1. Project Scope & Goals

- **Scope**: Build on an existing open-source LLM caching library to design, implement and evaluate an enhanced cache policy.

- **Primary Goal**: Demonstrate measurable improvement on some performance metric, and report your findings in a compelling and clear cut way that explains what you did, and establishes the claim to fame.

---

2. Selection of Baseline Framework

- **Criteria recomendation**

    - *Maturity & community support* (e.g. KV cache)

    - *Ease of modification* (clear code structure, tests, documentation)

- **Deliverable**: A one-page (max) justification of your choice, listing its main features, default eviction policy and other aspects relevant to your work.

---

3. Designing the Performance Test Suite

- **Workload Profiles**

  Your first step is to develop the ability to estimate the performance of your system. Here, you should write performance test, such as stress tests, hit-ratio measures, prompt handling time, throughput or latency. You focus both on finding a test suit (e.g., datasets or interesting scenarios to investigate) for example, *Repetitive short prompts* (to stress the hit/miss ratio,) *Novel long prompts (that are unlikely to be cached)* (to measure cache overhead).

- **Metrics to Collect**

    - Per-request latency (mean, p95, p99)

    - Cache hit rate
    - Memory & CPU/GPU utilization
    - Throughput (queries/tokens per second).
- **Automation**
    - Write benchmark scripts (e.g. with pytest-benchmark or custom Python)
    - Integrate into CI so all commits rerun the suite
- **Deliverable**: A README “how to benchmark” plus sample CSV or JSON logs.

---

4. Defining & Implementing Your Extension

Here you need to define your story, how are you going to improve the existing state of the open source project you choose. For example you can choose one of the following (but also other thing if they make sense).

- **Choose one or more strategies**, for example:
    - **Cost-Aware Eviction**: weigh entries by retrieval time or compute cost
    - **Hierarchical Cache**: small in-RAM LRU + larger on-disk LFU
    - **Approximate Matching**: embed prompts+responses and reuse close matches
- **API Considerations**
    - Maintain compatibility with baseline cache interface (I’ll give an exceptionally high grade (100) to project thats contribute to the open source community via push requests and have their improvement adopted).
    - Expose tunable parameters (cache size, weights, similarity threshold) and document them mainly, demonstrating their impact on performance.
- **Deliverable**:
    - Cleanly-factored code in a feature branch
    - Unit tests covering your new policy

---

5. Evaluation & Analysis

- **Experiment Design**
    - Run “vanilla” vs. “extended” under identical workloads
    - Sweep key parameters (e.g. cache sizes of 50 MB, 100 MB, 200 MB)
- **Data Analysis**
    - Plot latency distributions & hit-rate curves (and other things necessary to explore your improvement).
    - Compute relative improvements (e.g. % latency reduction at p95)
- **Ablation Study**
    - If you combined multiple ideas, isolate each to understand its impact
- **Deliverable**:
    - A pdf that describes your experiments, their result and the significance of these results or what each experiment actually tell you and how you use them to see the general picture.

---

6. Reporting & Publication

- **Code Publication**
    - Clean GitHub repo:
        - README with installation & benchmark instructions
        - Dockerfile or environment.yml for reproducibility
- **PDF Report Structure**
    - **Introduction** – problem statement & related work (baseline cache)

    - **Extension Design** – motivation and technical description of your policy
    - **Experimental Setup** – workloads, metrics, tooling, parameters
    - **Results** – tables & figures comparing baseline vs. extension
    - **Discussion** – analysis of trade-offs and sensitivity to parameters
    - **Conclusion & Future Work** - if you learned anything unexpected or have some ideas of how your project can be pushed forward. Keep it grounded though.
- **Formatting Tips**
    - Use consistent figure sizing and fonts (≥ 10 pt)
    - Label axes clearly, include legends & captions
    - Reference all code and data artifacts in an appendix
- **Deliverable**: A single PDF (8–12 pages) plus any appendices or supplementary materials.

---

7. Success Criteria

- **Correctness**: cache behaves as expected (passes tests) (40%)
- **Reproducibility**: anyone can clone & rerun your benchmarks (30%)
- **Performance Gain**: clear, statistically significant improvements (15%)
- **Clarity**: code is well-documented; report tells a coherent story (15%)

It is your goal to convince me that you pass these success criteria. That is, you need to convince me that the code is correct, reproducible, that there is a performance gain and that the results are clear. **I grade your assignment according to my impression** about how convinced I am of correctness, reproducibility, performance gain, and the clarity of the report (including quality of writing and presentation).

Use this guideline as your skeleton—fill in each section iteratively, and you'll end up with a well-structured, publishable project. Good luck!