

Article

Edge-Ready Romanian Language Models: Training, Quantization, and Deployment

T. A. Diac ¹ , P. F. de Viana ¹, A. F. Neagoe ² , A. Oprea ³, M. C. Raportaru ⁴ and A. Nicolin-Źacsek ^{4,*} 

¹ Faculty of Science, Lund University, P.O. Box 117, 22100 Lund, Sweden

² Faculty of Physics, University of Bucharest, Atomistilor 405, 077125 Magurele, Romania

³ T2, Ceaus Firica, 145100 Rosiori de Vede, Romania

⁴ Institute of Space Science—INFLPR Subsidiary, Atomistilor 409, 077125 Magurele, Romania

* Correspondence: alexandru.nicolin.zacsek@spacescience.ro

Abstract

We present RoBaseLM-S (125 M) and RoBaseLM-M (260 M), two compact Romanian decoder-only language models trained from scratch on a 4.3 B-token curated corpus. Architecturally, they follow a modern LLaMA-style recipe with pre-norm RMSNorm, rotary position embeddings, SwiGLU feed-forward blocks, grouped-query attention, and 4 k-token context windows. We release both full-precision (FP16) and post-training 5-bit (Q5_K_M) checkpoints in GGUF format for lightweight local inference. The 5-bit variants fit under 500 MB and generate text in real time on a Jetson Nano 4 GB, enabling fully offline Romanian text generation on consumer-grade edge hardware. We evaluate the models intrinsically (multi-domain perplexity across news, literary prose, poetry, and heterogeneous web text) and extrinsically (LaRoSeDa sentiment classification and RO-STS sentence similarity). Relative to Romanian GPT-2-style baselines at similar parameter scales, RoBaseLM-S and RoBaseLM-M reduce perplexity substantially, e.g., from 30.7 to 15.9 on our held-out news split. The 5-bit post-training quantized checkpoints remain within FP16 performance across all reported tasks. To our knowledge, these are the first Romanian small language models explicitly optimized for long-context inference, post-training quantization, and low-power on-device deployment.

Keywords: Romanian language models; small language models; long-context transformer; post-training quantization; on-device inference; sentiment analysis; semantic textual similarity

1. Introduction

In recent years, natural language processing (NLP) for Romanian has moved from proof-of-concept demonstrations to a coherent ecosystem of data, models, and evaluations. Although Romanian is not, strictly speaking, a low-resource language, it remains underrepresented compared to other languages of similar size in terms of standardized datasets and exhaustive performance reports, despite recent efforts to build a multi-domain large-scale Romanian corpus [1]. In practice, adoption in real applications is still hindered by several challenges: up-to-date architectures, long-context capabilities, quantization-ready packages with on-device profiling, and broader, better-motivated evaluation coverage. On the data side, web-scale inputs (Common Crawl derivatives) have been essential. The OSCAR and CC-100 corpora have provided critical mass for Romanian, provided there is a disciplined pipeline: normalization, boilerplate filtering, language identification, deduplication (at the sentence and document levels), and semantic pruning [2–4]. The right choice of



Academic Editor: Isidoros Perikos

Received: 24 November 2025

Revised: 13 January 2026

Accepted: 22 January 2026

Published: 6 February 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

tokenization has a direct impact on cost and quality: for Romanian (rich morphology, diacritics, clitics), monolingual BPE/SentencePiece tokenizers trained on clean text reduce over-segmentation, shorten sequences, and help KV-cache locality during decoding [5,6]. On the evaluation side, the LiRo benchmark has standardized splits and leaderboards for diverse tasks (POS, NER, dependencies, UD morphosyntax), enabling comparisons between models [7]. Measurement frameworks have also developed considerably, as shown by LaRoSeDa for sentiment analysis [8], RONEC/RONEC-v2 for named-entity recognition [9,10], and RO-STs for sentence similarity [11]. On the modeling side, early monolingual encoders (RoBERT: small/base/large) trained exclusively on Romanian text have outperformed multilingual baselines on core tasks [12]. For social-media text, RoBERT-Tweet (trained on the entire 2008–2022 Twitter stream) improved affect/toxicity and NER on tweet-style inputs [13]. In text generation, RoGPT-2 validated monolingual pretraining for Romanian [14]. More recently, the community project OpenLLM-Ro collects base and instruction-tuned variants (RoLLaMa-2/3/3.1, RoMistral, RoGemma/Gemma2), SFT/DPO resources and evaluation utilities, and curates Romanian datasets useful for fine-tuning and reproducible testing [15]. Nevertheless, notable gaps remain for real deployment scenarios in settings with strict cost and privacy constraints: (i) small decoders with long context are missing; (ii) quantization-ready checkpoints (e.g., GGUF) are still the exception; (iii) device-level profiling reports (tokens/s, TTFT, memory, and energy/token on embeddable CPU/GPU) are rare; and (iv) evaluation coverage is narrow, and the rationale for choosing benchmarks and interpreting results is often lacking. In the public sector, where privacy and continual connectivity make offline inference an important requirement, small quantized models plus lightweight runtimes make 100–500 M-parameter LMs feasible on commodity CPUs or embeddable GPUs, with direct benefits for security (local data) and research (edge viability becomes measurable). Also on the side of future developments, let us mention the need for automatized bias-detection instruments for both SLMs and LLMs, as well as automated audit instruments for these models covering ethics and security, such as the ones developed within the Autonomous Model of Textual Assistance—MA’AT Project, co-funded by European Union (Code SMIS 2021+: 330941).

To address the aforementioned challenges we present here our results for two edge-ready Romanian decoder-only language models, dubbed RoBaseLM-S (125 M) and RoBaseLM-M (260 M), which have been trained on a 4.3 B-token curated corpus and deployed on low-power devices. This corpus was derived from 20.5 GB of curated data across three distinct sources, each providing a specific domain profile to support meaningful statistical distributions. The OSCAR-Ro and CC100-Ro datasets, which comprise roughly 60% of the curated volume, are Common Crawl derivatives that can be heuristically approximated as containing general web text (60–70%), news (10–15%), forums (10%) or social media, and the remainder of literary, legal, or institutional texts. The remaining 40% of the corpus comes from a custom crawl of 8.3 GB that was curated with a focus on high-quality, easily accessible articles from static websites. Approximately 90% of this custom crawl comprises national and regional news articles, while the adjacent 10% represents public domain literary texts and encyclopedic content.

In the following sections we present all details regarding these two models with special emphasis on the acquisition and curation of the corpus, tokenizer, quantization and evaluation methodology, which are covered in Section 2, and learning dynamics, perplexity, sentiment classification, semantic textual similarity, and edge runtime results, which are covered in Section 3. We end this introductory discussion by listing below the main points of novelty of our research:

- Introduction of RoBaseLM-S (125 M) and RoBaseLM-M (260 M): We present the first Romanian small language models (SLMs) explicitly optimized for on-device

deployment and long-context inference (4096 tokens). The models are freely available on GitHub [16].

- **Curated 4.3 B-Token Monolingual Corpus:** Models were trained from scratch on a high-quality, 20.5 GB cleaned corpus that prioritizes professional news and literary prose to ensure robust syntactic and morphological understanding.
- **Quantization-Ready Deployment:** We provide post-training 5-bit (Q5_K_M) GGUF artifacts that reduce model size by nearly 3 times, while remaining within floating-point noise of FP16 performance across all our tasks.
- **Proven Edge Viability:** We demonstrate real-time, offline Romanian text generation on low-power consumer hardware, such as the Jetson Nano and Raspberry Pi, achieving interactive rates with a footprint under 500 MB.
- **Comprehensive Performance Benchmarking:** Our results show a substantial reduction in perplexity, dropping from 30.7 to 15.9 on news text, and maintain high utility as frozen encoders for sentiment classification (LaRoSeDa) and semantic similarity (RO-STs).

2. Materials and Methods

2.1. Corpus Acquisition and Curation

The pre-training corpus combines Romanian slices of Common Crawl derivatives (**OSCAR-Ro**, **CC100-Ro**) with an original, curated multi-genre crawl (national/regional news, technology and general-interest blogs, public forums with Romanian-language threads, encyclopedic content). Following common practice for web-scale curation [2,3], we implemented a multi-stage pipeline that generates line-oriented artifacts for reproducibility and auditing as following:

Source and license screening. Exclude paywalled/private/restricted sources; retain domains with permissive or explicit reuse (press releases, institutional sites, CC encyclopedic content).

Language and script checks. Document and paragraph-level identification (Romanian vs. other languages); drop items with high mixed-language rates. Apply NFC Unicode normalization; validate Romanian diacritics (*ă, â, î, ș, ț*); fix or discard malformed encodings; keep only UTF-8 clean lines; strip control characters.

Boilerplate and markup removal. Remove navigation/cookie banners/pagination via rules and structural cues (HTML density, repeating header/footer *n*-grams). Standardize punctuation (quotes, dashes, ellipses); collapse whitespace; remove template stubs (e.g., “Comentarii”, “Articole similare”).

Noise filters. Drop lines dominated by non-linguistic artifacts (ASCII art, tracking params, long hex strings); remove residual markup/code; reject abnormal digit/symbol ratios (more than 50% of line length). Remove snippets (less than 40 characters) unless part of poetry/dialog context.

Deduplication. MinHash near-duplicate detection at paragraph/document granularity to collapse syndicated/mirrored pages; within-domain URL-stem canonicalization for multipage duplicates.

Diacritics and typography. standardize quotes/dashes and Romanian-specific abbreviations (*d-l, d-na*) to reduce spurious subwords.

Balancing. Remove pages with very low lexical diversity and auto-generated feed lists. Light domain caps to avoid overfitting to a few large outlets while preserving topical breadth (news, culture, science, everyday discourse) [2,3,17].

Deep cleaning removed roughly 80% of OSCAR and CC100 while preserving coverage (Table 1); the personal curated crawl, built under much stricter inclusion rules, was retained fully.

Table 1. Corpus size before and after deep cleaning. Public Common Crawl extractions were heavily filtered; the custom crawl was curated using a distinct pipeline.

Corpus	Original Size	Curated Size
OSCAR-Ro	47 GB	9.4 GB
CC100-Ro	16 GB	2.8 GB
Custom crawl	8.3 GB	8.3 GB
Total	71.3 GB	20.5 GB

2.2. Tokenizer Design

We train subword vocabularies with SentencePiece in Byte-Pair Encoding (BPE) mode [6] at sizes of 32 k, 40 k, and 50 k, using the cleaned 4.3 B-token Romanian corpus (Section 2.1). We evaluate each tokenizer on a 50 k-document held-out set spanning multiple genres and report sequence length statistics and token-to-word ratios.

Larger vocabularies shorten sequences but show diminishing returns beyond 40 k, with a rise in rare units that fragment inflectional paradigms. The 40 k configuration yields the best balance: approximately 3% shorter documents vs. 32 k and negligible over-specialization vs. 50 k (see Table 2). We adopt a monolingual tokenizer trained solely on Romanian to avoid cross-lingual interference. All text is normalized to NFC; punctuation/quotes are standardized; special tokens cover sentence/document boundaries and padding. Packing to 4096 tokens guided frequent merge pairs to minimize padding waste.

Table 2. Tokenizer ablation on 50,000 held-out Romanian documents. The 40 k vocabulary is the selected configuration.

Vocab Size	Avg. Len.	Median	P90	P99	Tokens/Word
32 k	408.8	379	682	861	1.47
40 k	398.7	370	665	840	1.43
50 k	389.8	362	651	821	1.40

2.3. Architecture and Training

We use compact, decoder-only transformers with pre-norm RMSNorm, RoPE positional encoding, SwiGLU MLPs, and grouped-query attention (GQA) for memory-efficient KV caching [18–21] (see Table 3). Input/output embeddings are tied. Models support a 4096-token context.

Table 3. RoBaseLM configurations (context 4096; SentencePiece 40 k).

Model	Params	Layers	d_{model}	Heads	KV Heads	MLP Dim
RoBaseLM-S (125 M)	approx. 125 M	12	768	12	2	2816
RoBaseLM-M (260 M)	approx. 260 M	22	896	14	2	3136

Optimization. AdamW with ($\beta_1 = 0.9$, $\beta_2 = 0.95$), weight decay 0.1, grad-clip 1.0, cosine LR with 1000 warmup steps, peak 3×10^{-4} . Mixed precision Bfloat16 and gradient checkpointing. Sequences packed to 4096 tokens with document-boundary markers; effective batch of approximately 16 sequences (around 1.31×10^5 tokens/step). Each model consumes approximately 4.3 B tokens (one pass), consistent with compute-optimal scaling

at this size [22]. Please note that we did not employ dropout, instead relying on several key architectural and optimization strategies to ensure stability and mitigate overfitting during our single-pass approach. In addition to the AdamW optimization method, stability was also ensured through the use of a cosine learning-rate scheduler with 1000 warmup steps, a gradient clipping threshold of 1.0, and modern architectural components such as pre-norm RMSNorm and SwiGLU feed-forward blocks.

Implementation and hardware. HuggingFace Transformers (PyTorch). Single workstation: Ryzen 5950X, 128 GB RAM, RTX 4090 24 GB. The runs for both models completed in less than two weeks, demonstrating feasibility without multi-node infrastructure [18,22]. On the software side let us notice that we used Python v3.10, PyTorch v2.1, HuggingFace Transformers v4.35, SentencePiece v0.1.99, CUDA Toolkit v12.1, llama.cpp: v0.2.20 series (November 2025), Ollama v0.1.20, NumPy v1.26, SciPy v1.11, and scikit-learn v1.3.

2.4. Quantization and Deployment Targets

We apply post-training, weight-only quantization to Q5_K_M (5-bit, block-scaled, mixed outlier path) and export the GGUF artifacts. Q5_K_M partitions weight matrices into fixed-size blocks with per-block scales. Most of the weights are stored as 5-bit ints, while outliers remain in higher precision, thus limiting reconstruction error and tracking FP16 within noise on PPL and extrinsic scores [23,24]. Compression reduces the size by a factor of approximately 2.7–3 as evidenced in Table 4. Artifacts run with llama.cpp/Ollama backends in a uniform runtime path [25]. Please note that we did not test more aggressive schemes (such as 4-bit or 2-bit) because the Q5_K_M format proved sufficiently compact for our target hardware without compromising accuracy.

Table 4. Weights-only storage for deployment (training checkpoints are larger due to optimizer states).

Model	Full (GB)	FP16 (MB)	Q5_K_M (MB)
RoBaseLM-S (125 M)	approx. 1.76	240	89
RoBaseLM-M (260 M)	approx. 3.61	501	208

Deployment devices. We target three representative devices: NVIDIA Jetson Nano 4 GB (Cortex-A57; 128-core Maxwell), Raspberry Pi 4 GB (Cortex-A72; CPU-only), and iPhone 15 (A-series SoC with neural engines). For cross-device comparability and to stress the weight-bandwidth path, we execute primarily on CPUs [26]. On the software side let us mention that for Raspberry Pi Inference we have used Raspberry Pi OS (64-bit), Debian 12 (Bookworm) with Python v3.11, llama.cpp: v0.2.20 series (November 2025), Ollama v0.1.20, GCC v12.2, CMake v3.25, and NumPy v1.26, while for Jetson Nano Inference we have used Ubuntu 20.04 LTS (JetPack 4.6.4) with Linux Kernel v4.9, Python v3.8, CUDA Toolkit v10.2, cuDNN v8.2, TensorRT v8.2, llama.cpp: v0.2.20 series (November 2025), Ollama v0.1.20, GCC v9.4, CMake v3.22, and NumPy v1.24.

2.5. Evaluation Methodology

2.5.1. Intrinsic and Downstream Task Evaluation

We measure perplexity (PPL) on four Romanian held-out domains (news, poetry, Wikisource literary prose, and heterogeneous web). To control document-length bias and context carry-over, each domain uses a fixed budget of 200,000 tokens with a sliding window respective of each model’s context capacity and a half stride. For each window we accumulate NLL and report domain PPL as $\exp(-\frac{1}{\sum N} \sum \log P(x_i | x_{<i}))$.

For sentiment analysis, following LaRoSeDa [8], models act as frozen encoders: mean-pooled final-layer states feed (i) a binary polarity classifier and (ii) a four-class rating

classifier over four states, namely $\{1, 2, 4, 5\}$. We use the official given split and report accuracy/macro-F1.

For semantic textual similarity, following RO-STs [11], we compute cosine similarities between sentence embeddings and fit a linear mapping to gold scores (normalized to $[0, 1]$) on the train split. At test time we report Pearson r , Spearman ρ , and MSE.

2.5.2. Edge Inference Benchmarking

Runtime stack and model variants. We evaluate RoBaseLM-S (125 M) and RoBaseLM-M (260 M) in FP16 and 5-bit (Q5_K_M) GGUF using llama.cpp through Ollama, yielding a uniform inference path across devices and stable, comparable metrics (latency, throughput, memory, and where available, energy per token).

Devices. Jetson Nano 4 GB: quad-core ARM Cortex-A57, 128-core Maxwell GPU, unified 4 GB LPDDR4, microSD storage. Raspberry Pi 4 GB: Broadcom BCM2711, quad-core ARM Cortex-A72 (up to 1.5 GHz), VideoCore VI, 4 GB LPDDR4. To ensure parity, we enforce CPU-only inference. Experiments were carried out with the aforementioned SoCs used without enclosures (or additional cooling) at room temperature.

Experimental grid. We vary `num_thread` in 2, 4, `num_ctx` in {512, 1024, 2048, 4096}, and `num_predict` in {32, 64, 128}, with three independent repetitions per grid point ($\text{rep} = 1, 2, 3$). The server is configured with `keep_alive` set to 0 (no KV carry-over) and seed set to 42. For each configuration, we run two warm-up passes at context lengths 512 and 4096.

Deterministic prompt construction. To avoid prefill drift, prompts are synthesized by byte length from a fixed Romanian base paragraph (BASE_PAR) and a constant system string FIXED_SYS = “bench-stable v1; host = <hostname>; note = prompt-bytes-constant”. For each target context length, we map `num_ctx` to `n_chars` and then tile BASE_PAR (with a trailing space) until the total byte length reaches or slightly exceeds `n_chars`, after which we truncate exactly. Token counts are measured directly from the Ollama/llama.cpp JSON output, not inferred. The mapping used is: `num_ctx` = 512 as `n_chars` = 220; `num_ctx` = 1024 as `n_chars` = 520; `num_ctx` = 2048 as `n_chars` = 1200; `num_ctx` = 4096 as `n_chars` = 2600.

Measurement Pipelines (Two Runners per Device)

(A) Thin Bash runner over Ollama (streaming): executes the full grid over $\text{model} \times \text{num_thread} \times \text{num_ctx} \times \text{num_predict}$, CPU-only (OLLAMA_NUM_GPU=0). For each point we stream a request with a fixed seed and deterministic slice, parse the final JSON (prefill vs. decode, token counts), and record TTFT (time-to-first-token). Jetson: sample board telemetry at 1 Hz with `tegrastats`, compute average/max power/temps, record idle baseline, integrate energy; log process RSS/HWM and system free memory, and flag throttling messages. Raspberry Pi: log CPU temperature (`/sys/class/thermal/...` or `vcgencmd`); record RSS/HWM and free memory; omit energy columns. Each grid point is repeated three times, and rows are appended to a single CSV.

(B) Native llama.cpp Python runner (token-level): loads GGUF directly, constructs exact-length token prompts by using a fixed seed string, times prefill (prompt-only) and decode (streamed generation), captures TTFT and per-token inter-arrival gaps, reports overall decode TPS plus robust P50/P95. On Jetson, a 200 ms `tegrastats` thread samples power/temps and integrates energy (with optional idle subtraction); a 4096-context smoke test runs; peak RSS is logged; one wide CSV is written. On Pi, the same timing/memory pipeline runs, with continuous CPU temperature sampling and no energy columns.

Metrics and Computation

For each run, we parse the final JSON output from Ollama and extract the following fields: `prompt_eval_count` (number of prompt tokens processed during prefill), `eval_count` (number of generated tokens), `prompt_eval_duration`, `eval_duration`, and `total_duration`.

From these we compute three main metrics. Prompt throughput (prompt_tps) is the number of prompt tokens processed per second,

$$\text{prompt_tps} = \frac{\text{prompt_tokens}}{\text{prompt_sec}}$$

Generation throughput (gen_tps) is defined analogously for decoding,

$$\text{gen_tps} = \frac{\text{gen_tokens}}{\text{gen_sec}}$$

Time-to-first-token (TTFT) is measured as the wall-clock delay between the given request and receiving the first streamed token,

$$\text{TTFT} = \Delta(t_{\text{first_streamed_line}}, t_{\text{request}})$$

We also log memory, power, and thermals. Process resident set size (VmRSS) and high water mark (VmHWM) are read from `/proc/pid/status`, and available system memory from `/proc/meminfo`. On Jetson Nano, we use `tegrastats` to record power on the main rail (for example `POM_5V_IN`, in mW) and GPU/CPU temperatures. Total energy E is obtained by integrating power over time, $E = \int P(t) dt$, and is reported in joules. Raspberry Pi runs omit energy measurements because we do not have reliable whole-board power instrumentation.

Logged CSV Schema

Each run produces one CSV row. On Jetson Nano we record: timestamp (ts), host, system-on-chip (soc), operating system (os), model identifier (model), repetition index (rep), inference settings (num_thread, num_ctx, num_predict), token counts (prompt_tokens, gen_tokens), timing (prompt_sec, gen_sec, ttft_sec, total_sec), throughput (prompt_tps, gen_tps), process memory (proc_rss_mb, proc_hwm_mb), available system memory (sys_mem_avail_mb), average and maximum power (avg_power_w, max_power_w), idle power (idle_power_w), net power after subtracting idle (net_power_w), total energy (energy_total_j), energy per generated token (energy_per_generated_token_j), energy per all tokens (energy_per_all_tokens_j), average GPU and CPU temperatures (avg_gpu_temp_c, avg_cpu_temp_c), and a throttling flag (throttling).

On Raspberry Pi we log a reduced set: ts, host, soc, os, model, rep, num_thread, num_ctx, num_predict, prompt_tokens, gen_tokens, prompt_sec, gen_sec, ttft_sec, total_sec, prompt_tps, gen_tps, proc_rss_mb, proc_hwm_mb, sys_mem_avail_mb, and cpu_temp_c. Power and energy fields are not recorded on Raspberry Pi.

We also run a Python evaluation script that writes a single wide CSV aggregating all trials. This includes prefill and decode timing, TTFT, overall throughput and P50/P95 throughput, a 4096-token context stress test, peak memory (RSS and HWM), and average and maximum temperature. On Jetson Nano this Python CSV also includes average and maximum power, integrated energy, optional idle-subtracted energy per token, and throttle flags.

Consistency and Reproducibility

All experiments use fixed random seeds. The inference server is run with `keep_alive = 0`, so no key-value cache is reused across requests. Prompts are constructed by deterministic byte-exact slicing for each target context length. We perform two warm-up runs before measurement. Each configuration (num_thread, num_ctx, num_predict) is repeated three times. Each device then writes a single CSV aggregating all runs. We also verify that prompt generation matches the intended byte budget: for example, for num_ctx = 1024 we

generate the prompt (make_prompt 1024), confirm it is exactly 520 bytes, and inspect the first 160 characters to check deterministic tiling and truncation.

Deterministic Prompt Generator

Prompt text is generated by a small helper that enforces byte-level reproducibility. For each target context length ctx , we assign a byte budget n_chars according to a fixed mapping: $ctx = 512$ to $n_chars = 220$; $ctx = 1024$ to $n_chars = 520$; $ctx = 2048$ to $n_chars = 1200$; $ctx = 4096$ to $n_chars = 2600$; any other ctx to $n_chars = 400$. We then repeatedly append the predefined Romanian base paragraph followed by a single space until this byte budget is reached or exceeded, and finally truncate exactly to n_chars . The helper does not insert timestamps or run-specific metadata, so the same context always yields identical prompt text across repetitions and machines.

3. Results and Conclusions

We report results in the same order as our methodology: we first verify training dynamics, then quantify intrinsic language-model quality and frozen-encoder utility, and finally present the edge-deployment measurements obtained under the controlled benchmarking protocol from Section 2.5.2. Throughout the experiment, FP16 and Q5_K_M checkpoints are compared side-by-side to assess the quality/efficiency trade-off enabled by weight-only quantization.

3.1. Learning Dynamics

Figure 1 displays the evolution of the training objective across optimization steps for both parameter scales. The loss descends steeply in the initial regime for the two models, which is consistent with fast curvature exploitation under cosine learning-rate schedules, and then transitions to a smooth, nearly logarithmic decay characteristic of stable decoder-only training at this token budget. The occasional spikes are due to packing boundary conditions and do not alter the long-run trajectory. The 260 M model remains uniformly beneath the 125 M curve after the first few thousand steps, indicating that the additional width and depth, together with SwiGLU and GQA, are converted into better next-token predictions rather than oscillatory behavior.

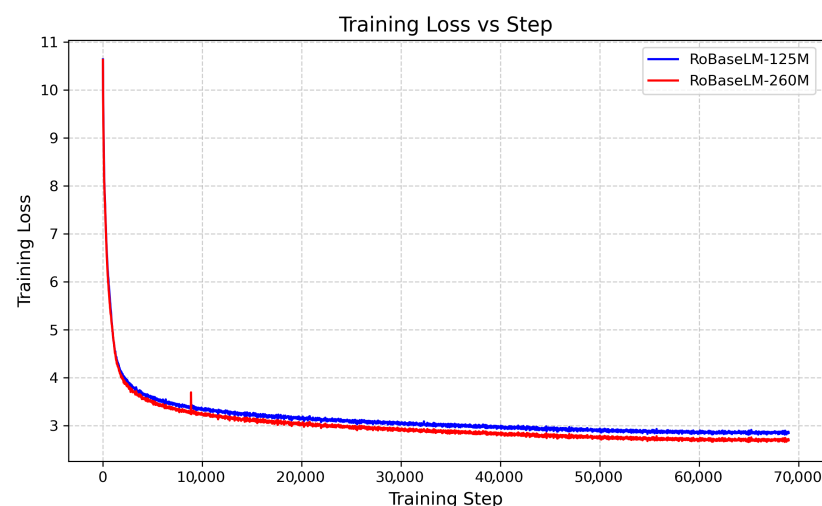


Figure 1. Training loss vs. training step for RoBaseLM-125 M and RoBaseLM-260 M. Curves reflect smoothed step-wise loss with evaluation-induced spikes visible but transient.

3.2. Perplexity Across Domains

Intrinsic language-model quality improves substantially over the Romanian GPT-2 baseline at comparable scale [14]: on contemporary news text, perplexity drops from 30.70

to 15.86 for the 125 M variant, while on heterogeneous web content it falls from 32.48 to 24.05. Some gains are present, though smaller, on poetry and Wikisource, where relative error reductions are approximately 8% and 6% percent, respectively. The 260 M model shows even better results, reducing perplexity over the 125 M by roughly 11% on news, 12% on poetry, 13% on Wikisource, and 17% on web, which agrees with the results reported for causal LMs [18,27]. Across all domains, Q5_K_M tracks closely FP16 (the largest drift we observe is on the order of one percent) which supports the use of low-bit GGUF artifacts in quality-sensitive offline settings without measurable perplexity losses.

3.3. Sentiment Classification (LaRoSeDa) and Semantic Textual Similarity (RO-STs)

In this subsection we present the LaRoSeDa and RO-STs benchmarks that we have selected over other established Romanian benchmarks, such as LiRo or RONEC, due to our objective to evaluate the models beyond basic morphosyntactic or sequence-labeling capabilities. While traditional tasks like Part-of-Speech tagging or Named Entity Recognition are essential for measuring architectural correctness, they often fail to capture the nuanced, high-level semantic manifolds that characterize human language. By focusing on sentiment classification through LaRoSeDa, we test the model’s ability to navigate the complex emotional and pragmatic distributions of the Romanian language. Similarly, RO-STs challenges the model to determine sentence-level similarity, a task that directly reflects how well the language model has internalized the underlying meaning of text rather than just its structural patterns. We find these benchmarks serve as a more rigorous proxy for evaluating how close a compact decoder comes to capturing the functional essence of human-like communication within a strict resource domain.

A single logistic probe on pooled final-layer states yields compact yet reliable sentiment performance. As shown in Table 5, binary accuracy and macro-F1 concentrate near 0.95 for both models, while the four-way rating task stabilizes around 0.83 macro-F1. The 260 M model maintains a small but consistent edge over 125 M in both settings, which is in line with the intuition that moderate scaling sharpens linear separability of sentiment-bearing subspaces and improves calibration without task-specific fine-tuning. Quantized checkpoints preserve decision geometry for shallow heads: Q5_K_M stays within a few tenths of a percentage point of FP16 on every metric, including those dominated by minority classes. The class-skew profile in Table 6 explains the systematic gap between accuracy and macro-F1 and further motivates macro-averaging as the primary criteria under this imbalance regime.

Table 5. LaRoSeDa sentiment with frozen encoders and a single logistic probe.

Model	Binary Acc.	Binary F1	4-Class Acc.	4-Class F1
RoBaseLM-S (125 M)	0.9493	0.9493	0.8270	0.8240
RoBaseLM-S (125 M) (Q5)	0.9533	0.9533	0.8263	0.8231
RoBaseLM-M (260 M)	0.9517	0.9517	0.8300	0.8290
RoBaseLM-M (260 M) (Q5)	0.9513	0.9513	0.8280	0.8267

Table 6. LaRoSeDa class distribution across splits.

Class	Train n	Train %	Test n	Test %	All n	All %
5	4982	41.5	1256	41.9	6238	41.6
1	4459	37.2	1102	36.7	5561	37.1
2	1541	12.8	398	13.3	1939	12.9
4	1018	8.5	244	8.1	1262	8.4
Total	12,000	100	3000	100	15,000	100

Sentence-level similarity recovered with a lightweight regression head over frozen representations attains validation dataset correlations (also referred to as development set or dev-set correlations) in the low 0.74 band for both Pearson and Spearman and test-set correlations around 0.63, with low mean-squared errors on $[0, 1]$ (Table 7). The 260 M model remains modestly but consistently ahead, which suggests the larger network carves a slightly better-structured semantic manifold even in the absence of specialization. The low-bit checkpoints mirror FP16 behavior closely, confirming that rank-based metrics are robust to weight-only compression at five bits when separability margins are not razor-thin.

Table 7. RO-STS with frozen encoders. Correlations are $\times 100$; MSE on $[0, 1]$.

Model	Spearman Dev	Spearman Test	Pearson Dev	Pearson Test	MSE Dev	MSE Test
RoBaseLM-S (125 M)	73.2	61.5	73.6	63.4	0.042	0.056
RoBaseLM-S (125 M) (Q5)	72.9	60.2	73.3	62.4	0.042	0.057
RoBaseLM-M (260 M)	73.9	61.6	74.1	63.5	0.042	0.058
RoBaseLM-M (260 M) (Q5)	72.1	61.8	72.3	63.4	0.045	0.059

On the subject of linear probes, let us add that for both LaRoSeda and RO-STS we ensured experimental consistency by utilizing fixed random seeds across all trials to maintain reproducibility. While the primary metrics reported in Tables 5 and 7 represent the performance of a single logistic probe and a lightweight regression head, respectively, all experimental configurations were subject to three independent repetitions to monitor for variance. We found the results were remarkably stable and showed minimal sensitivity to initialization or training variance; the performance for sentiment classification concentrated consistently near 0.95 binary accuracy and sentence similarity correlations remained steady in the low 0.74 band for development sets. This high level of stability suggests the frozen representations learned by RoBaseLM-S and RoBaseLM-M are robust and the semantic manifolds they define are sufficiently structured for linear readout without being overly dependent on the specific stochasticity of the probe's training phase.

3.4. Edge Runtime Results

We now turn to the edge deployments on Raspberry Pi and Jetson Nano SoCs used in their bare out-of-the-box versions. Naturally, the addition of dedicated enclosures and/or additional cooling will enhance the outputs of both models, but our goal is to chart the intrinsic computational capabilities of both devices.

Figures 2 and 3 summarize decode throughput in tokens per second as a function of context length for two- and four-thread runs, while Figures 4 and 5 report the complementary latency view via time to first token. The four panels together depict the overall computational framework: decode speed degrades with longer contexts because each generated token must attend over a larger KV cache, which increases per-step memory traffic and arithmetic. The slope is steeper for the larger model due to wider projections and fatter KV states but remains monotone and smooth, which is indicative of cache-coherent access. Quantization improves throughput across the board by reducing weight bandwidth and strengthening cache locality, with the absolute gains most visible in larger contexts where memory pressure dominates. The effect is more pronounced on Jetson Nano than on Raspberry Pi, a difference that is coherent with the two devices' microarchitectural balance and memory subsystems.

On Raspberry Pi, the two-thread configuration yields respectable interactive rates at short and medium contexts for the 125 M model, and it remains usable at 4 k with a moderate drop in tokens per second. The 260 M variant is slower, but maintains a smooth

decline and benefits visibly from quantization, which suggests the device is primarily memory-bound in this regime. Moving from two to four threads produces modest additional speed, which indicates the Cortex-A72 cores saturate the relevant memory channels early and scheduling overhead eats part of the theoretical parallelism. The shape of the curves and their stratification by model size and precision align with a bandwidth-limited decode phase whose per-token cost scales roughly linearly in context length.

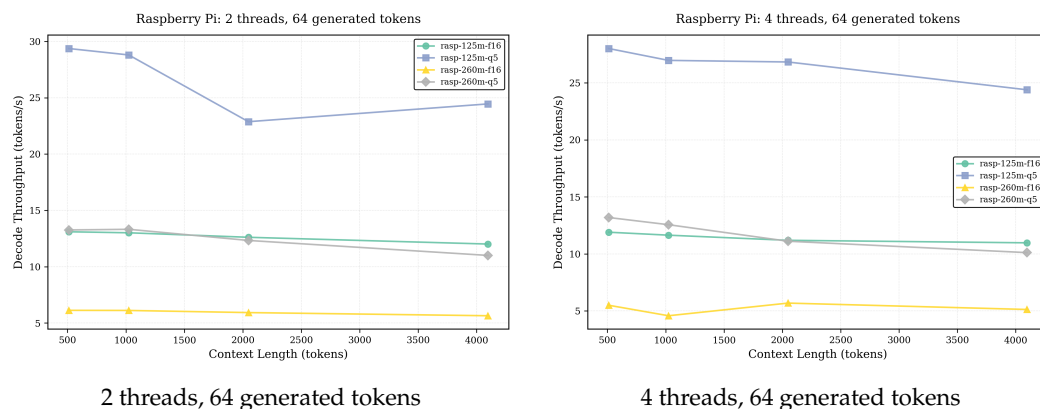


Figure 2. Raspberry Pi: decode throughput (tokens/s) vs. context length for two- and four-thread runs. Curves show predictable throughput decay with context due to growing KV-cache footprint, with quantized checkpoints consistently faster at fixed threads.

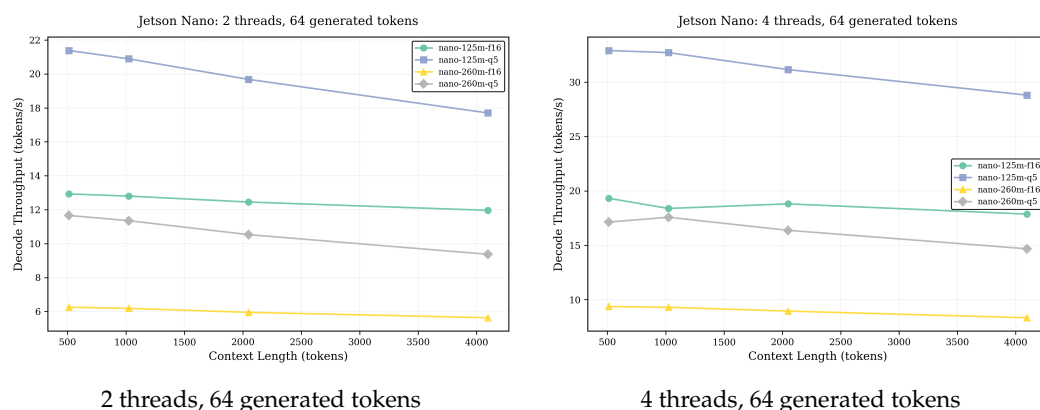


Figure 3. Jetson Nano: decode throughput (tokens/s) vs. context length for two- and four-thread runs. The Nano exhibits higher absolute throughput and larger multi-thread gains than Raspberry Pi, particularly for quantized checkpoints.

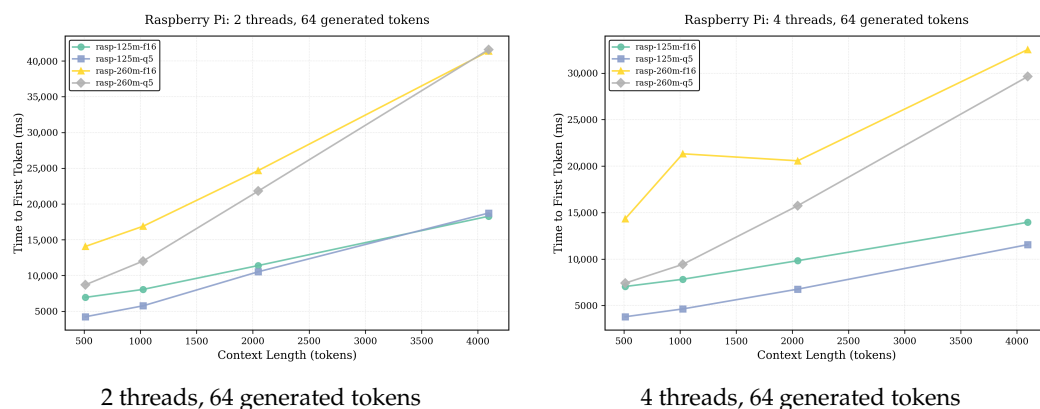


Figure 4. Raspberry Pi: time to first token (TTFT) vs. context length for two- and four-thread runs. TTFT grows with prompt length as expected for prefill-dominated latency. Quantization reduces TTFT more strongly at longer contexts.

The TTFT profiles on Raspberry Pi rise with context length in a near-linear fashion, reflecting the fact that prefill latency is proportional to the number of prompt tokens and is sensitive to attention’s quadratic scaling in the prompt phase. The four-thread configuration lowers TTFT relative to two threads but the gap narrows as context grows, which again signals a transition from compute-limited to memory-limited behavior as the key–value cache expands. The quantized checkpoints allow for additional millisecond gains across the entire range and their benefit becomes increasingly visible at 2 k–4 k contexts.

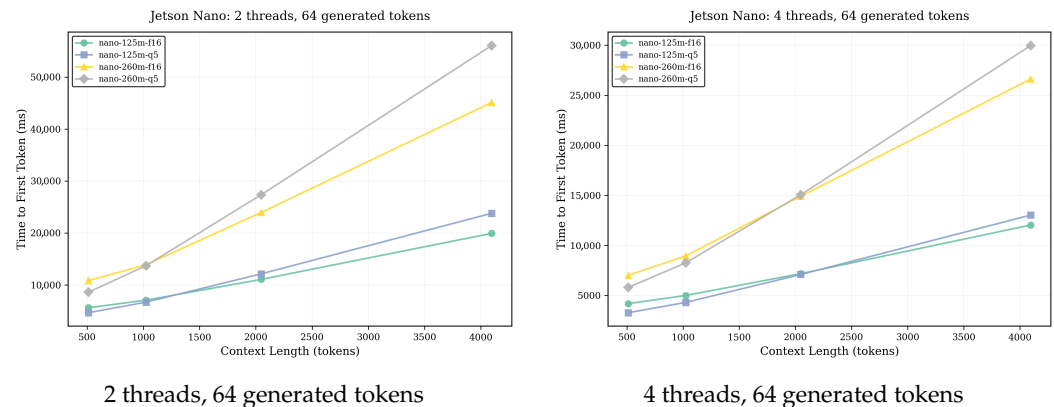


Figure 5. Jetson Nano: time to first token (TTFT) vs. context length for two- and four-thread runs. Prefill latency scales with prompt length; the four-thread and quantized settings reduce TTFT most strongly at 2 k–4 k contexts.

On Jetson Nano, decode throughput is uniformly higher than on Raspberry Pi across all contexts and thread counts, with the largest margins observed for the quantized 125 M model at small to medium contexts. The transition from two to four threads delivers a clearer benefit on this platform, which is consistent with better CPU scheduling and intrinsic thermal headroom on Cortex-A57 for these workloads. The curves remain ordered by model size and precision in the anticipated fashion, and the decline with context mirrors the Raspberry Pi pattern while starting from a higher baseline. The quantized 260 M model approaches the FP16 125 M in several settings, which suggests that low-bit compression narrows the gap between scales when memory bandwidth is the limiting factor.

The Nano’s TTFT curves confirm the decode picture. Prefill time scales with context and benefits from parallelism and quantization, with the four-thread configuration and the low-bit checkpoints jointly pushing TTFT into the interactive regime even at 2 k–4 k contexts for the 125 M model. The slope remains gentle enough to support long-context prompting without incurring prohibitive initial latency, and the stability of the lines across repetitions indicates that our warm-up and determinization controls are effective. Qualitative experiments on recent iPhone devices using the same GGUF artifacts suggest that memory footprints remain a small fraction of available RAM and that perceived responsiveness is on par with the Jetson configuration, although the platform’s opaque telemetry prevents the construction of quantitative energy and throughput profiles comparable to those reported here.

3.5. Discussion

The results in the previous sections show that RoBaseLM decoders at 125 M and 260 M parameters deliver substantive perplexity reductions on contemporary Romanian prose, preserve robust linear readout for polarity and ordinal ratings, and sustain frozen-encoder performance on RO-STs at levels expected for general-purpose representations. Weight-only 5-bit quantization leaves these trends essentially unchanged. This consistency across

evaluations indicates that the core design decisions, namely monolingual tokenizer, data curation, and decoder architecture, allow for the desired efficiency.

Frozen encoder analyses on LaRoSeDa show that a single logistic probe recovers sentiment structure with little effort: binary F1 concentrates near 0.95 and four-class macro-F1 around 0.83, these values being in agreement with other results reported in the literature [28,29]. The 260 M model delivers compact but reliable gains over 125 M, a result that is consistent with previous findings on the impact of dimensionality on the effectiveness of language models [30]. Macro-F1 is marginally below accuracy due to the class-frequency profile of LaRoSeDa. It remains the metric of choice for model selection because it better reflects the allocation of probability mass across minority labels, an aspect that governs error costs in moderation and feedback triage pipelines [31].

On RO-STs, frozen encoders with a lightweight regressor reach development correlations in the low 0.74 range and test correlations around 0.62–0.63. This profile is consistent with reports for unspecialized sentence encoders on English STS tasks and with the expectation that contrastive or Siamese adaptation is required to lift rank-based metrics further [32–35]. Embedding anisotropy compresses cosine dynamic range in frozen models, which dampens rank correlations unless otherwise corrected [36]. The consistent edge of 260 M over 125 M suggests that incremental capacity reshapes the semantic manifold sufficiently for linear regression to benefit even without adaptation.

Compact, quantized language models can potentially overcome the barrier of running inference directly on handheld devices, enabling an automated messaging service that reads, prioritizes, and drafts SMS or email responses locally. In such a design, the model parses an incoming message, infers intent and key entities, proposes a candidate reply consistent with a user profile or policy, and either sends it autonomously under constraints (e.g., “auto-acknowledge meeting invites” within defined hours) or routes it for user approval. Because computation remains on the device, latency is bounded by local inference and user I/O, and sensitive text never leaves the handset, aligning with privacy and availability requirements identified in recent work on on-device LLMs and edge inference [23,24,26].

Handheld platforms (smartphones, tablets, wearables) present strict resource envelopes: limited DRAM, thermal design power, and sporadic CPU/GPU/NPU availability under mobile OS schedulers. Directly deploying a multi-billion parameter FP16 model is untenable under such constraints, but our preliminary results on iPhone devices (not reported here because of opaque telemetry) have shown that our models run effectively without adding a considerable computational load. There is an on-going discussion regarding the two levers that dominate the feasibility of relatively large language models on handheld platforms, namely (i) model size/effective precision (parameters, activation footprint, KV-cache), and (ii) memory bandwidth per token at target context lengths, with the observation that weight-only post-training quantization to 8-, 5-, or 4-bit substantially cuts model footprint and memory traffic while preserving task quality [23,24,26], but our results suggest that models of only a few hundred million parameters can be easily deployed. The direct implication of our findings is that small-language models of natural-language-fluency token rates can run covertly on low-power devices due to their low computational load in inference mode. This in turn suggests that new security provisions should be considered for handheld platforms, as the possibility of having covert language models operating on them is not merely theoretical.

4. Conclusions and Outlook

We have reported the results of our cutting-edge investigations into the construction of language models developed for Romanian, showcasing RoBaseLM-S (125 M) and

RoBaseLM-M (260 M), two compact Romanian decoder-only language models that successfully bridge the gap between high-quality linguistic performance and resource-constrained edge deployment. By training from scratch on a curated 4.3 B-token corpus and utilizing a modern architecture, including 4 k-token context windows, RMSNorm, and grouped-query attention, we have demonstrated that small-scale decoders can achieve significant intrinsic quality gains, such as reducing news-domain perplexity from 30.7 to 15.9. Our results confirm that 5-bit post-training quantization (Q5_K_M) preserves the integrity of the semantic manifold, allowing these models to maintain high utility on tasks such as LaRoSeDa sentiment classification and RO-STS sentence similarity while maintaining a footprint under 500 MB. Ultimately, this work establishes a new frontier for the Romanian NLP ecosystem by providing the first publicly available, quantization-ready models capable of real-time, fully offline inference on consumer-grade edge hardware like the Jetson Nano and Raspberry Pi. The deployment of these edge-ready models suggests that local, privacy-preserving AI on handheld devices is not only feasible but can be achieved with a minimal computational footprint without sacrificing the statistical richness required for meaningful natural language generation.

On the side of future developments, let us note that two types of methods can further improve the viability of edge deployments, namely: (a) distillation/architectural compaction: transferring behavior from a larger teacher into a slim student (sub-1 B to a few B parameters) with task-aligned objectives; (b) parameter-efficient tuning (adapters/LoRA) that avoids growing the served model while enabling personalization under storage limits. Empirically, compact decoders in the 0.1–3 B range (trained with subword tokenization tuned to the target language [5,6]) strike an attractive accuracy-latency-energy frontier for handhelds, particularly when paired with 4–5-bit weights [23,24,26].

The same properties that make on-device language models attractive (offline operation, no external telemetry, data locality) could be abused by malicious software. A covert app could embed a quantized SLM to parse notifications, synthesize contextually appropriate replies, and exfiltrate conversation dynamics without invoking remote inference. Although published incidents to date have largely focused on the use of remote language models APIs in attack chains rather than fully local models, security analyses caution that endpoint-resident models shift the attack surface squarely onto the device and its app-level permissions [37,38]. Mitigation scenarios should include: least privilege for notification access and messaging APIs, runtime permission prompts for send actions, background execution limits, and on-device anomaly detection that monitors unusual volumes or linguistic patterns in automated replies. From a platform perspective, provenance controls for bundled model artifacts and code-signing for model loaders reduce the risk of stealthy AI inference within otherwise benign-looking apps.

Author Contributions: Conceptualization, T.A.D. and A.N.-Ț.; methodology, T.A.D. and A.N.-Ț.; software, T.A.D. and P.F.d.V.; validation, T.A.D., P.F.d.V., A.O., A.F.N. and M.C.R.; formal analysis, T.A.D. and A.N.-Ț.; investigation, T.A.D., P.F.d.V. and A.F.N.; resources, A.O., M.C.R. and A.N.-Ț.; data curation, T.A.D.; writing—original draft preparation, T.A.D.; writing—review and editing, T.A.D., A.F.N., A.O., M.C.R. and A.N.-Ț.; visualization, T.A.D., A.F.N. and A.N.-Ț.; supervision, A.N.-Ț.; project administration, A.N.-Ț. All authors have read and agreed to the published version of the manuscript.

Funding: For this work M.C.R. and A.N.-Ț were supported by the Romanian Ministry of Education and Research under Romanian National Core Program LAPLAS VII—contract no. 30N/2023, while A. Oprea was funded through the MA'AT Project, co-funded by European Union (Code SMIS 2021+: 330941).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is available from the authors upon request. Codes are freely available on GitHub.

Acknowledgments: We thank the Institute of Space Science—INFLPR Subsidiary for providing the equipment used in training and evaluation. The authors would like to acknowledge fruitful discussions with Daniela Cojoacă and Andrada Mihailov.

Conflicts of Interest: The authors declare no conflicts of interest. The founders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Diac, T.A.; Neagoe, A.F.; Raportaru, M.C.; Oprea, A.; Drăgan, R.-M.; Nicolin-Źaczek, A. Statistics of a large-scale Romanian corpus for language modelling. *Rom. J. Phys.* **2025**, *70*, 111. [[CrossRef](#)]
2. Ortiz Suárez, P.J.; Romary, L.; Sagot, B. A Monolingual Approach to Contextualized Word Embeddings for Mid-Resource Languages. In Proceedings of the ACL 2020, Seattle, WA, USA, 5–10 July 2020; pp. 1703–1714.
3. Conneau, A.; Khandelwal, K.; Goyal, N.; Chaudhary, V.; Wenzek, G.; Guzmán, F.; Grave, E.; Ott, M.; Zettlemoyer, L.; Stoyanov, V. Unsupervised Cross-Lingual Representation Learning at Scale. In Proceedings of the ACL 2020, Seattle, WA, USA, 5–10 July 2020; pp. 8440–8451.
4. CC-100. Monolingual Datasets from Web Crawl Data. Data Resource. 2020. Available online: <https://live.european-language-grid.eu/catalogue/corpus/11703> (accessed on 13 January 2026).
5. Sennrich, R.; Haddow, B.; Birch, A. Neural Machine Translation of Rare Words with Subword Units. In Proceedings of the ACL 2016, Berlin, Germany, 7–12 August 2016; pp. 1715–1725.
6. Kudo, T.; Richardson, J. SentencePiece: A Simple and Language-Independent Subword Tokenizer and Detokenizer. In Proceedings of the EMNLP 2018 (System Demonstrations), Brussels, Belgium, 31 October–4 November 2018; pp. 66–71.
7. Dumitrescu, S.D.; Rebeja, P.; Lorincz, B.; Gaman, M.; Avram, A.; Ilie, M.; Pruteanu, A.; Stan, A.; Rosia, L.; Iacobescu, C.; et al. LiRo: Benchmark and Leaderboard for Romanian Language Tasks. In Proceedings of the NeurIPS 2021 Datasets and Benchmarks, New Orleans, LA, USA, 6–14 December 2021.
8. Tache, A.; Găman, M.; Ionescu, R.T. LaRoSeDa: A Large Romanian Sentiment Data Set. In Proceedings of the EACL 2021, Online, 19–23 April 2021; pp. 949–956. Available online: <https://aclanthology.org/2021.eacl-main.81.pdf> (accessed on 13 January 2026).
9. Dumitrescu, S.D.; Avram, A.-M. Introducing RONEC—The Romanian Named Entity Corpus. In Proceedings of the LREC 2020, Marseille, France, 11–16 May 2020; pp. 4436–4443.
10. RONEC v2.0. Romanian Named Entity Corpus (Version 2.0). Data Resource. 2022. Available online: <https://github.com/dumitrescustefan/ronec> (accessed on 13 January 2026).

11. Dumitrescu, Ș.D.; Avram, A.-M. RO-STs: Romanian Semantic Textual Similarity. Dataset Card. 2021. Available online: <https://github.com/dumitrescustefan/RO-STs> (accessed on 13 January 2026).
12. Masala, M.; Rușeti, Ș.; Dascălu, M. RoBERT—A Romanian BERT Model. In Proceedings of the COLING 2020, Online, 8–13 December 2020; pp. 6626–6637.
13. Tăiatu, I.-M.; Avram, A.-M.; Cercel, D.-C.; Pop, F. RoBERTweet: A BERT Language Model for Romanian Tweets. In Proceedings of the NLDB 2023, LNCS 13913, Madrid, Spain, 21–23 June 2023; Springer: Berlin/Heidelberg, Germany, 2023; pp. 577–587.
14. Niculescu, M.A.; Rușeti, Ș.; Dascălu, M. RoGPT-2: Romanian GPT-2 for Text Generation. In Proceedings of the ICTAI 2021, Virtual, 1–3 November 2021; IEEE: New York, NY, USA, 2021; pp. 1154–1161.
15. Masala, M.; Ilie-Ablachim, D.C.; Corlatescu, D.; Zavelca, M.; Leordeanu, M.; Velicu, H.; Popescu, M.; Dascalu, M.; Rebedea, T. OpenLLM-Ro: Technical Report on Open-Source Romanian LLMs. *arXiv* **2024**, arXiv:2405.07703.
16. The Models Are Freely Available. Available online: <https://github.com/tudord14/edge-RoLM> (accessed on 13 January 2026).
17. Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; Liu, P.J. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *JMLR* **2020**, *21*, 1–67. Available online: <https://jmlr.org/papers/volume21/20-074/20-074.pdf> (accessed on 13 January 2026).
18. Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; et al. LLaMA: Open and Efficient Foundation Language Models. *arXiv* **2023**, arXiv:2302.13971. [CrossRef]
19. Su, J.; Ahmed, M.; Lu, Y.; Pan, S.; Bo, W.; Liu, Y. RoFormer: Enhanced Transformer with Rotary Position Embedding. *NeurIPS* **2021**, *34*, 22614–22627. [CrossRef]
20. Shazeer, N. GLU Variants Improve Transformer. *arXiv* **2020**, arXiv:2002.05202. [CrossRef]
21. Ainslie, J.; Lee-Thorp, J.; Dehghani, M.; Ontañón, S.; Tay, Y. GQA: Training Generalized Multi-Query Transformer Models. In Proceedings of the EMNLP 2023, Singapore, 6–10 December 2023; pp. 5402–5419.
22. Hoffmann, J.; Borgeaud, S.; Mensch, A.; Buchatskaya, E.; Cai, T.; Rutherford, E.; Casas, D.D.; Hendricks, L.A.; Welbl, J.; Clark, A.; et al. Training Compute-Optimal Large Language Models. *arXiv* **2022**, arXiv:2203.15556. [CrossRef]
23. Frantar, E.; Ashkboos, S.; Stock, P.; Alistarh, D. GPTQ: Accurate Post-Training Quantization for Generative Pretrained Transformers. In Proceedings of the ICML 2023, Honolulu, HI, USA, 23–29 July 2023; pp. 11745–11759.
24. Lin, J.; Tang, J.; Tang, H.; Yang, S.; Chen, W.M.; Wang, W.C.; Xiao, G.; Dang, X.; Gan, C.; Han, S. AWQ: Activation-Aware Weight Quantization for LLM Compression and Acceleration. *arXiv* **2023**, arXiv:2306.00978. [CrossRef]
25. Gerganov, G. llama.cpp: Port of LLaMA to C/C++. Software/Technical Report. 2023. Available online: <https://github.com/ggml-org/llama.cpp/tree/master/examples/embedding> (accessed on 13 January 2026).
26. Dettmers, T.; Lewis, M.; Belkada, Y.; Zettlemoyer, L. LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale. In Proceedings of the 36th Conference on Neural Information Processing Systems (NeurIPS 2022), New Orleans, LA, USA, 28 November–9 December 2022. Available online: https://proceedings.neurips.cc/paper_files/paper/2022/file/c3ba4962c05c49636d4c6206a97e9c8a-Paper-Conference.pdf (accessed on 13 January 2026).
27. Kaplan, J.; McCandlish, S.; Henighan, T.; Brown, T.B.; Chess, B.; Child, R.; Gray, S.; Radford, A.; Wu, J.; Amodei, D. Scaling Laws for Neural Language Models. *arXiv* **2020**, arXiv:2001.08361. [CrossRef]
28. Alain, G.; Bengio, Y. Understanding Intermediate Layers Using Linear Classifier Probes. In Proceedings of the ICLR 2017 Workshop, Toulon, France, 24–26 April 2017.
29. Peters, M.E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; Zettlemoyer, L. Deep Contextualized Word Representations. In Proceedings of the NAACL 2018, New Orleans, LA, USA, 1–6 June 2018; pp. 2227–2237.
30. Aghajanyan, A.; Gupta, A.; Zettlemoyer, L. Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning. In Proceedings of the ACL 2021, Online, 1–6 August 2021; pp. 7319–7328. Available online: <https://aclanthology.org/2021.acl-long.568.pdf> (accessed on 13 January 2026).
31. He, H.; Garcia, E.A. Learning from Imbalanced Data. *IEEE Trans. Knowl. Data Eng.* **2009**, *21*, 1263–1284. [CrossRef]
32. Cer, D.; Diab, M.; Agirre, E.; Lopez-Gazpio, I.; Specia, L. SemEval-2017 Task 1: Semantic Textual Similarity—Multilingual and Cross-lingual Focus. In Proceedings of the SemEval 2017, Vancouver, BC, Canada, 3–4 August 2017.
33. Agirre, E.; Cer, D.; Diab, M.; Gonzalez-Agirre, A. SemEval-2012 Task 6: A Pilot on Semantic Textual Similarity. In Proceedings of the SemEval 2012, Montréal, QC, Canada, 7–8 June 2012.
34. Reimers, N.; Gurevych, I. Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks. In Proceedings of the EMNLP-IJCNLP 2019, Hong Kong, China, 3–7 November 2019; pp. 3982–3992.
35. Gao, T.; Yao, X.; Chen, D. SimCSE: Simple Contrastive Learning of Sentence Embeddings. In Proceedings of the EMNLP 2021, Punta Cana, Dominican Republic, 7–11 November 2021; pp. 6894–6910.
36. Ethayarajh, K. How Contextual Are Contextualized Word Representations? In Proceedings of the EMNLP-IJCNLP 2019, Hong Kong, China, 3–7 November 2019; pp. 55–65.

37. Yao, Y.; Duan, J.; Xu, K.; Cai, Y.; Sun, Z.; Zhang, Y. A Survey on Large Language Model Security and Privacy. *High-Confid. Comput.* **2024**, *4*, 100211. [[CrossRef](#)]
38. Das, S.; Sen, S. Security and Privacy Challenges of Large Language Models: A Survey. *arXiv* **2024**, arXiv:2402.00888. [[CrossRef](#)]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.