Low‑Level Technical Specification: AI Estimation Service

### **1. Project Structure & Modules**

* **Entry Point**
  + app.py – starts FastAPI server, loads model, initializes router
* **API Layer** (api/)  
  + routers/estimate.py – defines /estimate endpoint
  + models.py – Pydantic request/response schemas
* **Inference Engine** (inference/)  
  + llama\_client.py – wrapper around llama.cpp for model loading & inference
  + prompt\_builder.py – functions to construct and sanitize prompts
* **Cache & Queue** (cache/)  
  + redis\_client.py – Redis connection for result caching and rate‑limiting
* **Config & Utilities** (utils/)  
  + config.py – load and validate ENV settings via Pydantic/BaseSettings
  + logger.py – structlog or Python logging setup
  + errors.py – custom exception classes and handlers

### **2. Dependencies & Versions**

* **Python Runtime**: Python 3.10+ (PSF license)
* **Web Framework**: FastAPI v0.95 (MIT)
* **ASGI Server**: Uvicorn v0.22 (MIT)
* **Model Inference**: llama.cpp Python bindings (latest, MIT)
* **HTTP Client**: httpx v0.24 (BSD‑3)
* **Caching**: redis-py v5.x (MIT)
* **Configuration**: pydantic v2.x (MIT)
* **Logging**: structlog v24.x (MIT) or stdlib logging
* **Testing**: pytest v8.x (MIT), pytest-asyncio v0.21 (MIT)

### **3. API Contract**

* **POST** /estimate

**Request JSON** json  
CopyEdit  
{

"storyId": "uuid-string",

"description": "Detailed user story text."

}

**Response JSON** json  
CopyEdit  
{

"storyId": "uuid-string",

"suggestedValue": 5,

"confidence": 0.82,

"modelVersion": "llama-7B-v1",

"latencyMs": 120

}

* + **Status Codes**
    - 200 OK – valid suggestion returned
    - 400 Bad Request – missing/invalid payload
    - 503 Service Unavailable – model not loaded or overloaded

### **4. Inference & Prompting**

* **Model Initialization**
  + Load LLaMA 7B quantized model via llama.cpp on service start
  + Keep single global model instance to amortize load time

**Prompt Template** sql  
CopyEdit  
You are an agile estimation assistant. Given the following user story, suggest a Fibonacci-based story point estimate (1,2,3,5,8,13…).

Story: "{description}"

Estimate:

* **Inference Settings**
  + n\_ctx: 2048 tokens
  + temperature: 0.2
  + top\_p: 0.95
  + n\_threads: auto-detect cores minus one
* **Post‑Processing**
  + Parse numeric output; clamp to nearest valid Fibonacci value
  + Compute confidence as 1 − (output entropy / max entropy)

### **5. Performance & Scalability**

* **Latency Target**: ≤ 300 ms 90th percentile per request
* **Concurrency**: handle up to 50 simultaneous inference calls
* **Batching** (future): allow small prompt batches (e.g., queue 5 requests within 50 ms window)
* **Caching**
  + Cache suggestions per storyId for 24 h in Redis
  + Key: ai:suggestion:{storyId} → JSON response

### **6. Error Handling & Retries**

* **Timeout**: abort inference if > 500 ms, return 503
* **Retries**: on llama.cpp internal error, retry once after 100 ms backoff
* **Circuit Breaker**: if > 10 errors in 60 s, short‑circuit further calls for 30 s
* **Validation Errors**: respond 400 with Pydantic error details

### **7. Logging & Monitoring**

* **Request Logging**
  + Log storyId, payload size, response status, latency
* **Error Logging**
  + Stack trace on inference failures or timeouts
* **Metrics** (Prometheus via prom-client equivalent)  
  + Counter: ai\_requests\_total{status="success"|"error"}
  + Histogram: ai\_request\_latency\_ms
  + Gauge: ai\_queue\_length (if batching implemented)

### **8. Configuration & Environment**

* **Environment Variables** (managed via config.py)  
  + MODEL\_PATH – filesystem path to quantized model
  + REDIS\_URL – e.g., redis://localhost:6379/0
  + SERVER\_HOST, SERVER\_PORT
  + MAX\_WORKERS – threads for inference
  + TIMEOUT\_MS, RETRY\_COUNT
* **Secure Handling**: do not log raw story descriptions in production logs

### **9. Security Considerations**

* **Input Sanitization**: strip control characters, enforce max length (e.g., 2000 chars)
* **Rate Limiting**: per‑IP or per‑user limit (e.g., 30 requests/min) via Redis
* **CORS**: restrict origins to known frontend domains

### **10. Testing Strategy**

* **Unit Tests**
  + Prompt builder outputs correct template
  + Llama client returns numeric suggestion for mocked responses
* **Integration Tests**
  + Spin up test Redis; send real POST /estimate using small dummy model
* **Performance Tests**
  + Benchmark single and concurrent requests via pytest-benchmark
* **Error Scenarios**
  + Invalid payload, model unavailable, Redis down

### **11. Deployment**

* **Containerization**: single Docker image  
  + Base: python:3.10-slim
  + Copy model weights into /models/
  + Entry: uvicorn app:app --host 0.0.0.0 --port $SERVER\_PORT --workers 1
* **Resource Requests** (K8s)  
  + requests: 2 CPU, 4 GB RAM
  + limits: 4 CPU, 8 GB RAM
* **Health Check**
  + GET /healthz verifying model loaded and Redis reachable