Benchmarking SmolLM-135M: Energy Optimization and Performance Analysis

Julien Delavande

20/12/2024

1. Introduction

As large language models become more prevalent, balancing performance with energy efficiency is crucial. This report benchmarks the SmolLM-135M model [5] on a subset of the SQuAD dataset [6], focusing on energy consumption, CO_2 emissions, and inference time per token. Beyond basic hardware and backend comparisons, we evaluate various quantization strategies. The emphasis here is as much on the benchmarking methodology and analysis as on the optimizations themselves.

All experiments are fully reproducible. The exact script and instructions can be found at: SmoLM energy optimisations analysis.

2. Experimental Methodology

Hardware and Software Setup

We conducted our experiments on Google Colab instances located in South Carolina, USA, using:

- **CPU:** Intel Xeon (2 vCPUs, 2.20GHz).
- **GPU:** NVIDIA T4 (16 GB VRAM) running under CUDA-compatible drivers.
- **Memory:** 12.675 GB RAM evaluated as 4.75W power.

Software stack:

- **Model:** SmolLM-135M [5] from Hugging Face Hub.
- **Frameworks:** Hugging Face Transformers [3], ONNX Runtime [4], and Optimum [2].
- **Monitoring:** CodeCarbon [1] for energy and emissions tracking.

For exact versions (Transformers, CUDA, ONNX Runtime), see the linked repository.

Dataset and Benchmarking Procedure

We selected 50 samples from the SQuAD test split for inference. While this is a relatively small number, it allows for rapid iteration and demonstration of the methodology. For more robust conclusions, a larger set or multiple subsets should be tested, and experiments should be repeated multiple times (e.g., three independent runs) to report means and standard deviations.

Each run follows a three-step pipeline:

1. **Data Loading:** Load 50 samples from the SQuAD test set.
2. **Model Inference:** Run each sample through the specified model and backend configuration. All parameters, such as batch size and sequence length, remain consistent across runs.
3. **Energy and Emissions Tracking:** Use CodeCarbon to measure CPU/GPU/RAM energy draw, averaging over multiple measurements. CO_2 emissions are computed based on the local energy mix of South Carolina, USA.

Energy and Carbon Intensity Tracking

CodeCarbon monitors power usage:

- **Carbon Intensity:** Derived from local grid energy sources and mixed to estimate gCO_2/kWh.
- **Energy Measurement:** CPU energy is read from RAPL interfaces (Linux) or Intel Power Gadget. GPU power is monitored via NVIDIA's `pynvml`. RAM usage is estimated at a fixed rate (3 W per 8 GB).

The sampling interval is every 15 seconds, which introduces some averaging. In shorter runs, this may reduce the precision of per-token measurements. Future work may involve synchronizing measurements at finer-grained intervals.

Optimizations Evaluated

We compared different combinations of hardware (CPU vs. GPU), backends, and quantization strategies:

- **Hugging Face (HF) Default Backend:** Standard model inference pipeline.
- **ONNX Backend:** Uses an optimized graph format for potential CPU acceleration and dynamic quantization.
- **Quantization Approaches:**
  - **No Quantization:** Baseline full-precision model.
  - **Static 8-bit and 16-bit Quantization:** Reduces model weights to lower precision at load time, potentially decreasing memory usage and energy. However, this may not always yield speed-ups and can sometimes introduce overheads.
  - **Dynamic 16-bit Quantization:** Applies quantization during inference, potentially balancing performance and efficiency. This can be done without model retraining.

3. Results and Observations

3.1 Summary of Results

Hardware	**Backend**	**Optimization**	**Energy (Wh)**	**Time (s/token)**	**CO_2 (kg)**
CPU	HF Default	None	1.33e-3	1.01e-1	4.64e-7
GPU	HF Default	None	6.94e-4	**3.26e-2**	2.42e-7
GPU	HF Default	8-bit	2.19e-3	1.01e-1	7.65e-7
GPU	HF Default	16-bit	1.03e-3	4.79e-2	3.59e-7
CPU	ONNX	None	7.56e-4	5.77e-2	2.64e-7
CPU	ONNX	16-bit	**4.63e-4**	3.54e-2	**1.62e-7**

Table 1: Summary of benchmarking results.

3.2 Analysis of the Results

Comparing inference with the default backend on CPU and GPU immediately shows that the GPU configuration is both more energy-efficient and faster per token. With the default backend, the GPU uses less energy per token and produces fewer emissions, while achieving lower inference times than the CPU in a non-quantized setting.

However, introducing static quantization (8-bit) under the default backend on GPU did not improve the situation. On the contrary, it increased energy consumption per token and slowed down inference, indicating that this form of quantization is not optimal. While static 16-bit quantization performed better than 8-bit, it still failed to match the non-quantized GPU's balance of energy efficiency and speed.

The ONNX backend on CPU, by contrast, achieves performance closer to that of the GPU. Moreover, when applying dynamic 16-bit quantization with ONNX, energy emissions per token improved even further, suggesting that careful selection of backend and quantization approach can bring CPU-based inference in line with GPU-level efficiency and speed.

4. Critique

While these results provide useful insights, several limitations warrant caution. The analysis was conducted on a small subset of samples, limiting representativeness. The experients were only run once, future works could focus on performing multiple runs (e.g., $n = 5$) and report average and standard deviation to increase confidence in the results. Only inference metrics were examined, ignoring overheads such as model conversion or calibration steps. Furthermore, quantization strategies, backend optimizations, and hardware variations were only partially explored, leaving potential gains or drawbacks unaddressed. Differences in carbon intensity or regional energy sources may also influence emissions. Future work should use larger datasets, consider a broader range of techniques, and include the full lifecycle of model deployment to offer a more reliable and holistic evaluation.

5. Conclusion

This study shows that hardware, backend selection, and quantization strategies significantly influence both performance and energy efficiency for the SmolLM-135M model. While the GPU with the default backend is generally more efficient and faster than the CPU, naive static quantization—especially at 8-bit—can negate these benefits. In contrast, using ONNX on CPU can achieve performance close to GPU levels, and applying dynamic 16-bit quantization further reduces energy usage and emissions. Ultimately, the optimal balance depends on specific project priorities, be it speed, cost, or sustainability.

References

[1] Codecarbon. `https://github.com/mlco2/codecarbon`. Accessed: 2024-12-20.

[2] Hugging face optimum. `https://huggingface.co/docs/optimum/index`. Accessed: 2024-12-20.

[3] Hugging face transformers. `https://huggingface.co`. Accessed: 2024-12-20.

[4] Onnx runtime. `https://onnxruntime.ai`. Accessed: 2024-12-20.

[5] Smollm-135m. `https://huggingface.co/HuggingFaceTB/SmolLM-135M`. Accessed: 2024-12-20.

[6] Squad dataset. `https://huggingface.co/datasets/rajpurkar/squad`. Accessed: 2024-12-20.