Zagreus and Nesso lineup: seven models in total—four base (pretrained) checkpoints for bilingual models and three post-trained variants. Notably, our post-trained models are designed to compete on standard benchmarks with state-of-the-art models of comparable size, demonstrating that carefully engineered small models can achieve near frontier-level performance within their parameter class.

#### *Base models:*

- zagreus-0.4B-base-ita English Italian bilingual model
- zagreus-0.4B-base-spa English Spanish bilingual model
- zagreus-0.4B-base-por English Portuguese bilingual model
- zagreus-0.4B-base-fra English French bilingual model

#### *Post-trained models:*

- Nesso-0.4B-instruct English Italian for conversational use cases
- Nesso-0.4B-agentic English Italian for agentic and function calling use cases
- Open-Zagreus-0.4B Fully open source data used to train this model

We are releasing this detailed blog post, covering every step and data point required to reproduce the project, as we strongly believe in the importance of open source in reducing technological and geopolitical dependencies.

### II. TECHNOLOGY STACK: FRAMEWORK SELECTION

There are numerous frameworks available for creating an LLM from scratch. We conducted a comparative analysis of several options. Below is a summary of our testing and the rationale behind our ultimate decision to utilize Nanotron by Hugging Face.

#### *A. Framework Comparative Analysis*

**Megatron-LM:** Developed by NVIDIA, this is a powerful framework designed for training large transformer models with billions of parameters. While it is likely an optimal choice for large, well-resourced teams, we found it challenging to set up and deploy effectively on our specific cluster infrastructure.

**Llama-Factory:** A versatile and user-friendly open-source framework that simplifies fine-tuning, training, and deployment of a wide range of LLMs. However, our evaluation suggests it is more specialized for fine-tuning than for pre-training from scratch.

**nanoGPT and nanochat:** Both created by Andrej Karpathy, these projects prioritize simplicity and educational value.

**nanoGPT** is a minimalist, readable codebase designed as a learning tool, though it is now considered deprecated in favor of its successor. **nanochat** is the evolution of nanoGPT, offering a full-stack, end-to-end pipeline for building a complete ChatGPT-like chatbot. It covers the entire lifecycle, from tokenization and pre-training to fine-tuning and a web interface, all within a compact and hackable codebase. Although nanochat had not yet been released when we commenced this project, we believe it has a promising future, especially given its recent integration into the Transformers library.

#### *B. Our Choice: Hugging Face Nanotron*

Ultimately, we selected [Hugging Face Nanotron](#). It is a minimalistic library focused on 3D parallelism (Data, Tensor,

and Pipeline) specifically for pre-training transformer models. We value Hugging Face for its commitment to openness. We found the library well-suited for multi-node training; furthermore, it is natively integrated into the Hugging Face ecosystem (Accelerate, Datasets, hf-cli), ensuring that workflows—from data tokenization to model release—remain cohesive.

During the development cycle, we identified minor bugs and are actively contributing to the library via Pull Requests. We also established a [fork of Nanotron](#) optimized to run directly on a Slurm cluster.

### III. DATA ENGINEERING: THE TOKENIZATION PIPELINE

Data is the *sine qua non* for creating an LLM. The volume of data required is contingent upon the target model size and the available compute budget. Operating as a GPU-constrained team—and thanks to the sponsorship from Seeweb—we chose to build a small language model of  $\sim 500$  million parameters, trained on approximately 1 trillion tokens.

#### A. Dataset Sources

We utilized exclusively open source datasets by the Hugging Face team for creating our four bilingual foundational model released. Below is the data distribution per model:

##### **mii-llm/nesso-0.4B-ita:**

- <https://huggingface.co/datasets/HuggingFaceFW/fineweb/viewer/sample-350BT> (350 billion tokens)
- [https://huggingface.co/datasets/HuggingFaceFW/fineweb-2/viewer/ita\\_Latn](https://huggingface.co/datasets/HuggingFaceFW/fineweb-2/viewer/ita_Latn)
- [https://huggingface.co/datasets/HuggingFaceFW/finepdfs/viewer/ita\\_Latn](https://huggingface.co/datasets/HuggingFaceFW/finepdfs/viewer/ita_Latn)
- <https://huggingface.co/datasets/bigcode/starcoderdata> (250 billion tokens)

##### **mii-llm/nesso-0.4B-fra:**

- <https://huggingface.co/datasets/HuggingFaceFW/fineweb/viewer/sample-350BT> (350 billion tokens)
- [https://huggingface.co/datasets/HuggingFaceFW/fineweb-2/viewer/fra\\_Latn](https://huggingface.co/datasets/HuggingFaceFW/fineweb-2/viewer/fra_Latn)
- [https://huggingface.co/datasets/HuggingFaceFW/finepdfs/viewer/fra\\_Latn](https://huggingface.co/datasets/HuggingFaceFW/finepdfs/viewer/fra_Latn)
- <https://huggingface.co/datasets/bigcode/starcoderdata> (250 billion tokens)

##### **mii-llm/nesso-0.4B-por:**

- <https://huggingface.co/datasets/HuggingFaceFW/fineweb/viewer/sample-350BT> (350 billion tokens)
- [https://huggingface.co/datasets/HuggingFaceFW/fineweb-2/viewer/por\\_Latn](https://huggingface.co/datasets/HuggingFaceFW/fineweb-2/viewer/por_Latn)
- [https://huggingface.co/datasets/HuggingFaceFW/finepdfs/viewer/por\\_Latn](https://huggingface.co/datasets/HuggingFaceFW/finepdfs/viewer/por_Latn)
- <https://huggingface.co/datasets/bigcode/starcoderdata> (250 billion tokens)

##### **mii-llm/nesso-0.4B-spa:**

- <https://huggingface.co/datasets/HuggingFaceFW/fineweb/viewer/sample-350BT> (350 billion tokens)
- [https://huggingface.co/datasets/HuggingFaceFW/fineweb-2/viewer/spa\\_Latn](https://huggingface.co/datasets/HuggingFaceFW/fineweb-2/viewer/spa_Latn)

- [https://huggingface.co/datasets/HuggingFaceFW/finepdfs/viewer/spa\\_Latn](https://huggingface.co/datasets/HuggingFaceFW/finepdfs/viewer/spa_Latn)
- <https://huggingface.co/datasets/bigcode/starcoderdata> (250 billion tokens)

#### B. The Tokenization Process

Raw datasets are not ready for immediate training; they must first be tokenized. Tokenization is a CPU intensive process that transforms text strings into token sequences (numerical IDs). As a rule of thumb for storage estimation, for every 1 GB of text, approximately 3 GB of tokenized outputs are generated. For  $\sim 1$  trillion tokens, one typically requires at least 3 to 5 terabytes of disk space (depending on format, sharding strategy, and compression).

We selected the Llama-3.2 tokenizer (from the Llama-3.2-1B model) because its multilingual tokenization capabilities are robust and widely adopted. Using the [datatrove](#) library, the process took over three weeks of continuous computation to generate  $\sim 1$  trillion tokens, stratified as roughly 400B English, 400B Italian, and 200B Code.

Below is the Python script as example used for Slurm pipeline execution:

```

1 import os
2 import sys
3 from pathlib import Path
4 from datatrove.pipeline.readers import ParquetReader
5 from datatrove.pipeline.tokens import DocumentTokenizer
6 from datatrove.executor import SlurmPipelineExecutor
7
8 def create_and_run_tokenization_job(input_dir,
9     base_output_dir):
10    """
11        Create and execute a tokenization pipeline for a
12        specific directory.
13    """
14    dir_name = os.path.basename(input_dir)
15    output_dir = os.path.join(base_output_dir, f"tokenized_{dir_name}")
16
17    # Create output directory if it doesn't exist
18    os.makedirs(output_dir, exist_ok=True)
19
20    # Create pipeline for tokenization
21    pipeline = [
22        ParquetReader(
23            data_folder=input_dir,
24            glob_pattern="*.parquet",
25            text_key="text",
26        ),
27        DocumentTokenizer(
28            tokenizer_name_or_path="/hub/models--meta-llama--Llama-3.2-1B",
29            output_folder=output_dir,
30            local_working_dir=dir_name,
31            save_filename=f"{dir_name}_tokenized",
32            shuffle_documents=False,
33        ),
34    ]
35    # Configure and run the SLURM executor
36    executor = SlurmPipelineExecutor(
37        job_name=f"tokenize_{dir_name}",
38        pipeline=pipeline,
39        tasks=1,
40        workers=1,
41        time="24:00:00",
42        partition="boost_usr_prod",
43        logging_dir=os.path.join(output_dir, f"{dir_name}_logs"),
44        mem_per_cpu_gb=16,
45        slurm_logs_folder=os.path.join(output_dir, f"{dir_name}_slurm_logs"),
46    )

```

```

44     # also pass the SBATCH gres directive to ensure 0
45     # GPUs allocated
46     sbatch_args={
47         "account": "YOUR_ACCOUNT",
48         "gres": "gpu:0",
49     },
50
51     executor.run()
52
53 def main():
54     # Base paths
55     base_input_path = "INPUT_DIR"
56     base_output_path = "OUTPUTDIR"
57
58     # List of directories to process
59     # Discover directories under base_input_path instead
60     # of using a static list
61     try:
62         entries = os.listdir(base_input_path)
63     except FileNotFoundError:
64         print(f"Base input path {base_input_path} not
65             found")
66     directories = []
67     else:
68         directories = sorted(
69             [
70                 name
71                 for name in entries
72                 if os.path.isdir(os.path.join(
73                     base_input_path, name))
74                 and name.startswith("CC-MAIN")
75             ]
76         )
77
78     # Process each directory
79     # if i need other 20 directories
80     # dir_name in directories[20:40]:
81     for dir_name in directories[60:]: # Example: limit
82         # to first 20 directories
83         input_dir = os.path.join(base_input_path,
84             dir_name)
85         if os.path.exists(input_dir):
86             print(f"Launching tokenization job for {
87                 dir_name}")
88             create_and_run_tokenization_job(input_dir,
89                 base_output_path)
90         else:
91             print(f"Warning: Directory {input_dir} does
92                 not exist, skipping...")
93
94 if __name__ == "__main__":
95     main()

```

Listing 1. Tokenization pipeline script for Slurm execution

#### IV. PRE-TRAINING: THE CORE ENGINE

Pre-training is the foundational step in building an LLM, transforming raw tokenized data into a model capable of context aware text completion. This is the most time consuming and GPU intensive phase. While massive models may require thousands of GPUs, our sub 1 billion parameter model was effectively trained on the 64 GPU cluster provided by Seeweb.

We utilized Nanotron, which supports multiple architectures, including Llama-3.2, Qwen-2.5, and Mixture-of-Experts (MoE) variants. For this project, we adopted a modified Llama-3.2 fully dense architecture. Our design choice was motivated by the hypothesis that, in the small-parameter regime ( $\sim 500M$  parameters), fully dense models provide better compute utilization and more stable training dynamics than sparse architectures such as MoE. In tightly constrained capacity settings, the routing overhead and expert under-utilization typical of MoE architectures may offset their theoretical efficiency advantages.

Working with a GPU cluster is streamlined by HPC tools; we employed the Slurm scheduler. Slurm allows the cluster to be viewed as a unified Linux system where jobs can be executed across many GPUs in parallel, while handling checkpoints and logs in real time. The most challenging aspect remains ensuring the software stack—from drivers and CUDA/NCCL to Python libraries—functions harmoniously, often requiring resolution of version and ABI incompatibilities.

Successfully running a distributed training job on the tokenized data was a profound milestone. Observing the loss curve decrease from raw data after days of waiting conveys the sense of operating at the edge of scientific and engineering capability—a genuinely intense moment for a researcher.

For out-of-the-box functionality, we recommend our fork: <https://github.com/mii-llm/nanotron> (a fork of <https://github.com/huggingface/nanotron/>), pending the merge of our Pull Request.

#### A. Nanotron Training Configuration

Below is the configuration used for the pre-training run:

```

1 checkpoints:
2     checkpoint_interval: 5000
3     checkpoints_path: checkpoints_zagreus_ita_v2
4     checkpoints_path_is_shared_file_system: false
5     resume_checkpoint_path: /training/pretraining/nanotron/
6         checkpoints_zagreus_ita_v2/630000
7     save_final_state: false
8     save_initial_state: false
9     data_stages:
10    - data:
11        dataset:
12            dataset_folder:
13                - /training/pretraining/fineweb-ita/tokenized
14                - /training/pretraining/fineweb-edu-350BT/000
15                    _tokenized_output
16                    - /training/pretraining/fineweb-edu-350BT/011
17                        _tokenized_output
18                        - /training/pretraining/fineweb-edu-350BT/012
19                            _tokenized_output
20                            - /training/pretraining/fineweb-edu-350BT/013
21                                _tokenized_output
22                                - /training/pretraining/fineweb-edu-350BT/014
23                                    _tokenized_output
24                                    - /training/pretraining/fineweb-edu-350BT/015
25                                        _tokenized_output
26                                        - /training/pretraining/fineweb-edu-350BT/016
27                                            _tokenized_output
28                                            - /training/pretraining/finepdf-ita/000
29                                                _tokenized_output
30                                                - /training/pretraining/starcoder_tokenized/000
31                                                    _tokenized_output
32                                                    num_loading_workers: 0
33                                                    seed: 8
34                                                    name: stable phase
35                                                    start_training_step: 1
36 general:
37     benchmark_csv_path: null
38     consumed_train_samples: null
39     ignore_sanity_checks: true
40     project: zagreus
41     run: zagreus-350M
42     seed: 8
43     step: null
44     logging:
45         iteration_step_info_interval: 1
46         log_level: info
47         log_level_replica: info
48     model:
49         ddp_bucket_cap_mb: 100
50         dtype: bfloat16
51         init_method:
52             std: 0.03227
53         make_vocab_size_divisible_by: 1

```

```

44 model_config:
45   bos_token_id: 128000
46   eos_token_id: 128001
47   hidden_act: silu
48   hidden_size: 960
49   initializer_range: 0.02
50   intermediate_size: 2560
51   is_llama_config: true
52   max_position_embeddings: 4096
53   num_attention_heads: 15
54   num_hidden_layers: 32
55   num_key_value_heads: 5
56   pad_token_id: null
57   pretraining_tp: 1
58   rms_norm_eps: 1.0e-05
59   rope_interleaved: false
60   rope_scaling: null
61   rope_theta: 10000.0
62   tie_word_embeddings: true
63   use_cache: true
64   vocab_size: 128256
65 optimizer:
66   accumulate_grad_in_fp32: true
67   clip_grad: 1.0
68   learning_rate_scheduler:
69     learning_rate: 0.003
70     lr_decay_starting_step: 750000
71     lr_decay_steps: 50000
72     lr_decay_style: linear
73     lr_warmup_steps: 4000
74     lr_warmup_style: linear
75     min_decay_lr: 1.0e-07
76   optimizer_factory:
77     adam_beta1: 0.9
78     adam_beta2: 0.95
79     adam_eps: 1.0e-08
80     name: adamW
81     torch_adam_is_fused: true
82   weight_decay: 0.01
83   zero_stage: 0
84 parallelism:
85   dp: 64
86   expert_parallel_size: 1
87   pp: 1
88   pp_engine: lfb1b
89   recompute_layer: false
90   tp: 1
91   tp_linear_async_communication: true
92   tp_mode: REDUCE_SCATTER
93   tp_recompute_allgather: true
94   profiler: null
95   tokenizer:
96     tokenizer_max_length: null
97     tokenizer_name_or_path: meta-llama/Llama-3.2-1B
98     tokenizer_revision: null
99   tokens:
100    batch_accumulation_per_replica: 1
101    limit_test_batches: 0
102    limit_val_batches: 0
103    micro_batch_size: 4
104    sequence_length: 4096
105    train_steps: 2000000
106    val_check_interval: 5000

```

Listing 2. Nanotron pre-training configuration (YAML)

## B. Slurm Execution

The command for launching Nanotron on Slurm with 64 GPUs across 8 nodes (based on the provided configuration context) is as follows:

```

1 #SBATCH --job-name=350_it
2 #SBATCH --account=YOUR_ACCOUNT
3 #SBATCH --partition=PARTITION
4 #SBATCH --nodes=8          # 4 nodes
5 #SBATCH --gres=gpu:8      # 8 A100 per node
6 #SBATCH --cpus-per-task=32
7 #SBATCH --time=4-00:00:00
8 #SBATCH --output=slurm-%j.out
9

```

```

10 ##### 0. Environment #####
11 module purge
12 module load profile/global
13 module load python/3.11 cuda/12.2 cudnn nccl gcc
14
15 source /path/to/venv/nanotron/bin/activate
16
17 export HF_HOME=/path/to/hf_home
18 export TRANSFORMERS_OFFLINE=1
19 export HF_HUB_OFFLINE=1
20 export HF_DATASETS_OFFLINE=1
21 export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
22 # NCCL over IB
23 export NCCL_IB_DISABLE=0
24 export NCCL_SOCKET_IFNAME="ib0,eno,eth"
25 export WANDB_MODE=disabled
26
27
28 ##### 1. Distributed vars #####
29 GPUS_PER_NODE=4
30 NNODES=$SLURM_JOB_NUM_NODES           # 2
31 NODE_RANK=$SLURM_NODEID               # 0 or 1
32 MASTER_ADDR=$(scontrol show hostnames $SLURM_JOB_NODELIST
| head -n1)
33 MASTER_PORT=29400                      # free port on
                                           master
34 RDZV_ID=$SLURM_JOB_ID                 # unique per job
35
36 ##### 2. Launch #####
37 srun torchrun \
38   --nnodes $NNODES \
39   --nproc_per_node $GPUS_PER_NODE \
40   --rdzv_id $RDZV_ID \
41   --rdzv_backend c10d \
42   --rdzv_endpoint $MASTER_ADDR:$MASTER_PORT \
43   /path/to/nanotron/run_train.py \
44   --config-file smolm2/zagreus_350M_ita.yaml

```

Listing 3. Slurm batch script for Nanotron pre-training

When successful, the training logs indicate the model convergence (“the magic happens”):

```

.....
12/21 01:39:51 [INFO|DP=0|TP=0|lrdn0007]: iteration:
211364 / 1500000 |
consumed_tokens: 480B | time_per_iteration_ms: 8.79K |
tokens_per_sec: 182K | tokens_per_sec_per_gpu: 699 |
global_batch_size: 1.6M | grad_norm: 0.112 | lm_loss:
2.04 |
lr: 0.0001 | model_tflops_per_gpu: 15.4 |
eta: 131 days, 1:45:39
12/21 01:40:00 [INFO|DP=0|TP=0|lrdn0007]: iteration:
211365 / 1500000 |
consumed_tokens: 480B | time_per_iteration_ms: 8.74K |
tokens_per_sec: 183K | tokens_per_sec_per_gpu: 703 |
global_batch_size: 1.6M | grad_norm: 0.105 | lm_loss:
2.06 |
lr: 0.0001 | model_tflops_per_gpu: 15.5 |
eta: 130 days, 9:39:28

```

Listing 4. Sample training log showing model convergence

## C. Model Conversion

Once checkpoints are generated, they are not compatible with the Transformers library by default. Nanotron provides a script to convert the checkpoint into a fully compatible Hugging Face model:

```

1 torchrun --nproc_per_node=1 -m examples.llama.
           convert_nanotron_to_hf \
2   --checkpoint_path=checkpoints/544000 \
3   --save_path=hf_checkpoints/544000 \
4   --tokenizer_name meta-llama/Llama-3.2-1B

```

Listing 5. Convert Nanotron checkpoint to Hugging Face format

## V. POST-TRAINING: SHAPING BEHAVIOR

Creating a base model from scratch represents a major technical achievement, and we consider this work a contribution to the open community. However, a foundation model alone—even with a fully reproducible pipeline and transparent data distribution—is rarely sufficient for direct real-world deployment. The post-training phase is responsible for shaping the model’s behavior toward practical usability.

This phase typically requires significantly fewer GPUs and a smaller data volume compared to pre-training. However, the *quality* and *curation strategy* of the data become substantially more important than raw scale.

We utilized **Axolotl** for post-training due to our extensive experience with the framework and its stability in multi-GPU environments. While we initially encountered configuration challenges when integrating it with our Slurm-based HPC setup, we successfully adapted the workflow to support distributed execution.

We possess extensive experience in post-training language models. Over the past several years, we have post-trained models for domain-specific applications including finance, cybersecurity, structured function calling, and agentic execution patterns. Through this work, we have curated a substantial internal dataset collection that enables controlled experimentation across varied instruction-following regimes.

This dataset collection, built with meticulous care and long-term iteration, constitutes a strategic asset for our research group. For this reason, we have decided not to publish it as open source, as we consider it a competitive advantage. Nevertheless, we believe that releasing the trained models and all evaluation results provides significant value to the broader community.

Most importantly, we demonstrate that we have been able to build and release a model that performs competitively head to head with state of the art models of similar parameter scale.

We are releasing three primary post-trained models:

- **Nesso-0.4B-instruct**: optimized for conversational and instruction-following use cases.
- **Nesso-0.4B-agentic**: optimized for function calling, structured outputs, and agentic execution patterns.

Both models utilize **Nesso-0.4B-ita** as the base and are trained on a bilingual corpus (English/Italian).

It is important to note that both models are currently at the **SFT (Supervised Fine-Tuning)** stage. In the coming weeks, we will execute the **DPO (Direct Preference Optimization)** stage and subsequently update both the models and their evaluation results.

We also released a third, fully open model: **Open-Zagreus-0.4B**.

Thanks to the work of the Italian open-source community **mii-llm**, and in particular Michele Montebovi who published the SFT dataset *OpenItalianData*—all data used and all training recipes for this model are fully open and reproducible as a full open source model from data to weights.

### A. Post-Training Slurm Script

```

1 #!/bin/bash
2 #SBATCH --job-name=ax_2n
3 #SBATCH --account=MII
4 #SBATCH --nodes=4                                # 2 nodes
5 #SBATCH --gres=gpu:8                            # 4 A100 per node
6 #SBATCH --cpus-per-task=32
7 #SBATCH --time=12:00:00
8 #SBATCH --output=slurm-%j.out
9
10 ##### 0. Environment #####
11 module purge
12 module load profile/global
13 module load cuda/12.2 cudnn nccl
14
15 source /training-venv/bin/activate
16
17 export HF_HOME=/
18 export TRANSFORMERS_OFFLINE=1
19 export HF_HUB_OFFLINE=1
20 export HF_DATASETS_OFFLINE=1
21 export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
22
23 # NCCL over IB
24 export NCCL_IB_DISABLE=0
25 export NCCL_SOCKET_IFNAME="ib0,eno,eth"
26
27 ##### 1. Distributed vars #####
28 GPUS_PER_NODE=4
29 NNODES=$SLURM_JOB_NUM_NODES
30 NODE_RANK=$SLURM_NODEID
31 MASTER_ADDR=$(scontrol show hostnames $SLURM_JOB_NODELIST
| head -n1)
32 MASTER_PORT=29400
33 RDZV_ID=$SLURM_JOB_ID
34
35 ##### 2. Launch #####
36 srun torchrun \
    --nnodes $NNODES \
    --nproc_per_node $GPUS_PER_NODE \
    --rdzv_id $RDZV_ID \
    --rdzv_backend c10d \
    --rdzv_endpoint $MASTER_ADDR:$MASTER_PORT \
    -m axolotl.cli.train \
    training/opendata-zagreus-350M-sft-fsdp-debug.yaml

```

Listing 6. Slurm batch script for Axolotl post-training

### B. Axolotl Configuration

```

1 base_model: giux78/zagreus-0.4B-ita
2 strict: false
3 output_dir: ./ale_outputs/opendata-zagreus-sft-final
4 seed: 42
5 chat_template_jinja: "{%- for message in messages -%}\n"
6   {{- \"<|im_start|>\" + message.role + \"\\n\"}}
7   + message.content + \"<|im_end|>\" + \"\\n\\n\" -}}\n"
8 {%- endfor -%}\n"
9 {%- if add_generation_prompt -%}\n"
10 \t{{- \"<|im_start|>assistant\\n\" -}}\n"
11 {%- endif -%}"
12 datasets:
13   - path: /training/openitaliandata
14     type: chat_template
15     field_messages: conversation
16     roles_to_train: ["assistant"]
17     train_on_eos: turn
18
19 dataset_prepared_path: ./ale_outputs/dataset_cache/
opendata-zagreus-sft
20
21 sequence_len: 4096
22 sample_packing: true
23 eval_sample_packing: true
24 pad_to_sequence_len: true
25
26 # Cosine schedule knobs
27 cosine_constant_lr_ratio: 0.8
28 cosine_min_lr_ratio: 0.3
29
30 optimizer: adamw_torch_fused

```

```

31 lr_scheduler: constant
32 learning_rate: 1.0e-03
33
34 max_grad_norm: 1.0
35 micro_batch_size: 1
36 gradient_accumulation_steps: 8
37
38 num_epochs: 3
39
40 bf16: auto
41 flash_attention: true
42 gradient_checkpointing: true
43
44 logging_steps: 10
45 eval_strategy: steps
46 eval_steps: 300
47 save_strategy: steps
48 save_steps: 500
49 save_total_limit: 3
50 val_set_size: 10000
51
52 fsdp_config:
53   fsdp_sharding_strategy: FULL_SHARD
54   fsdp_auto_wrap_policy: TRANSFORMER_BASED_WRAP
55   fsdp_transformer_layer_cls_to_wrap: LlamaDecoderLayer
56   fsdp_backward_prefetch_policy: BACKWARD_PRE
57   fsdp_state_dict_type: FULL_STATE_DICT
58
59 special_tokens:
60   pad_token: <|im_end|>
61   eos_token: <|im_end|>

```

Listing 7. Axolotl SFT configuration

## VI. PRE-TRAINED FOUNDATIONAL MODELS EVALUATIONS

This section presents quantitative evaluations of our pre-trained foundational models. We include multiple data points to demonstrate how our data curation strategy and architectural configuration enabled the training of competitive small language model families.

These results serve both as validation and as a reproducible baseline for future experiments.

We are contributors to **lm-evaluation-harness** for multilingual benchmarks and relied extensively on this framework. For each benchmark, we provide the exact command used to ensure the evaluation reproducibility.

### A. Zagreus-0.4B-ita-base

#### 1) Evaluation Command:

```

1 lm-eval --model hf --model_args pretrained=checkpoint \
2   --tasks m_mmlu_it --num_fewshot 5 --device cuda:0 --
3     batch_size 1
4
5 lm-eval --model hf --model_args pretrained=LiquidAI/LFM2
6   -350M \
7   --tasks hellaswag_it,arc_it --device cuda:0 --
8     batch_size 1

```

Checkpoint progression:

### B. Zagreus-0.4B-spa-base (Spanish)

#### 1) Evaluation Command:

```

1 lm-eval --model hf --model_args pretrained=checkpoint \
2   --tasks m_mmlu_es --num_fewshot 5 --device cuda:0 --
3     batch_size 1
4
5 lm-eval --model hf --model_args pretrained=LiquidAI/LFM2
6   -350M \
7   --tasks hellaswag_es,arc_es --device cuda:0 --
8     batch_size 1

```

TABLE I  
ZAGREUS-0.4B-ITA-BASE: CHECKPOINT EVALUATION

Checkpoint	mmlu_it	hellawag_it	arc_it	Media
v2-95k	0.2529	0.3366	0.2652	0.2849
v2-205k	0.2628	—	—	0.2628
v2-290k	0.2428	0.3492	0.2335	0.2752
v2-305k	0.2598	0.3562	0.2652	0.2937
v2-365k	0.2566	0.3664	0.2712	0.2981
v2-390k	0.2556	0.3438	0.2498	0.2831
v2-460k	0.2540	0.3778	0.2549	0.2956
v2-520k	0.2540	0.3778	0.2549	0.2956
v2-590k	0.2547	0.3651	0.2455	0.2884
v2-630k	0.2562	0.3632	0.2643	0.2946
v2-680k	0.2538	0.3740	0.2592	0.2957
v2-775k	0.2535	0.3750	0.2583	0.2956

Zagreus-0.4B-ita Training Progress

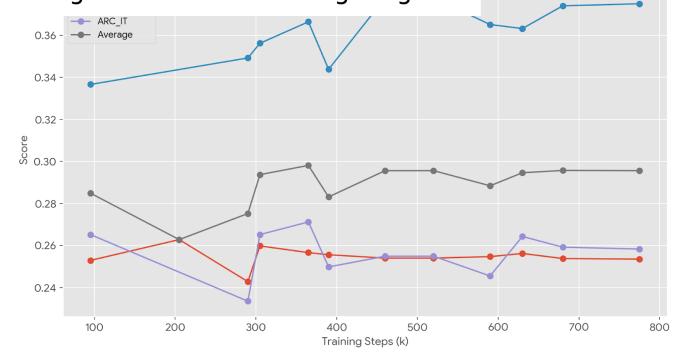


Fig. 1. Zagreus-0.4B-ita-base: training progression across checkpoints

### C. Zagreus-0.4B-fra (French)

#### 1) Evaluation Command:

```

1 lm-eval --model hf --model_args pretrained=checkpoint \
2   --tasks m_mmlu_fr --num_fewshot 5 --device cuda:0 --
3     batch_size 1
4
5 lm-eval --model hf --model_args pretrained=LiquidAI/LFM2
6   -350M \
7   --tasks hellaswag_fr,arc_fr --device cuda:0 --
8     batch_size 1

```

Evaluation procedure identical to previous sections.

### D. Zagreus-0.4B-por (Portuguese)

#### 1) Evaluation Command:

```

1 lm-eval --model hf --model_args pretrained=checkpoint \
2   --tasks m_mmlu_pt --num_fewshot 5 --device cuda:0 --
3     batch_size 1
4
5 lm-eval --model hf --model_args pretrained=LiquidAI/LFM2
6   -350M \
7   --tasks hellaswag_pt,arc_pt --device cuda:0 --
8     batch_size 1

```

### E. lm-evaluation-harness-pt

For the Portuguese base model we also evaluate against the fantastic work of [Eduardo Garcia](#), a fork of [lm-eval](#) that has also an important [leaderboard](#) comparing many open source models. Below the results and the comparison with Qwen3-0.6B-Base.

TABLE II  
ZAGREUS-0.4B-SPA-BASE: CHECKPOINT EVALUATION

Steps	mmlu_es	arc_es	hellawag_es	Average
146k	0.254	0.265	0.409	0.309
216k	0.237	0.270	0.414	0.307
292k	0.254	0.262	0.417	0.311
406k	0.254	0.269	0.423	0.315
518k	0.255	0.280	0.429	0.321

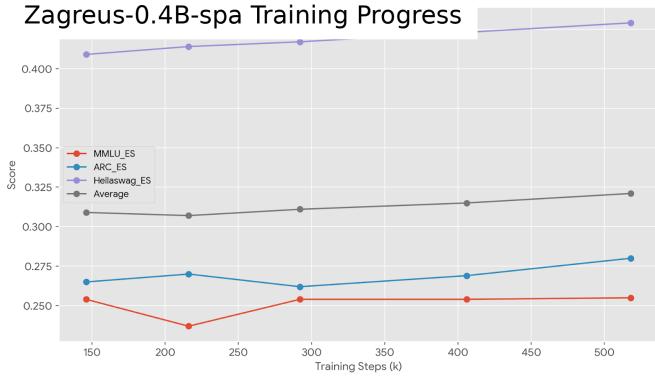


Fig. 2. Zagreus-0.4B-spa-base: training progression across checkpoints

TABLE III  
ZAGREUS-0.4B-FRA-BASE: CHECKPOINT EVALUATION

Steps	m_mmlu_fr	arc_fr	hellawag_fr	Average
129k	0.262	—	—	0.262
231k	0.263	—	—	0.263
365k	0.256	0.278	0.414	0.316
456k	0.267	—	—	0.267
603k	0.256	0.278	0.414	0.316
705k	0.266	0.281	0.417	0.321

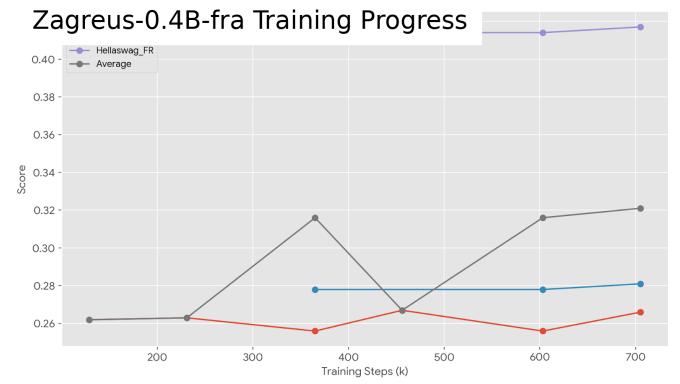


Fig. 3. Zagreus-0.4B-fra-base: training progression across checkpoints

```

1 lm_eval \
2   --model huggingface \
3   --model_args "pretrained=giux78/zagreus-3B-165000,
4     revision=main" \
5   --tasks enem_challenge,bluex,oab_exams,assin2_rte,
6     assin2_sts, \
7   faquad_nli,hatebr_offensive,portuguese_hate_speech,
8     tweetsentbr \
9   --device cuda:0 \
10  --output_path "./"

```

## VII. POST-TRAINED NESSO MODELS EVALUATIONS

In this section, we analyze the performance of **Nesso-0.4B-instruct** and **Nesso-0.4B-agentic** relative to comparable models. Since these models are pre-trained in English Italian we evaluate the models on english and italian benchmark as in the commands below.

```

1 lm-eval --model hf --model_args pretrained=checkpoint \
2   --tasks m_mmlu_it --num_fewshot 5 --device cuda:0 --
3     batch_size 1
4 lm-eval --model hf --model_args pretrained=checkpoint \
5   --tasks mmmlu --num_fewshot 5 --device cuda:0 --
6     batch_size 1
7 lm-eval --model hf --model_args pretrained=LiquidAI/LFM2
8   -350M \
9   --tasks hellawag_it,arc_it --device cuda:0 --
10    batch_size 1
11 lm-eval --model hf --model_args pretrained=LiquidAI/LFM2
12   -350M \
13   --tasks hellawag,arc --device cuda:0 --batch_size 1
14 lm-eval --model hf --model_args pretrained=LiquidAI/LFM2
15   -350M \
16   --tasks ifeval-ita --device cuda:0 --batch_size 1

```

```

16 lm-eval --model hf --model_args pretrained=LiquidAI/LFM2
17   -350M \
18   --tasks ifeval --device cuda:0 --batch_size 1

```

## A. Discussion

As observed, Qwen maintains a clear advantage on MMLU (both English and Italian). However, across several other benchmarks—particularly instruction-following and reasoning-oriented tasks—Nesso achieves competitive or superior performance.

Considering that MMLU is a widely used and often saturated benchmark, frequently incorporated into training corpora, we believe our results demonstrate that we have created a highly competitive small language model optimized for English/Italian edge inference scenarios.

## B. Open-Nesso-0.4B Evaluation

Open-Nesso-0.4B-ita is our fully open-source variant. It is based on Nesso-0.4B-ita and trained on the publicly available dataset published by Michele Montebovi.

Download: <https://huggingface.co/datasets/DeepMount00/OpenItalianData>

The model and dataset demonstrate that it is possible to build competitive English Italian language models using exclusively open-source resources.

## VIII. CONCLUSION

The *Zagreus and Nesso Model Families* project stands as a remarkable and highly important contribution to the field

TABLE IV  
ZAGREUS-0.4B-POR-BASE: CHECKPOINT EVALUATION

Checkpoint	ARC	HellaSwag	MMLU	Media
153k	0.2667	0.3732	0.2685	0.3028
207k	0.2705	0.3768	0.2671	0.3048
276k	0.2718	0.3789	0.2664	0.3057
345k	0.2564	0.3796	0.2669	0.3009
414k	0.2682	0.3842	0.2673	0.3066
483k	0.2667	0.3865	0.2658	0.3063
582k	0.2786	0.3865	0.2688	0.3113

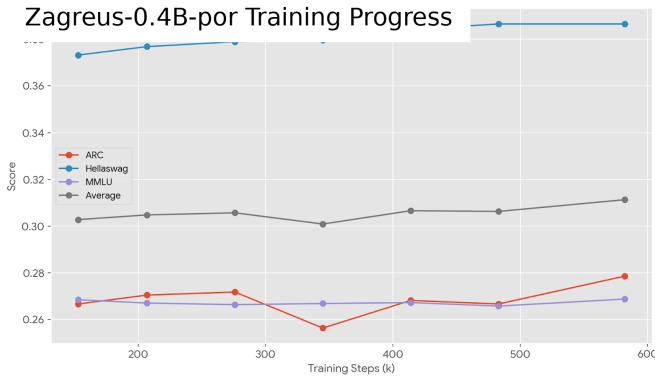


Fig. 4. Zagreus-0.4B-por-base: training progression across checkpoints

of language model research, particularly within the realm of **Small Language Models (SLMs)**. At a time when the community is largely focused on scaling models ever larger, this work demonstrates that **starting from scratch and engineering a small, efficient model can be both feasible and impactful**. The initiative directly addresses the critical need for models that are capable of **intelligent reasoning at the edge**, optimally suited for deployment on everyday devices with limited compute resources—a paradigm that will only grow in strategic importance as AI becomes more ubiquitous across hardware platforms.

This report does not merely describe a model; it **documents the entire empirical journey** of developing SLMs from first principles, from motivation and data engineering to pre-training, validation, and deployment. By releasing **seven distinct models**, including multilingual foundational checkpoints and post-trained variants optimized for conversational and agentic use cases, the project sets a new standard for reproducibility and openness in LLM research. Importantly, the work emphasizes that **carefully engineered small models can match or approach the performance of much larger counterparts on standardized benchmarks**, underlining a strategic shift in how the community can think about computational efficiency without sacrificing capability.

In addition to the scientific and technical achievements, the *Zagreus-Nesso SLM* effort embodies a broader philosophical commitment: the advancement and **democratization of AI**

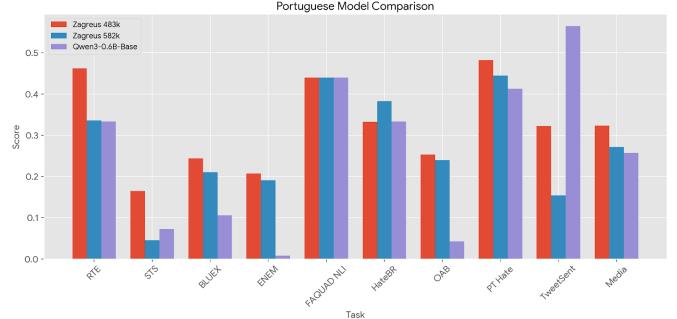


Fig. 5. Portuguese extended benchmark comparison

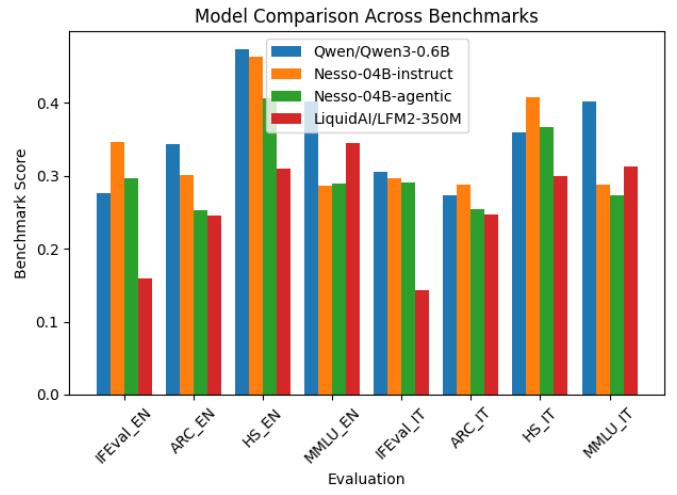


Fig. 6. Nesso model evaluation: all benchmarks combined

**through open research and open source tools.** By providing detailed data pipelines, architectural choices, and a transparent account of trade-offs encountered in training at scale, this work becomes an invaluable resource for anyone seeking to replicate or build upon it. Therefore, the importance of this report lies not only in its immediate results but also in its **lasting influence on how future small language models may be conceived, trained, and deployed**.

TABLE V  
PORTUGUESE EXTENDED BENCHMARK VS. QWEN3-0.6B-BASE

Rank	Model / Checkpoint	RTE	STS	BLUEx	ENEM	FAQUAD NLI	HateBR	OAB	PT Hate	TweetSent	Media
1 <sup>st</sup>	<b>zagreus 483k</b>	0.4624	0.1650	0.2434	0.2071	0.4397	0.3327	0.2528	0.4817	0.3220	<b>0.3230</b>
2 <sup>nd</sup>	<b>zagreus 582k</b>	0.3361	0.0449	0.2100	0.1903	0.4397	0.3825	0.2392	0.4444	0.1542	<b>0.2713</b>
3 <sup>rd</sup>	<b>Qwen3-0.6B-Base</b>	0.3333	0.0726	0.1057	0.0077	0.4397	0.3333	0.0428	0.4123	0.5646	<b>0.2569</b>

TABLE VI  
POST-TRAINED NESSO MODEL EVALUATION: ENGLISH AND ITALIAN BENCHMARKS

Model	English				Media EN	Italian				Media IT
	IFEval EN	ARC_EN	HS_EN	MMLU_EN		IFEval IT	ARC_IT	HS_IT	MMLU_IT	
Qwen/Qwen3-0.6B	0.2758	0.3430	0.4742	0.4013	0.3736	0.3058	0.2729	0.3598	0.4025	0.3353
Nesso-04B-instruct	0.3465	0.3003	0.4629	0.2871	0.3492	0.2962	0.2874	0.4076	0.2875	0.3197
Nesso-04B-agentic	0.2962	0.2534	0.4062	0.2889	0.3112	0.2914	0.2541	0.3673	0.2730	0.2965
LiquidAI/LFM2-350M	0.1595	0.2457	0.3092	0.3445	0.2647	0.1427	0.2464	0.2994	0.3132	0.2504

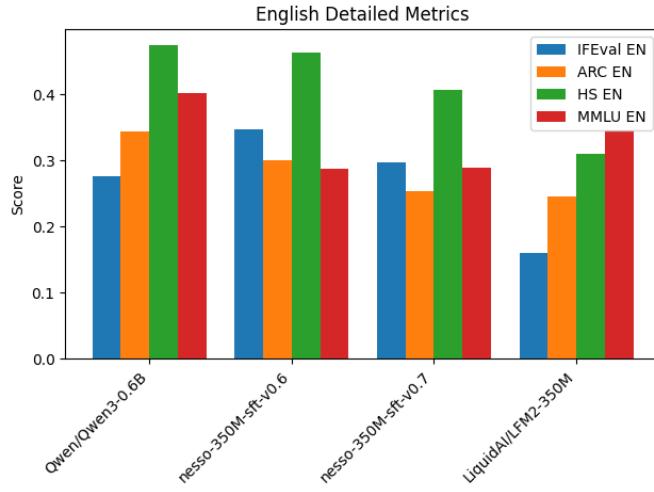


Fig. 8. Nesso model evaluation: English benchmarks