PROJECT EVALUATION RUBRIC
Scoring Guide for Technical Project Assessment

OVERVIEW

This rubric is used to evaluate technical project implementations, specifically for backend systems involving AI/LLM integration, RAG implementations, and case study submissions. Each parameter is scored on a scale of 1-5. The final score is a weighted average of all parameters.

SCORING PARAMETERS

1. CORRECTNESS (PROMPT & CHAINING) (Weight: 30%)

Description:
Evaluate whether the project correctly implements the required functionality, including prompt design, LLM chaining logic, RAG context injection, and overall system correctness.

Scoring Scale:

1 - NOT IMPLEMENTED
• Core functionality is missing
• No LLM integration or it doesn't work
• No RAG implementation
• Requirements not met
• System doesn't run or crashes immediately

2 - MINIMAL ATTEMPT
• Basic structure present but mostly incomplete
• LLM integration exists but doesn't work properly
• No actual chaining or very broken chaining
• RAG attempted but not functional
• Many core requirements missing
• Significant bugs prevent proper operation

3 - WORKS PARTIALLY
• Core functionality partially implemented
• LLM integration works for simple cases
• Basic chaining logic present but limited
• RAG retrieves context but integration is weak
• Some requirements met, others missing
• Works for happy path but fails on edge cases

4 - WORKS CORRECTLY
• All core functionality implemented and working
• Proper LLM integration with good prompts
• Chaining logic works reliably
• RAG effectively retrieves and injects context
• All major requirements met
• Handles most edge cases
• Minor issues or missing optimizations

5 - FULLY CORRECT + THOUGHTFUL
• Excellent implementation of all requirements
• Sophisticated prompt design with clear structure
• Robust multi-stage LLM chaining
• Effective RAG with proper chunking and retrieval
• All requirements exceeded
• Thoughtful design decisions
• Handles edge cases elegantly
• Goes beyond basic requirements with intelligent solutions

2. CODE QUALITY & STRUCTURE (Weight: 25%)

Description:
Evaluate the cleanliness, modularity, reusability, and testability of the code.

Scoring Scale:

1 - POOR
• Messy, unorganized code
• No structure or separation of concerns
• Hard-coded values everywhere
• No comments or documentation
• Copy-pasted code with lots of duplication
• Very difficult to understand or modify

2 - SOME STRUCTURE
• Basic organization present
• Some separation of concerns
• Limited modularity
• Minimal comments
• Some code duplication
• Difficult to extend or maintain

3 - DECENT MODULARITY
• Good basic structure
• Clear separation of concerns
• Functions/classes are reasonably sized
• Some reusable components
• Basic error handling
• Code is readable with some comments
• Could use more abstraction
• No tests or very minimal testing

4 - GOOD STRUCTURE + SOME TESTS
• Clean, well-organized code
• Strong separation of concerns
• Good modularity and abstraction
• Reusable components
• Clear naming conventions
• Well-commented where needed
• Configuration separated from code
• Some unit tests or integration tests
• Easy to understand and extend

5 - EXCELLENT QUALITY + STRONG TESTS
• Exceptional code quality
• Excellent architecture and design patterns
• Highly modular and reusable
• Comprehensive error handling
• Clear, self-documenting code
• Configuration-driven
• Comprehensive test coverage (unit + integration)
• Follows industry best practices
• Production-ready code quality

3. RESILIENCE & ERROR HANDLING (Weight: 20%)

Description:
Evaluate how well the system handles failures, implements retry logic, manages long-running processes, and controls randomness/nondeterminism.

Scoring Scale:

1 - MISSING
• No error handling
• System crashes on errors

• No retry logic
• No timeout protection
• Doesn't handle API failures
• LLM randomness not controlled

2 - MINIMAL
• Basic try-catch blocks
• Generic error messages
• No retry mechanisms
• Limited error logging
• Fails ungracefully on API errors
• No consideration for LLM stability

3 - PARTIAL HANDLING
• Some error handling present
• Basic retry logic for some operations
• Some timeout protection
• Logs errors but limited detail
• Handles some failure scenarios
• Basic LLM temperature control
• Inconsistent error handling across the system

4 - SOLID HANDLING
• Good error handling throughout
• Retry logic with exponential backoff
• Proper timeout protection
• Detailed error logging
• Handles most failure scenarios gracefully
• LLM temperature controlled for consistency
• Graceful degradation on failures
• Clear error messages to users

5 - ROBUST, PRODUCTION-READY
• Comprehensive error handling
• Intelligent retry strategies with backoff
• Circuit breaker pattern for external services
• Detailed logging and monitoring
• Handles all foreseeable failure scenarios
• Multiple strategies for LLM consistency (temperature + validation)
• Graceful degradation with partial results
• Dead letter queue for failed jobs
• Health checks and self-healing capabilities
• Production-grade resilience patterns

4. DOCUMENTATION & EXPLANATION (Weight: 15%)

Description:
Evaluate the quality of README, setup instructions, code comments, and explanations of design decisions and trade-offs.

Scoring Scale:

1 - MISSING
• No README or very minimal
• No setup instructions
• No explanation of how to run the project
• No documentation of design decisions
• No comments in code

2 - MINIMAL
• Basic README present
• Incomplete setup instructions
• Missing many details
• No explanation of design choices
• Minimal or no code comments
• Difficult to get started

3 - ADEQUATE
• README covers basics
• Setup instructions present but may lack detail
• Basic explanation of project structure
• Some design decisions mentioned
• Basic code comments
• Possible to get started with some effort
• Missing some important details

4 - CLEAR
• Comprehensive README
• Clear, step-by-step setup instructions
• Good explanation of architecture
• Design decisions documented
• Code is well-commented
• Easy to get started and understand
• Includes examples of usage
• Mentions some trade-offs

5 - EXCELLENT + INSIGHTFUL

• Outstanding documentation
• Multiple documentation files (README, SETUP, ARCHITECTURE)
• Crystal-clear setup instructions with troubleshooting
• Detailed explanation of all design decisions
• Thorough discussion of trade-offs and alternatives considered
• Excellent code comments where needed
• API documentation (e.g., Swagger/OpenAPI)
• Diagrams or visualizations of architecture
• Examples and usage scenarios
• Rationale for technology choices
• Known limitations and future improvements discussed

5. CREATIVITY / BONUS (Weight: 10%)

Description:
Evaluate additional features, innovative solutions, and improvements beyond the basic requirements.

Scoring Scale:

1 - NONE
• Strictly minimum requirements only
• No additional features
• No innovation or creative solutions
• Basic implementation throughout

2 - VERY BASIC
• One or two minor extras
• Basic implementation of bonuses
• Limited additional value
• Extras don't add much to the project

3 - USEFUL EXTRAS
• Several additional features that add value
• Examples:
  - Additional API endpoints (health check, statistics)
  - Basic monitoring or logging
  - Docker setup for easy deployment
  - Environment-based configuration
• Thoughtful but not exceptional additions

4 - STRONG ENHANCEMENTS
• Multiple valuable additions beyond requirements

• Examples:
  - Comprehensive API with extra endpoints
  - Advanced monitoring (e.g., Flower for Celery)
  - Full Docker Compose setup
  - Multiple LLM provider support
  - Caching layer
  - API rate limiting
  - Admin dashboard
• Demonstrates strong engineering thinking
• Adds significant value to the project

5 - OUTSTANDING CREATIVITY
• Exceptional additional features and innovations
• Examples:
  - Multi-model ensemble approach
  - Real-time streaming updates (WebSocket/SSE)
  - Advanced caching with invalidation
  - A/B testing framework for prompts
  - Automated prompt optimization
  - Comprehensive monitoring and alerting
  - Authentication and authorization system
  - Deployment to cloud with CI/CD
  - Performance optimizations
  - Unique innovative solutions to problems
• Production-ready additional features
• Shows exceptional technical depth and breadth

CALCULATION METHOD

Final Project Score = Weighted Average of All Parameters

Example Calculation:
• Correctness: 4 (weight 0.30) = 1.20
• Code Quality: 4 (weight 0.25) = 1.00
• Resilience: 4 (weight 0.20) = 0.80
• Documentation: 5 (weight 0.15) = 0.75
• Creativity: 3 (weight 0.10) = 0.30

Final Score = 1.20 + 1.00 + 0.80 + 0.75 + 0.30 = 4.05

INTERPRETATION GUIDE

Project Score Interpretation:
• 4.5 - 5.0: Exceptional work, exceeds expectations
• 4.0 - 4.4: Strong work, meets all expectations
• 3.5 - 3.9: Good work, meets most expectations
• 3.0 - 3.4: Adequate work, meets basic expectations
• 2.5 - 2.9: Below expectations, significant gaps
• Below 2.5: Well below expectations, major issues

EVALUATION GUIDELINES

1. Focus on implementation quality, not just functionality
2. Consider both correctness and code craftsmanship
3. Value resilience and production-readiness
4. Reward clear communication through documentation
5. Appreciate creative solutions and extras
6. Be fair but maintain high standards
7. Provide specific, actionable feedback

COMMON PITFALLS TO WATCH FOR

• Hard-coded credentials or API keys in code
• No error handling for external API calls
• Synchronous operations that should be asynchronous
• Missing or inadequate documentation
• Copy-pasted code without understanding
• No separation between configuration and code
• Overly complex solutions to simple problems
• No consideration for edge cases
• Missing tests entirely
• Poor naming conventions

EXCELLENCE INDICATORS

• Clean architecture with clear separation of concerns
• Comprehensive error handling with graceful degradation
• Well-designed prompts with context injection
• Effective use of RAG for improved accuracy
• Proper async processing for long-running tasks
• Extensive documentation with diagrams
• Test coverage for critical paths
• Production-ready deployment setup

• Thoughtful trade-off discussions
• Creative solutions that add value

NOTES FOR EVALUATORS

• Provide constructive feedback for improvement
• Highlight specific examples of good and bad practices
• Consider the candidate's experience level
• Focus on potential and learning ability
• Value clean, simple solutions over complex ones
• Encourage best practices and production thinking
• Recognize effort in documentation and testing
• Appreciate thoughtful design decisions