Self Project 2025-08-22

Vehicle-Scale LLMs: Low-Rank Residuals + 4-Bit Quantization for In-Vehicle AI

A memory-efficient in-vehicle LLM inference pipeline combining INT4 quantization with low-rank residual compensation for constrained edge hardware.

No cover image available.

Project Snapshot

DATE
2025-08-22

CATEGORY
Self Project

SHORT DESCRIPTION
A memory-efficient in-vehicle LLM inference pipeline combining INT4 quantization with low-rank residual compensation for constrained edge hardware.

TAGS / TECH STACK
LLM Inference · Quantization · Low-Rank Adaptation · Edge AI · Automotive AI

Project Links

GitHub Repository

Full Project Content

> A compact starter for memory-efficient, vehicle-oriented LLM inference using **INT4 quantization** and **low-rank adapter compensation**.

GitHub Repository: [dranubhaparashar/Vehicle-Scale-LLMs-Integrating-Low-Rank-Residuals-and-4-Bit-Quantization-for-In-Vehicle-AI]

This post documents a compact, reproducible starter for **vehicle-scale LLM inference**. The repository presents **DAC+Q4-ITS** as a minimal pipeline that demonstrates a complete flow from corpus preparation to export, INT4 quantization, eigenspace computation, adapter injection, inference, and evaluation.

Vision

Large language models can be useful in in-vehicle AI systems, but deployment is difficult when memory and compute budgets are tight. This project explores a practical approach: combine **4-bit quantization** with **low-rank residual compensation** so the model remains lightweight while recovering part of the quality lost through aggressive compression.

Why This Matters

> **Note** The repository is explicitly framed as a **minimal reproducible starter** for an **INT4 + low-rank adapter compensation pipeline**.

> **Important** The README describes an end-to-end flow: **corpus → export → quantize (INT4) → eigenspace (SVD) → build adapters (U, D) → inject → inference → evaluation**.

> **Tip** The starter is designed to prove the flow on a toy transformer first, then be swapped with a LLaMA-derived model later.

> **Warning** The current public repo is a starter implementation, not a full production deployment kit.

> **Caution** Claims about real-world automotive performance should be validated with a real model, real deployment stack, and benchmark data.

What This Starter Does

```
Corpus Preparation
        ↓
ONNX Export
        ↓
INT4 Quantization
        ↓
Eigenspace Computation (SVD)
        ↓
Build Low-Rank Adapters
        ↓
Inject Adapters
        ↓
Run Inference
        ↓
Evaluate Output
```

The repository README describes this exact flow and positions it as a proof-of-pipeline for memory-efficient inference.

Project Attributes

Attribute	Description
`problem-statement`	Full-precision LLM inference is too heavy for constrained in-vehicle environments.
`primary-objective`	Demonstrate a memory-efficient inference pipeline using INT4 quantization plus low-rank residual compensation.
`core-technologies`	PyTorch, ONNX export, INT4 quantization, SVD/eigenspace methods, low-rank adapters.
`target-setting`	In-vehicle AI and other edge-like environments where memory and compute are limited.
`key-mechanism`	Compressed forward pass described by the repo as `dequantized(INT4 matmul) + U(Dx)`.

Attribute	Description
`current-scope`	A minimal reproducible starter built around a toy transformer.
`future-direction`	Swap the toy model with a LLaMA-derived model and wire TensorRT for Jetson when ready.

Repository Structure

The repository currently includes dedicated folders for configs, data, Docker, docs, environment setup, scripts, templates, tests, and source code under `src/dac_q4_its`.

```
Vehicle-Scale-LLMs-Integrating-Low-Rank-Residuals-and-4-Bit-Quantization-for-In-Vehicle-AI/
├── configs/
├── data/
├── docker/
├── docs/
├── env/
├── guidelines/
├── scripts/
├── src/
│   └── dac_q4_its/
├── templates/
├── tests/
├── Makefile
├── README.md
└── pyproject.toml
```

Quickstart

The README provides this starter flow:

```
python -m venv .venv && source .venv/bin/activate
pip install -r env/requirements.txt
python scripts/01_prepare_calibration.py
python scripts/02_export_onnx.py
python scripts/03_quantize_int4.py
python scripts/04_compute_eigenspaces.py
python scripts/05_inject_adapters.py
python scripts/07_run_inference.py --prompt "avoid tolls and reach airport by 6pm"
python scripts/08_eval_nln.py
```

This sequence shows the intended end-to-end execution path from calibration preparation to inference and evaluation.

What the Starter Gives

According to the README, the starter includes:

- Working INT4 quantization with per-row scales
- Rank-8 adapters computed from calibration activations
- A compressed forward pass based on dequantized INT4 matmul plus low-rank residual correction
- Simple EM / SOFT-F1 metrics demo

Core Compression Idea

```
def compressed_forward(x, w_int4, scales, U, D):
    base = dequantized_int4_matmul(x, w_int4, scales)
    correction = U @ (D @ x)
    return base + correction
```

The exact README wording summarizes the compressed forward path as:

```
dequantized(INT4 matmul) + U(Dx)
```

That is the central systems idea of this starter.

Why This Design Is Interesting

- It targets memory efficiency directly through **INT4 quantization**
- It adds a **low-rank correction path** instead of fully undoing compression
- It keeps the pipeline modular, with separate scripts for export, quantization, eigenspace computation, adapter injection, inference, and evaluation
- It is positioned as a bridge from a toy transformer to a real LLaMA-derived deployment path

---

Real-Model Upgrade Path

The README suggests the following path to move beyond the toy transformer:

- Replace `src/dac_q4_its/modeling/loader.py` to load real HF / LLaMA weights
- Populate `scripts/02_export_onnx.py` to export real projections

- Replace toy heads with real Q / K / V / FFN matrices
- Keep the rest of the pipeline structure intact

This is useful because it shows that the starter is intentionally modular rather than being locked to the toy model.

Current Limitations

- Public repo is still a starter, not a fully benchmarked release
- README is concise and focused on flow, not exhaustive evaluation
- Real deployment details for Jetson / TensorRT are not yet fully wired in the public starter
- The repo currently proves pipeline structure more than production-grade validation

---

What I Would Add Next

- Benchmark comparisons against FP16 / INT8 / plain INT4 baselines
- Memory usage charts for embedded automotive targets
- End-to-end latency numbers on Jetson-class hardware
- Accuracy trade-off analysis as rank changes
- A worked example using a real LLaMA-derived checkpoint

These additions would make the repo stronger as both a research artifact and an engineering deployment guide.

Key Innovation

> **Important** The main innovation is the combination of **INT4 quantization** with a **low-rank residual compensation path** in a pipeline designed for constrained, vehicle-oriented inference settings.

Conclusion

This repository is a strong starter for exploring **vehicle-scale LLM inference** under tight memory budgets. Its value is not only in compression, but in the fact that it demonstrates a full reproducible flow: prepare, export, quantize, compensate, inject, infer, and evaluate.

> From full-precision overhead to **vehicle-ready efficient inference** — that is the direction this project is pushing toward.

Source: src/content/posts/Vehicle-Scale-LLMs.md