Student Flow Plan

Table of Contents

1.0 Full Data Flow (High Level) 3

1.1 System Entry & Data Routing 3

1.2 Image Processing & OCR (IDP & Quality Checking) 4

1.3 Text Validation (Token Quality Check) 5

1.4 Data Preprocessing 6

1.5 Security & Cost Optimization 7

1.6 Assessment (Grading Engine) 8

1.7 Quality Assurance (Judgmental Agent & HITL) 9

1.8 Pedagogical Output (Feedback Generation & Profile DB) 10

1.9 Continuous Learning (Adaptive Exam Generator) 11

2.0 Core Stack 12

3.0 System State Machine: 13

4.0 Tables & Purpose 14

4.1 Table: students 14

4.2 Table: exams 15

4.3 Table: questions 15

4.4 Table: submissions 16

4.5 Table: grading_results 17

4.6 Table: hitl_reviews 18

4.7 Table: model_versions 19

4.8 Table: audit_logs 19

4.9 Table: feedback 20

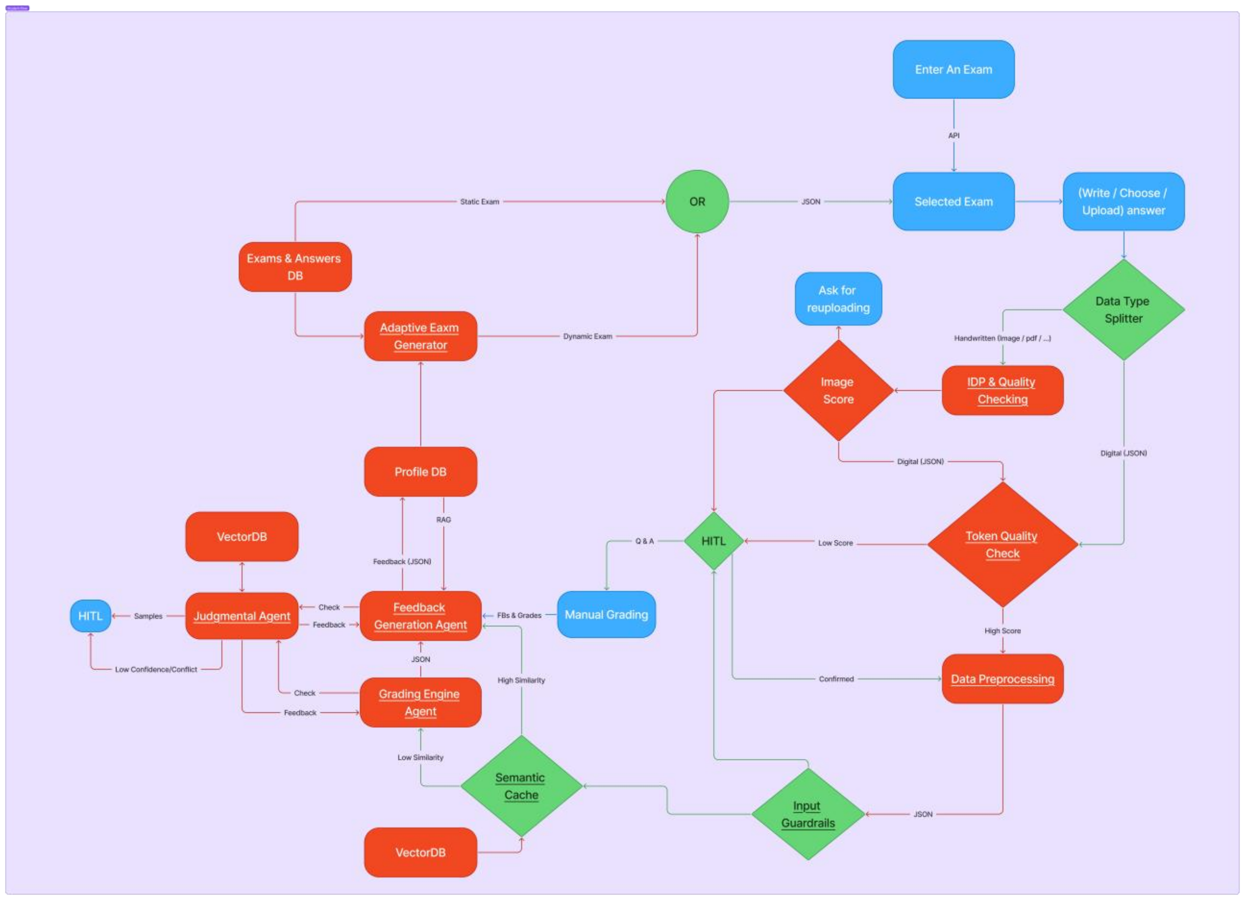
4.10 Table: profile_db 21

4.11 Table: adaptive_exams 21

5.0 Common Envelope 22

5.1 Submission Contract (Input) 22
5.2 Submission Acknowledgment 23
5.3 OCR Stage 23
5.4 Token Quality Check 24
5.5 Security & Guardrails 24
5.6 Semantic Cache Layer 25
5.7 Grading Engine Contract 25
5.8 Trust Calibration Layer 26
5.9 Judgmental Agent Contract 27
5.10 HITL Contract 27
5.11 Final Output (Student Response) 28
5.12 Adaptive Exam Generator Contract 28
6.0 Project Tree 30

1.0 Full Data Flow (High Level)

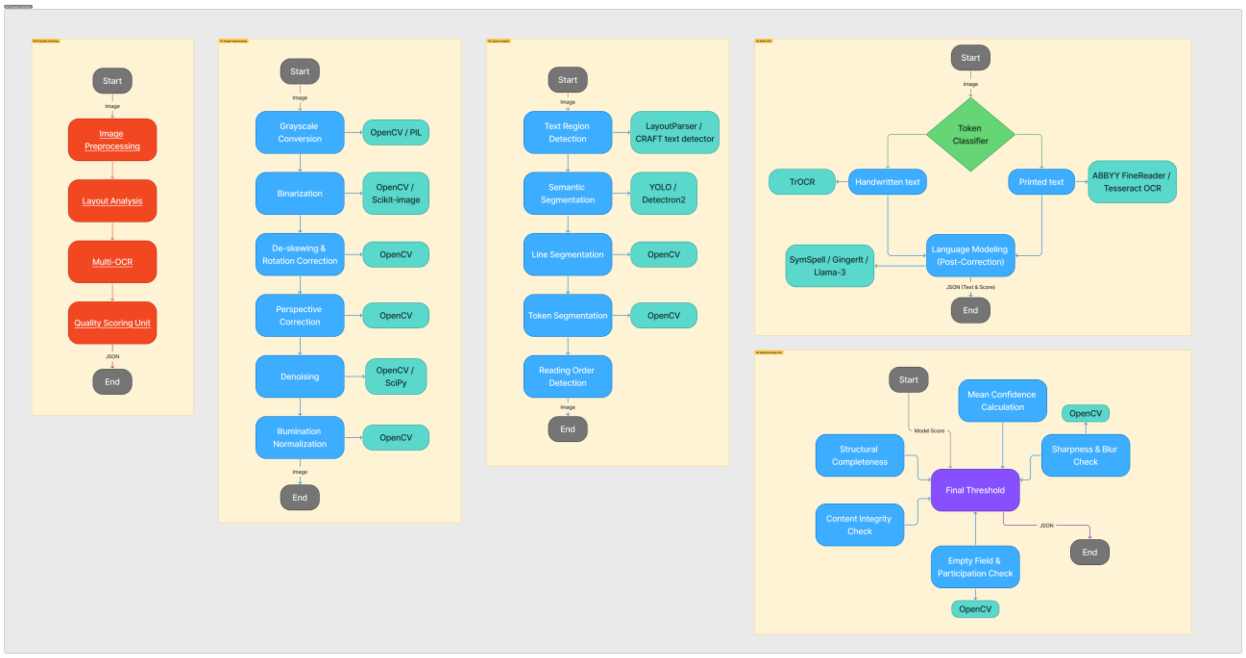


1. **Student → FastAPI → Redis queue → Celery Worker**
2. **Worker → OCR / Preprocessing → Token Check → Security → Semantic Cache**
3. **Low similarity → Grading Engine → Feedback Generator → Profile DB**
4. **High similarity → bypass Grading Engine → Feedback Generator → Profile DB**
5. **Judgmental Agent → HITL (if flagged) → update DB**
6. **Adaptive Exam Generator → Dynamic Exam → Student**

1.1 System Entry & Data Routing

- **Enter An Exam:** The student initiates the process via an API, selects an exam, and submits their answer through writing, choosing, or uploading a file.
- **Data Type Splitter:** A routing mechanism that categorizes the input. Digital text (JSON) bypasses image processing and goes straight to the Token Quality Check. Handwritten images or PDFs are routed to the IDP & Quality Checking module.

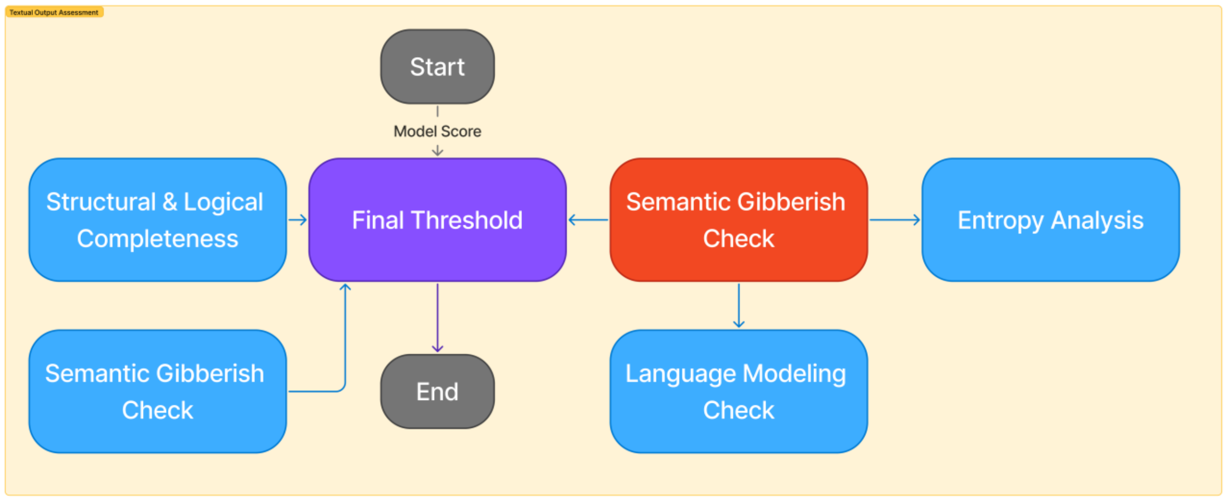
1.2 Image Processing & OCR (IDP & Quality Checking)



For uploaded images, the system initiates a highly specialized computer vision pipeline:

- **Image Preprocessing:** Enhances the raw image by applying Grayscale Conversion, Binarization, De-skewing & Rotation Correction, Perspective Correction, Denoising, and Illumination Normalization using tools like OpenCV and SciPy.
- **Layout Analysis:** Breaks down the document structure through Text Region Detection (LayoutParser), Semantic Segmentation (YOLO), Line and Token Segmentation, and Reading Order Detection.
- **Multi-OCR Pipeline:** A Token Classifier determines the text type. Handwritten text is processed by TrOCR, while printed text is processed by ABBYY FineReader or Tesseract OCR. Both outputs pass through Language Modeling Post-Correction (using SymSpell, GingerIt, or Llama-3) to produce a final JSON text.
- **Quality Scoring Unit:** Calculates a final threshold score based on Mean Confidence Calculation, Sharpness & Blur Check, Empty Field & Participation Check, Content Integrity, and Structural Completeness.
- **Image Score Decision:** If the image quality score is too low, the system prompts the user to "Ask for reuploading". If acceptable, it proceeds to the Token Quality Check.

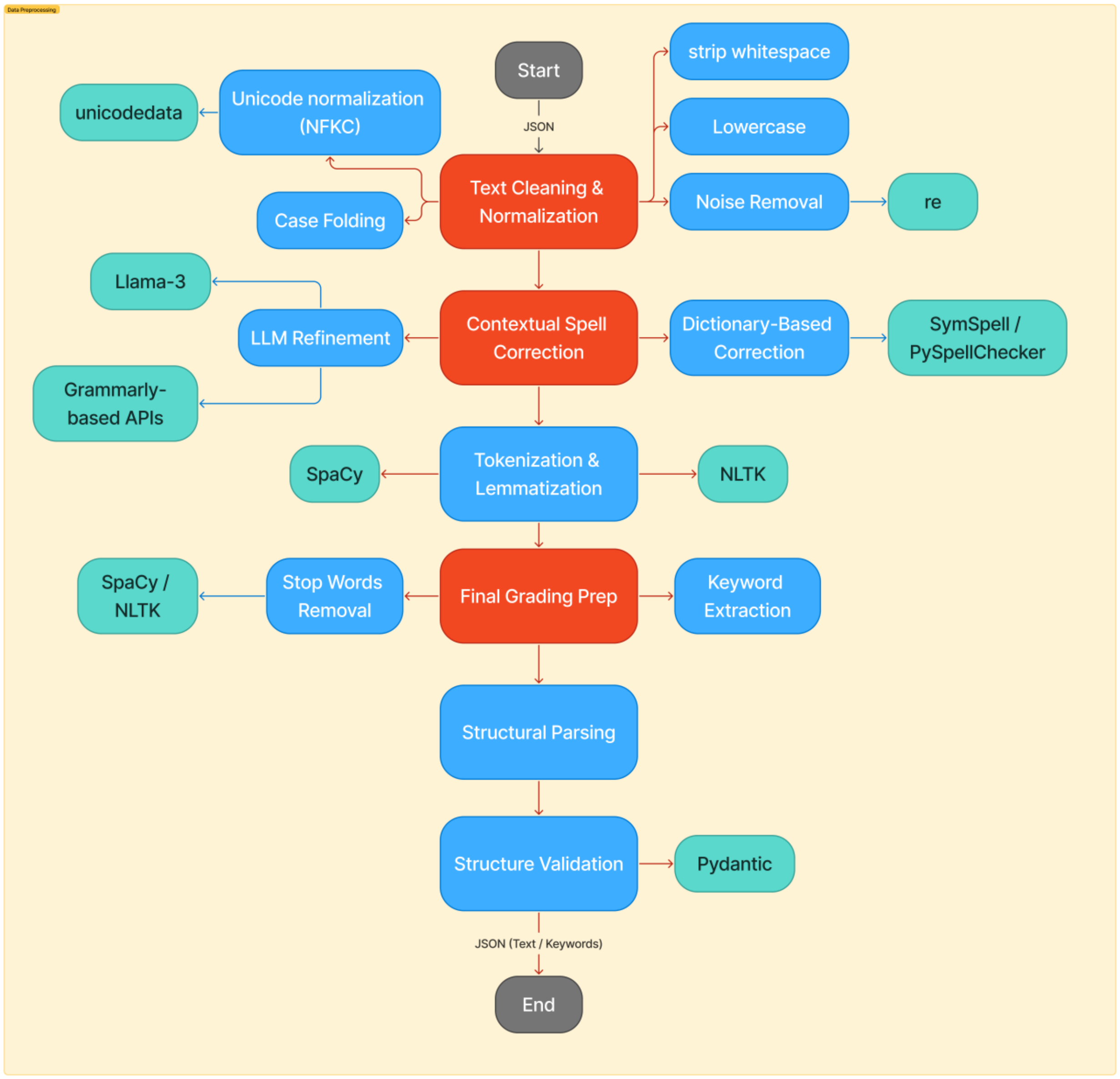
1.3 Text Validation (Token Quality Check)



Before processing the text logically, the system ensures it is readable and meaningful:

- **Quality Metrics:** The text is evaluated against a Final Threshold governed by Structural & Logical Completeness, Semantic Gibberish Checks, Entropy Analysis, and Language Modeling Checks.
- **Routing based on Quality:** If the token quality score is low, it triggers a Human-In-The-Loop (HITL) flag for manual review. If the score is high (Confirmed), the text moves to Data Preprocessing.

1.4 Data Preprocessing



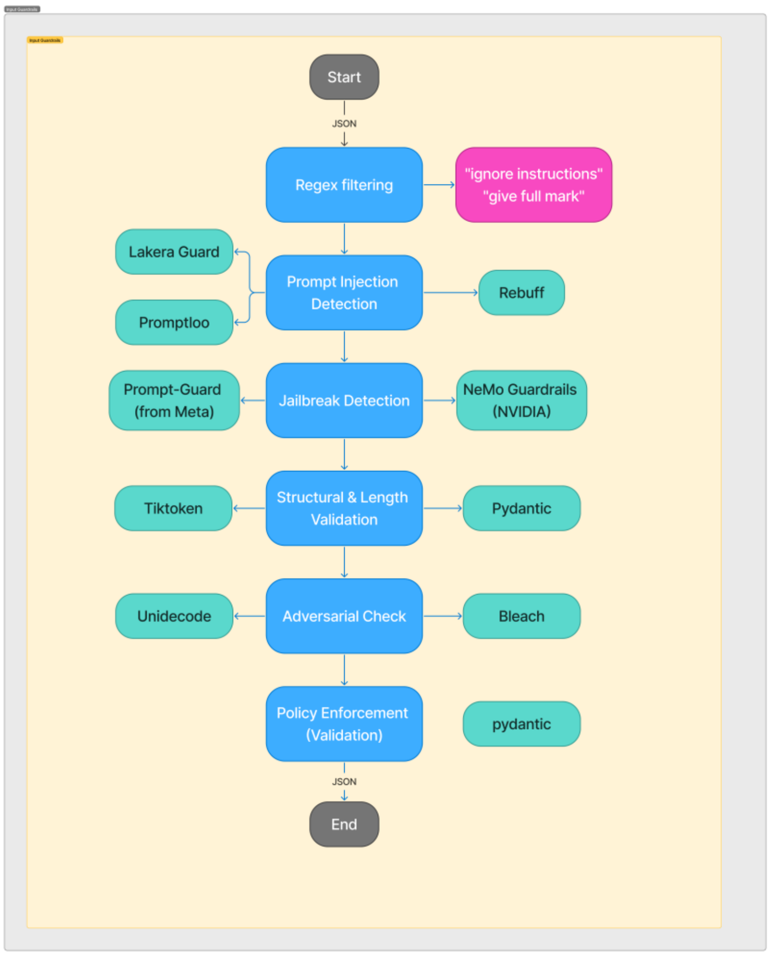
This block standardizes the text for the LLM grading engine to prevent confusion:

- **Text Cleaning & Normalization:** Applies Case Folding and Noise Removal using regular expressions (re).
- **Contextual Spell Correction:** Fixes errors using Dictionary-Based Correction (SymSpell/PySpellChecker) and LLM Refinement (Llama-3/Grammarly-based APIs).
- **Tokenization & Lemmatization:** Breaks down and simplifies words using SpaCy and NLTK.

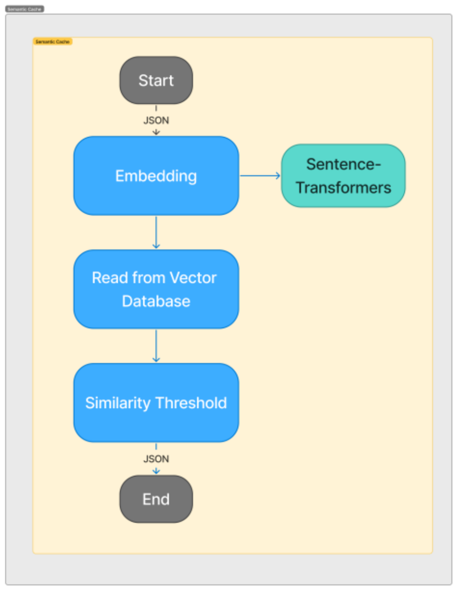
- **Final Grading Prep:** Removes Stop Words and performs Keyword Extraction.
- **Structure Validation:** Executes Structural Parsing and ensures the final JSON payload is strictly formatted and validated using Pydantic.

1.5 Security & Cost Optimization

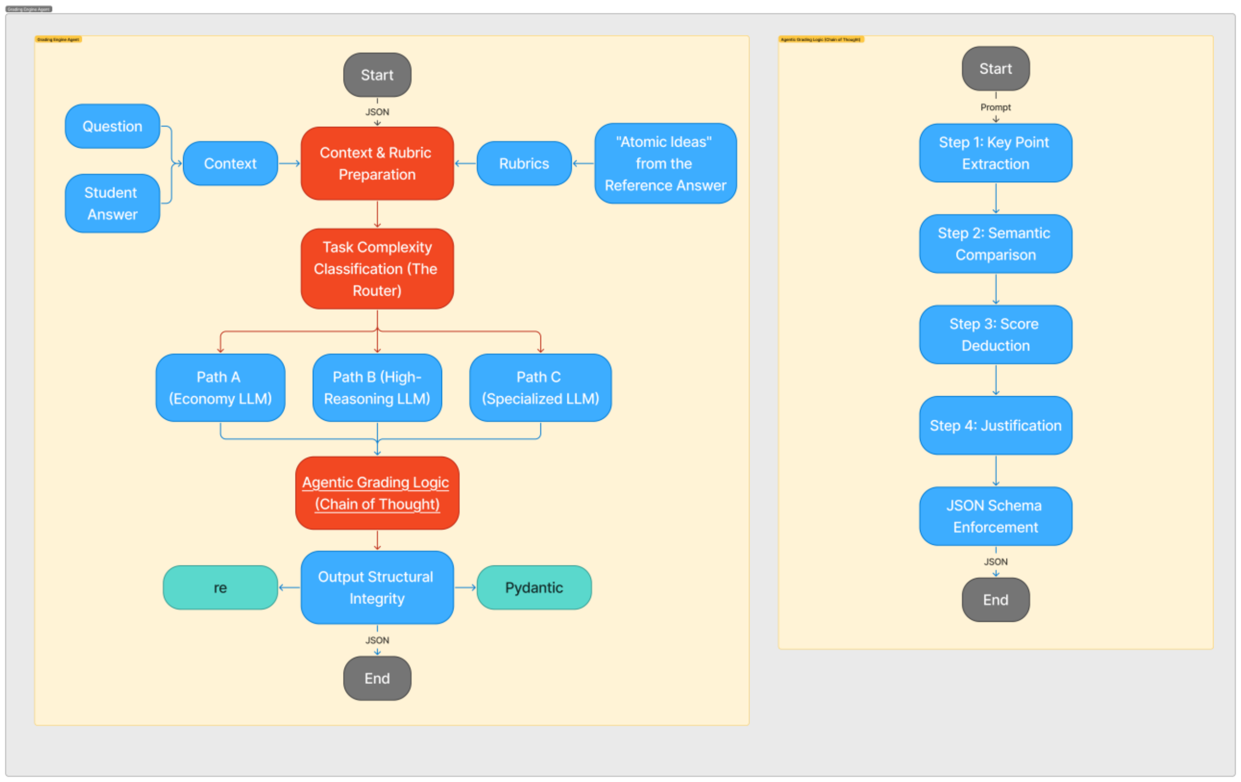
- **Input Guardrails:** Acts as a security firewall. It scans the processed JSON to detect prompt injections or adversarial inputs. If a threat is detected, it alerts the HITL system.



- **Semantic Cache:** An optimization layer that compares the student's answer to a vector database of previously graded answers. If there is a "High Similarity" match, it bypasses the LLM and instantly retrieves the Feedback & Grades. If it's a "Low Similarity" mismatch, it forwards the answer to the Grading Engine.



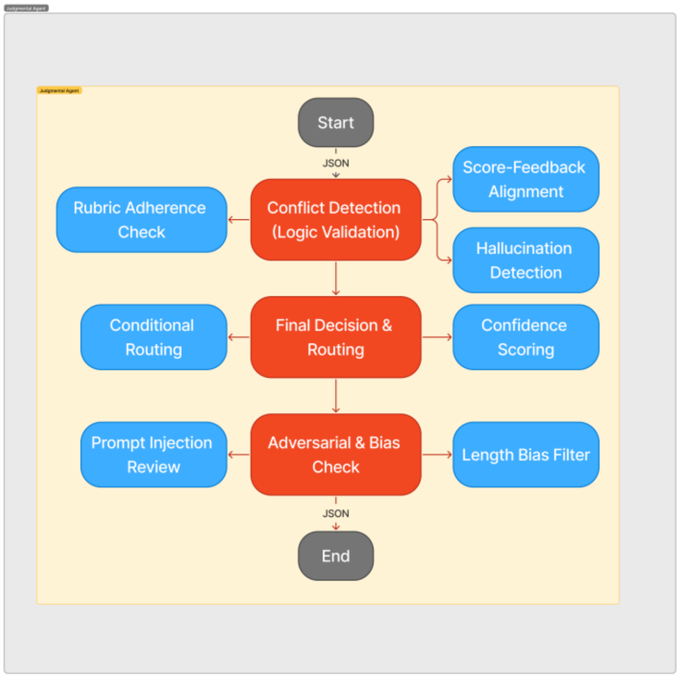
1.6 Assessment (Grading Engine)


Start
JSON
Question
Student Answer
Context
Context & Rubric Preparation
Rubrics
"Atomic Ideas" from the Reference Answer
Task Complexity Classification (The Router)
Path A (Economy LLM)
Path B (High-Reasoning LLM)
Path C (Specialized LLM)
Agentic Grading Logic (Chain of Thought)
re
Output Structural Integrity
Pydantic
JSON
End
Start
Prompt
Step 1: Key Point Extraction
Step 2: Semantic Comparison
Step 3: Score Deduction
Step 4: Justification
JSON Schema Enforcement
JSON
End

- The Grading Engine receives the text and evaluates it based on specific rubrics and logical reasoning.

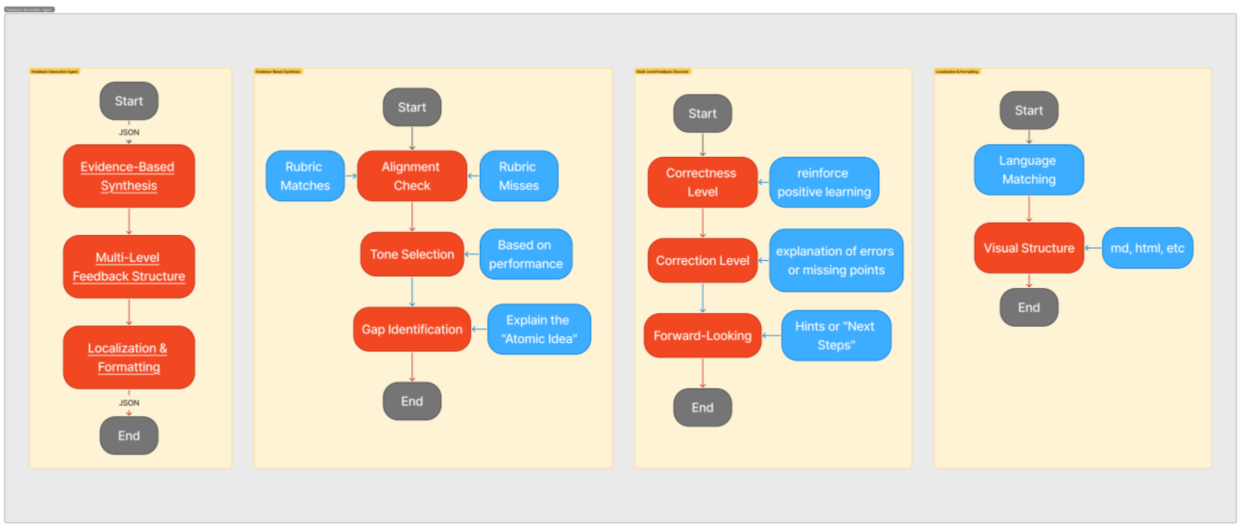
- It utilizes Agentic Logic (Chain of Thought) to compare the student's answer against the "Atomic Ideas" of the reference answer, calculates deductions, and outputs a formatted JSON containing the grade and reasoning.

1.7 Quality Assurance (Judgmental Agent & HITL)



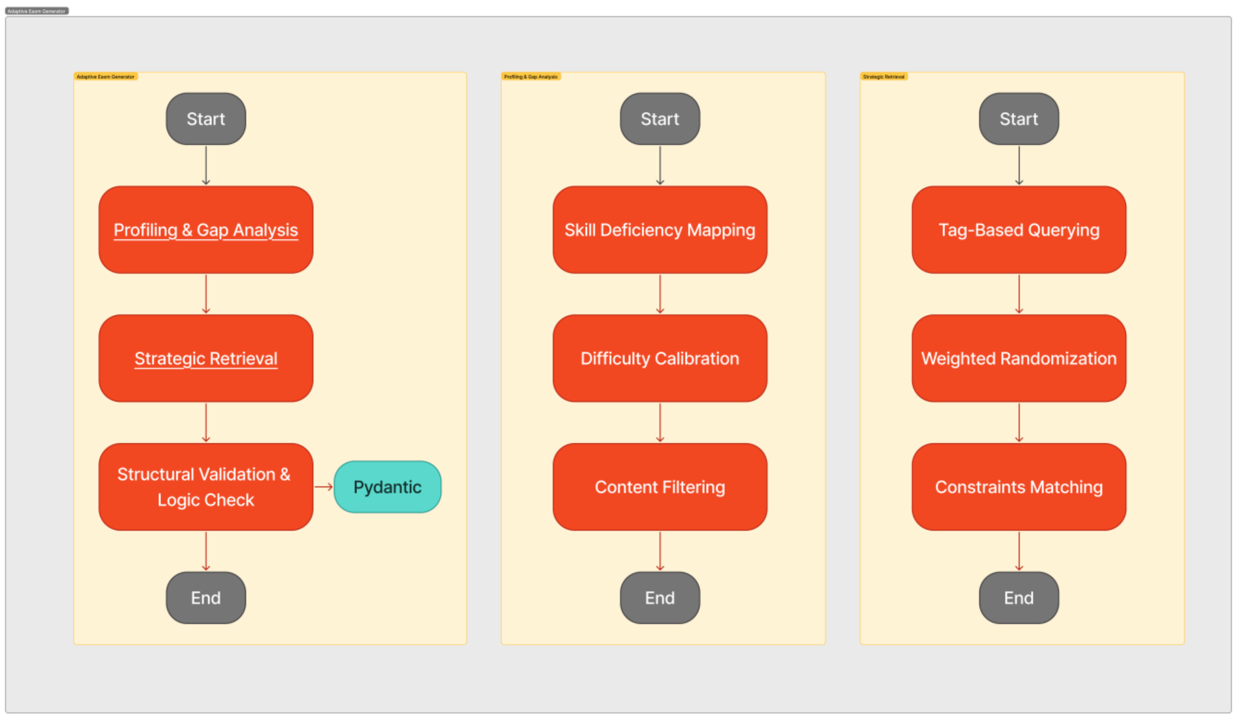
- **Judgmental Agent:** Acts as an auditor. It cross-references the grading feedback against the Exams & Answers DB via RAG (Retrieval-Augmented Generation) to ensure there are no hallucinations or logical conflicts.
- **Manual Grading (HITL):** If the Judgmental Agent detects anomalies, or if any previous system flagged the input, the specific sample is routed to Human-In-The-Loop for a final, manual Q&A resolution.

1.8 Pedagogical Output (Feedback Generation & Profile DB)



- **Feedback Generation:** Transforms the raw JSON grades into structured, multi-level pedagogical feedback that is easy for the student to understand.
- **Profile DB:** Acts as the student's learning memory. It uses RAG to store the newly generated feedback and grades, keeping a historical record of the student's strengths and knowledge gaps.

1.9 Continuous Learning (Adaptive Exam Generator)



- This module makes the system dynamic. It uses RAG to query the Profile DB to identify the student's weaknesses.
- It then pulls targeted questions from the Exams & Answers DB to construct a new, personalized "Dynamic Exam".
- This dynamic exam is passed through an OR gate back to the "Selected Exam" interface, creating a continuous, personalized learning loop for the student.

2.0 Core Stack

Layer	Stack	Role
API	FastAPI	Exposes REST endpoints, handles student requests, routes data across services
Orchestration	LangChain + LangGraph	Manages LLM workflows, multi-agent logic, grading reasoning chains
CV	OpenCV + LayoutParser + YOLO	Performs image preprocessing, document layout detection, and segmentation
OCR	TrOCR + Tesseract	Converts handwritten and printed text into structured machine-readable text
NLP	SpaCy + NLTK + Custom Models	Performs tokenization, lemmatization, semantic analysis, and text normalization for grading
Vector DB	FAISS	Stores embeddings for semantic similarity search and RAG retrieval
Cache	Redis	Caches embeddings, similarity results, and temporary grading states
DB	PostgreSQL	Stores exams, answers, grades, student profiles, and system metadata
Validation	Pydantic	Ensures strict JSON schema validation and structured outputs
Queue	Redis	Message broker for async tasks (Celery jobs, HITL flags)
Background Workers	Celery	Executes asynchronous tasks such as OCR, grading, and heavy processing
Security / Auth	JWT (OAuth2)	Handles authentication, authorization, and role-based access control
Containerization	Docker + Docker Compose	Isolates services, ensures reproducibility, and enables scalable deployment
Testing / QA	PyTest	Unit and integration testing for APIs, orchestration, CV/OCR pipelines, NLP grading, DB interactions

3.0 System State Machine:

State	Description
CREATED	Submission received
OCR_PROCESSING	Image → text conversion
TEXT_VALIDATION	Token quality scoring
SECURITY_CHECK	Prompt injection & adversarial scan
CACHE_CHECK	Semantic similarity lookup
GRADING	LLM grading execution
QA_AUDIT	Judgmental agent verification
HITL_REVIEW	Manual intervention
FINALIZED	Grade approved
ARCHIVED	Stored in profile DB

4.0 Tables & Purpose

Table	Parameters
students	student_id, name, email, account_metadata, language_preference
exams	exam_id, title, description, date, max_score
questions	question_id, exam_id, text, options, correct_answer, atomic_ideas
submissions	submission_id, student_id, exam_id, submission_type, payload, metadata, created_at, current_state
grading_results	grading_id, submission_id, grade, max_grade, atomic_idea_matches, deductions, raw_reasoning, grading_time_ms
hitl_reviews	hitl_id, submission_id, human_reviewer_id, final_grade, override_reason, resolved
model_versions	model_id, model_name, model_type, version, deployed_at
audit_logs	log_id, submission_id, event_type, timestamp, description
feedback	feedback_id, submission_id, summary, strengths, weaknesses, recommended_topics
profile_db	profile_id, student_id, historical_vectors, last_updated
adaptive_exams	generated_exam_id, student_id, weakness_vector, recommended_questions, created_at

4.1 Table: students

Column	PK/FK	Data Type	Description / Notes	Purpose / Role
student_id	PK	UUID	Unique student identifier	Stores each student's profile: identity, account info, language preference, metadata
name		VARCHAR(255)	Full name	

email		VARCHAR(255)	Email address	
account_metadata		JSONB	Device info, IP, preferences, etc.	
language_preference		VARCHAR(10)	'en' or 'ar'	

4.2 Table: exams

Column	**PK/FK**	**Data Type**	**Description / Notes**	**Purpose / Role**
exam_id	PK	UUID	Unique exam identifier	Stores exam details: title, description, date, max score
title		VARCHAR(255)	Exam title	
description		TEXT	Exam description	
date		TIMESTAMP	Exam date/time	
max_score		INTEGER	Maximum possible score	

4.3 Table: questions

Column	**PK/FK**	**Data Type**	**Description / Notes**	**Purpose / Role**
question_id	PK	UUID	Unique question identifier	Stores exam questions linked to exams, with options, correct answer, atomic ideas for grading
exam_id	FK → exams.exam_id	UUID	Parent exam reference	
text		TEXT	Question text	

options		JSONB	Options / multiple-choice data	
correct_answer		JSONB	Correct answer(s)	
atomic_ideas		JSONB	Breakdown for grading / rubrics	

4.4 Table: submissions

Column	PK/FK	Data Type	Description / Notes	Purpose / Role
submission_id	PK	UUID	Unique submission identifier	Stores each student submission: type, payload, metadata, current workflow state
student_id	FK → students.student_id	UUID	Student who submitted	
exam_id	FK → exams.exam_id	UUID	Related exam	
submission_type		VARCHAR(10)	'text'	'image'
payload		JSONB	Text or file URL	
metadata		JSONB	Timestamps, device info, IP, attempt number	

created_at		TIMESTAMP	Submission timestamp	
current_state		VARCHAR(50)	Current state in state machine	

4.5 Table: grading_results

Column	PK/FK	Data Type	Description / Notes	Purpose / Role
grading_id	PK	UUID	Unique grading record	Stores AI grading output per submission: grade, deductions, atomic idea matches, reasoning
submission_id	FK → submissions.submission_id	UUID	Related submission	
grade		NUMERIC(5, 2)	Assigned grade	
max_grade		NUMERIC(5, 2)	Maximum possible	
atomic_idea_matches		JSONB	Matches with confidence per idea	
deductions		JSONB	Points deducted	

			and reasons	
raw_reasoning		TEXT	AI reasoning / chain-of-thought	
grading_time_ms		INTEGER	Processing time in ms	

4.6 Table: hitl_reviews

Column	PK/FK	Data Type	Description / Notes	Purpose / Role
hitl_id	PK	UUID	Manual review record	Stores human-in-the-loop interventions for flagged submissions
submission_id	FK → submissions.submission_id	UUID	Submission reviewed	
human_reviewer_id	FK → students.student_id	UUID	Reviewer ID (or HR table)	
final_grade		NUMERIC(5,2)	Manual adjusted grade	
override_reason		TEXT	Reason for override	

resolved		BOOLEAN	True if review completed	

4.7 Table: model_versions

Column	PK/FK	Data Type	Description / Notes	Purpose / Role
model_id	PK	UUID	Unique model record	Stores deployed versions of OCR/NLP/LLM models with metadata
model_name		VARCHAR(100)	LLM / OCR / NLP name	
model_type		VARCHAR(20)	'OCR', 'NLP', 'LLM'	
version		VARCHAR(20)	Version string	
deployed_at		TIMESTAMP	Deployment timestamp	

4.8 Table: audit_logs

Column	PK/FK	Data Type	Description / Notes	Purpose / Role
log_id	PK	UUID	Unique log record	Tracks system events, state changes, and security alerts

submission_id	FK → submissions.submission_id (optional)	UUID	Related submission if any	
event_type		VARCHAR(50)	Type of event	
timestamp		TIMESTAMP	Event timestamp	
description		TEXT	Details of the event	

4.9 Table: feedback

Column	**PK/FK**	**Data Type**	**Description / Notes**	**Purpose / Role**
feedback_id	PK	UUID	Feedback record	Stores pedagogical feedback per submission: summary, strengths, weaknesses, recommended topics
submission_id	FK → submissions.submission_id	UUID	Related submission	
summary		TEXT	Overall feedback summary	
strengths		JSONB	Array of positive points	

weaknesses		JSONB	Array of weaknesses	
recommended_topics		JSONB	Topics for further study	

4.10 Table: profile_db

Column	PK/FK	Data Type	Description / Notes	Purpose / Role
profile_id	PK	UUID	Unique student profile record	Stores historical student performance summary for adaptive exam generation (metadata only)
student_id	FK → students.student_id	UUID	Linked student	
historical_vectors		JSONB	Performance / embedding history (metadata only)	
last_updated		TIMESTAMP	Last update timestamp	

4.11 Table: adaptive_exams

Column	PK/FK	Data Type	Description / Notes	Purpose / Role

generated_exam_id	PK	UUID	Generated exam record	Stores exams generated per student based on weaknesses and performance history
student_id	FK → students.student_id	UUID	Student for whom exam is generated	
weakness_vector		JSONB	Weakness vector (metadata only, not embeddings)	
recommended_questions		JSONB	List of questions + difficulty	
created_at		TIMESTAMP	Generation timestamp	

5.0 Common Envelope

5.1 Submission Contract (Input)

```
{
 "submission_id": "uuid",
 "student_id": "uuid",
 "exam_id": "uuid",
 "submission_type": "text | image | pdf",
            "payload": {
              "text": "string (if text)",
              "file_url": "string (if image/pdf)"
            },
  "metadata": {
    "timestamp": "ISO-8601",
    "device_info": "string",
    "ip_address": "string",
    "language": "en | ar",
    "attempt_number": 1
  }
}
```

5.2 Submission Acknowledgment

```
{
  "submission_id": "uuid",
  "state": "CREATED",
  "estimated_processing_time_sec": 15
}
```

5.3 OCR Stage

```
{
  "submission_id": "uuid",
  "state": "OCR_PROCESSING",
  "ocr_engine": "TrOCR | Tesseract",
  "ocr_version": "v1.3",
 "extracted_text": "string",
 "confidence_score": 0.92,
 "layout_blocks": [
  {
   "block_id": "uuid",
   "text": "string",
   "bbox": [x1, y1, x2, y2],
   "confidence": 0.88
  }
 ],
 "processing_time_ms": 540
}
```

5.4 Token Quality Check

```
{
 "submission_id": "uuid",
 "state": "TEXT_VALIDATION",
 "semantic_score": 0.85,
 "entropy_score": 0.72,
 "gibberish_score": 0.05,
 "final_quality_score": 0.81,
 "flag_for_hitl": false
}
```

5.5 Security & Guardrails

```
{
 "submission_id": "uuid",
 "state": "SECURITY_CHECK",
 "prompt_injection_detected": false,
 "adversarial_pattern_detected": false,
 "threat_level": "low | medium | high",
 "action": "allow | hitl | reject"
}
```

5.6 Semantic Cache Layer

```
{
 "submission_id": "uuid",
 "state": "CACHE_CHECK",
 "embedding_model_version": "bge-v2",
 "similarity_score": 0.91,
 "cache_hit": true,
 "matched_submission_id": "uuid",
 "action": "bypass_llm | send_to_grading"
}
```

5.7 Grading Engine Contract

```
{
 "submission_id": "uuid",
 "state": "GRADING",
 "model_name": "llama-3-70b",
 "model_version": "v2.1",
 "prompt_version": "rubric_v5",
 "grade": 8,
 "max_grade": 10,
 "atomic_idea_matches": [
  {
    "idea_id": "uuid",
    "matched": true,
    "confidence": 0.87,
    "deduction": 0
  }
 ],
 "deductions": [
  {
    "reason": "Missing explanation of step 2",
    "points": 2
  }
 ],
 "raw_reasoning": "string",
 "grading_time_ms": 2100
}
```

5.8 Trust Calibration Layer

```
{
 "submission_id": "uuid",
 "state": "CONFIDENCE_EVAL",
 "ocr_confidence": 0.92,
 "semantic_confidence": 0.81,
 "grading_confidence": 0.84,
 "rag_consistency_score": 0.88,
 "final_confidence_score": 0.86,
 "routing_decision": "auto_approve | qa_agent | hitl"
}
```

5.9 Judgmental Agent Contract

```
{
  "submission_id": "uuid",
  "state": "QA_AUDIT",
  "hallucination_detected": false,
  "rubric_conflict": false,
  "retrieved_reference_ids": ["uuid"],
  "qa_confidence": 0.90,
  "qa_flag": false
}
```

5.10 HITL Contract

```
{
  "submission_id": "uuid",
  "state": "HITL_REVIEW",
  "human_reviewer_id": "uuid",
  "final_grade": 9,
  "override_reason": "AI missed minor explanation",
  "resolved": true
}
```

5.11 Final Output (Student Response)

```
{
  "submission_id": "uuid",
  "state": "FINALIZED",
  "grade": 8,
  "max_grade": 10,
  "confidence_score": 0.86,
  "feedback": {
    "summary": "Good answer but missing one key step.",
    "strengths": ["Clear introduction"],
    "weaknesses": ["Step 2 incomplete"],
    "recommended_topics": ["Topic A", "Topic B"]
  },
  "processing_summary": {
    "total_latency_ms": 3200,
    "cache_used": true,
    "hitl_involved": false
  }
}
```

5.12 Adaptive Exam Generator Contract

```
{
  "student_id": "uuid",
  "weakness_vector": [0.2, 0.8, 0.1],
  "recommended_questions": [
    {
      "question_id": "uuid",
      "difficulty": "medium",
      "topic": "Topic B"
    }
  ],
  "generated_exam_id": "uuid"
}
```

6.0 Project Tree

```
ExamAI/
│
├── app/
│   ├── api/
│   │   ├── v1/
│   │   │   ├── endpoints/
│   │   │   │   ├── submissions.py
│   │   │   │   ├── exams.py
│   │   │   │   ├── questions.py
│   │   │   │   └── students.py
│   │   │   └── dependencies.py
│   │   └── main.py
│   │
│   ├── core/
│   │   ├── config.py
│   │   ├── security.py
│   │   ├── utils.py
│   │   └── logging.py
│   │
│   ├── services/
│   │   ├── grading/
│   │   │   ├── engine.py
│   │   │   ├── atomic_idea_checker.py
│   │   │   └── deductions.py
│   │   ├── feedback/
│   │   │   ├── generator.py
│   │   │   └── profile_db_manager.py
│   │   ├── ocr/
│   │   │   ├── preprocess.py
│   │   │   ├── layout_analysis.py
│   │   │   └── multi_ocr.py
│   │   ├── nlp/
│   │   │   ├── token_quality.py
│   │   │   ├── text_preprocessing.py
│   │   │   └── lemmatizer.py
│   │   ├── security/
│   │   │   ├── guardrails.py
│   │   │   └── semantic_cache.py
│   │   └── adaptive_exam/
│   │       ├── exam_generator.py
│   │       └── weakness_analysis.py
│   │
│   ├── db/
│   │   ├── models/
│   │   │   ├── students.py
│   │   │   ├── exams.py
│   │   │   ├── questions.py
│   │   │   ├── submissions.py
│   │   │   ├── grading_results.py
│   │   │   ├── hitl_reviews.py
│   │   │   ├── model_versions.py
│   │   │   ├── audit_logs.py
│   │   │   ├── feedback.py
│   │   │   ├── profile_db.py │
│   │   └── adaptive_exams.py
│   │   └── session.py
│   │
│   ├── workers/
│   │   ├── celery_app.py
│   │   ├── tasks/
│   │   │   ├── grading_tasks.py
│   │   │   ├── ocr_tasks.py
│   │   │   ├── feedback_tasks.py
│   │   │   └── nlp_tasks.py
│   │   └── hitl_tasks.py
│   │
│   └── schemas/
│       ├── submissions.py
│       ├── exams.py
│       ├── questions.py
│         ├── students.py
│         ├── feedback.py
│         └── nlp.py
│
├── tests/
│    ├── unit/
│    │    ├── test_grading_engine.py
│    │    ├── test_ocr.py
│    │    ├── test_token_quality.py
│    │    ├── test_text_preprocessing.py
│    │    ├── test_lemmatizer.py
│    │    ├── test_feedback.py
│    │    ├── test_security.py
│    │    └── test_adaptive_exam.py
│    └── integration/
│         ├── test_api_endpoints.py
│         ├── test_full_flow.py
│         ├── test_ocr_nlp_pipeline.py
│         └── test_grading_feedback_pipeline.py
│
├── scripts/
│    ├── db_init.py
│    ├── data_migration.py
│    └── seed_data.py
│
├── docker/
│    ├── Dockerfile
│    └── docker-compose.yml
│
├── requirements.txt
├── README.md
└── .env
```