**Title:** Comparative Study: Rust vs. C# for CLI and Batch Processing Tools

**🧩 Executive-Level Final Report Structure**

**📌 1. Introduction**

* Problem statement: Need for performant and maintainable backend tooling
* Project scope: CLI and batch tools — **no microservices**
* Objective: Compare Rust vs. C# on performance, efficiency, and maintainability

**🧱 2. Background**

* Why this comparison matters (e.g., GC vs. zero-cost, ecosystem trade-offs)
* Stakeholders and archetypes (summarized from advisory doc)
* Scope boundaries (excluded: API services, GUIs, DB integration)

**🛠 3. Technical Setup Summary *(link to design doc)***

* Tooling stack: Rust, C#, .NET 9.0
* Code structure (multi-project layout)
* Benchmark tooling used (hyperfine, PowerShell)
* Dataset description (large\_input.csv)

**🔬 4. Benchmark Results Summary *(link to full analysis doc)***

* Key results from:
  + Manual tool
  + Batch job
  + Parallel job
* One clean table with all performance metrics (time, memory, CPU)
* Charts (if included)
* Interpretation: where/why Rust wins

**🧩 5. Client Fit & Strategy Summary *(link to strategy doc)***

* Recap of client archetypes
* Rust fit per client
* Migration recommendations
* Prioritization table

**🔁 6. Evaluation & Reflection**

* What worked well (e.g., benchmarking repeatability, tool parity)
* Challenges faced (e.g., PLINQ overhead, matching structure)
* What you would change in a second iteration (e.g., async, file streaming vs. memory load)

**🎯 7. Conclusion**

* Final statement: Is Rust "worth it"?
* Where Rust should be used in tooling pipelines
* Where C# remains appropriate
* Practical recommendation: hybrid, targeted migration

**📎 8. Appendices**

* Table of tool versions
* System hardware specs
* CLI commands used
* Link to GitHub repo or local codebase structure