Software Design and Analysis (SE-2002)

PROJECT

Software: AnkiDroid

Group#10

Sana Idrees(23i-2039), Manahil(23i-3000),

Maryam Fatima(23i-3007) and Ayesha Khan(23i-3037)

BS-Software Engineering(4B)

Submitted to: Dr. Behjat Zuhaira

Client: Group # 1

1. Weakness-Based Recommendation Engine

Step 1: Domain Analysis

Goal: Quantify and predict learners' knowledge gaps.

Actions:

- **Extend** `Learner` **class:**

```
class Learner {
  +Map<String, Float> tagWeaknessScores;  // {"verbs": 0.87, "tenses": 0.45}
  +List<ReviewRecord> reviewHistory;     // [{card_id: "123", correct: false, response_time: 2.3}]
        }
```

- **Define weakness metrics:**
    - Error rate: Tag is weak if correct response rate <50% over last 20 reviews.
    - Response time: Tag is weak if average response time >2x deck average.
    - Postponement rate: Tag is weak if skipped in >30% of sessions.

Output:

- **Documented schema:**

  Weakness = 0.6*(error_rate) + 0.3*(normalized_response_time) + 0.1*(postponement_rate)

Step 2: Dataset Collection

Schema:

Field	Type	Source Class	Example
learner_id	String	Learner	"user_123"
card_tags	List<String>	Flashcard	["verbs", "past_tense"]
response_time	Float	ReviewSession	2.34 (sec)
correct	Boolean	ProgressTracker	false
timestamp	DateTime	ReviewSession	2025-05-11T14:30:00Z

Integration:

- Observer Pattern: **ProgressTracker** subscribes to **ReviewSession** events.

Step 3: Data Cleaning & Preprocessing

Pipeline:

1. **Handle missing data:**
    - Impute **card_tags** with "untagged".
    - Fill null **response_time** with deck median.

2. **Normalization:**
    - Scale **response_time** to **[0, 1]** per deck:

```
normalized_time = (time - deck_min) / (deck_max - deck_min)
```

3. **Outlier removal:**
    - Discard reviews with **response_time > 30s** (deemed non serious attempts).

Step 4: Model Architecture

LSTM Network:

- **Input:** Sequence of last 10 reviews (features: [**tags, response_time, correct]**).
- **Layers:**

```
model = Sequential([
LSTM(64, input_shape=(10, 3)),  # 10 timesteps, 3 features
Dense(32, activation='relu'),
Dense(len(tags), activation='sigmoid')  # Output per-tag probabilities
    ])
```

- **Output:** Weakness probabilities (e.g., **{"verbs": 0.92, "tenses": 0.45}**).

Why LSTM?

- Models temporal dependencies (e.g., forgetting curves).
- Handles variable-length sequences (pad/review counts may vary).

Step 5: Model Training

Process:

1. **Split:** 80% training, 20% validation (stratified by learner).
2. **Training:**
    - Optimizer: **Adam** (learning rate: 0.001).
    - Loss: **BinaryCrossentropy** (per-tag weakness is binary).
3. **Metrics:**
    - Hit Rate @5 (HR@5): % of true weak tags in top-5 predictions.
    - AUC-ROC: Measures ranking quality.

Validation:

- Early stopping if validation HR@5 doesn't improve for 5 epochs.

Step 6: Evaluation & Deployment

Thresholds:

- **Deploy if:**
    - HR@5 ≥ 75% (minimum useful recommendation quality).
    - AUC-ROC ≥ 0.85 (discriminative power).
- **Fallback:**
    - Use Anki's SM-2 scheduler if model confidence <80%.

Monitoring:

- Log HR@5 daily via **ProgressTracker**.
- Retrain monthly with new data.

---

2. Battle Opponent Matchmaker

Step 1: Domain Analysis

Goal: Define "good" battle matches.

Criteria:

1. **Deck overlap ≥60%**: Ensures common ground.
2. **Weakness similarity (cosine ≥0.7)**: Comparable skill gaps.
3. **Rank delta ≤50**: Avoids mismatches (e.g., rank 10 vs. rank 1000).

Output:

- Match quality formula:

  ```
  match_score = 0.5*cosine_sim + 0.3*deck_overlap + 0.2*(1 - rank_delta_norm)
  ```

Step 2: Dataset Collection

Schema:

Field	Type	Source Class	Example
learner_id	String	Learner	"user_123"
deck_ids	List<String>	Deck	["deck_1", "deck_2"]
weakness_scores	Map<String, Float>	WeaknessPredictor	{"verbs": 0.8}
rank	Integer	Leaderboard	150

Integration:

- **Singleton Access**: **BattleMatchmaker** queries **Leaderboard** and **DeckManager.**
- **Privacy**: Anonymize **learner_id** before clustering.

Step 3: Data Cleaning & Preprocessing

Steps:

1. **Impute ranks**: New learners get **rank = 1000** (global median).
2. **Normalize ranks**: Min-max scale to **[0, 1]**:

   ```
   normalized_rank = (rank - min_rank) / (max_rank - min_rank)
   ```

3. **Filter inactive learners**: Last battle >30 days ago.

Step 4: Model Architecture

k-NN Clustering:

- **Features**:
    1. **Cosine similarity** of **weakness_scores** vectors.
    2. **Deck overlap %**: **(shared_decks / total_decks) * 100.**
- **Distance metric**:

  distance = 1 - (0.7*cosine_sim + 0.3*deck_overlap)

- **k=5**: Return top 5 candidate opponents.

Why k-NN?

- **Interpretable**: Matches are explainable ("80% deck overlap").
- **Low-latency**: Real-time inference suitable for battles.

Step 5: Model Training

Process:

- **Data**: 50,000 anonymized learner profiles.
- **Metric**: Battle completion rate (target >85%).
- **Validation**:
    - Synthetic edge cases (e.g., new learners with no history).

Step 6: Evaluation & Deployment

A/B Testing:

- **Group A (AI)**: 50% of battles use k-NN matches.
- **Group B (Control)**: 50% use random matches.
- **Compare**: Completion rates, post-battle ratings.

Fallback:

- **Random matching** if:
    - Model latency >500ms.
    - Fewer than 5 candidates found.

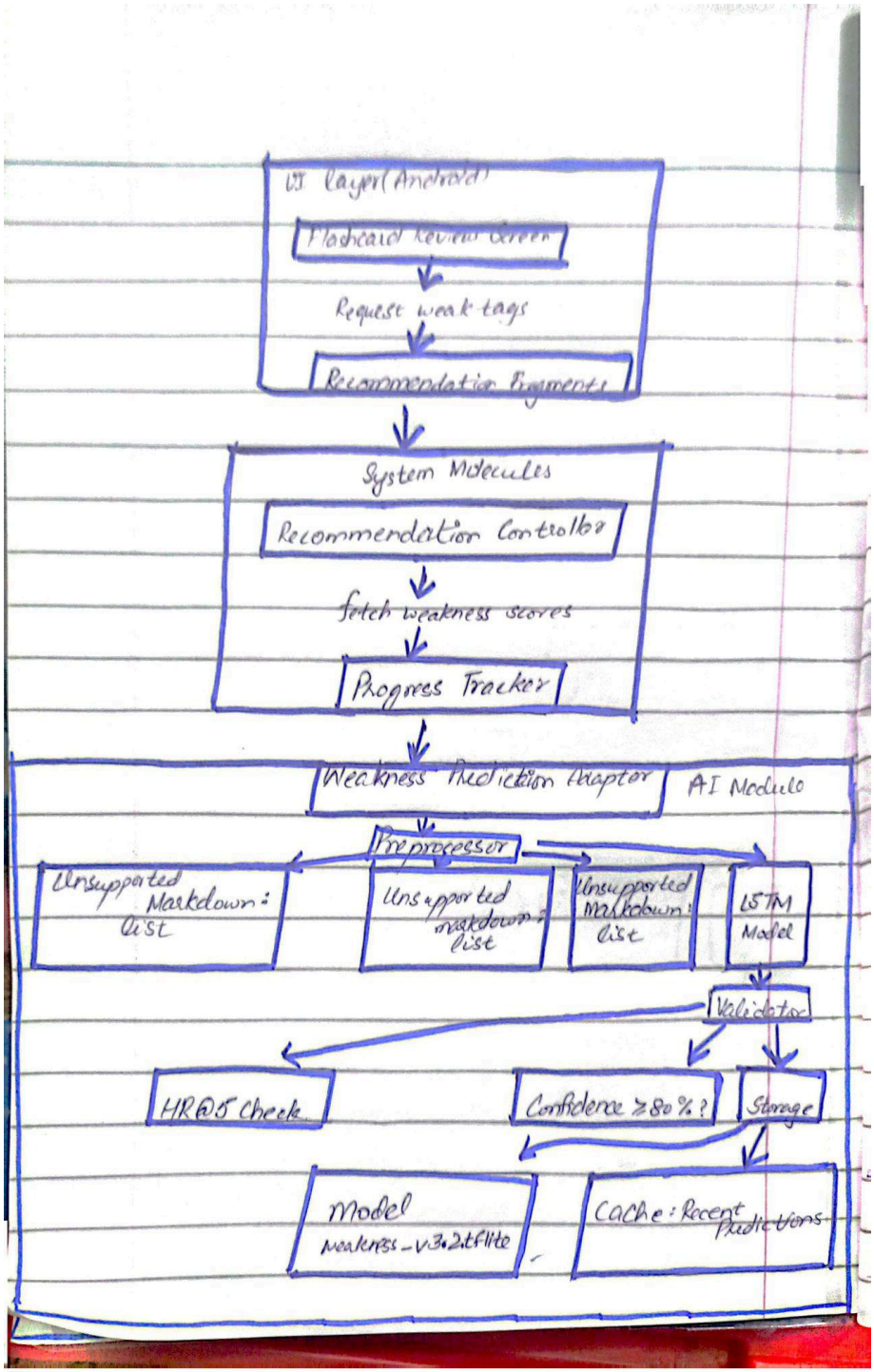
Monitoring:

- Track match quality via **Leaderboard** analytics.

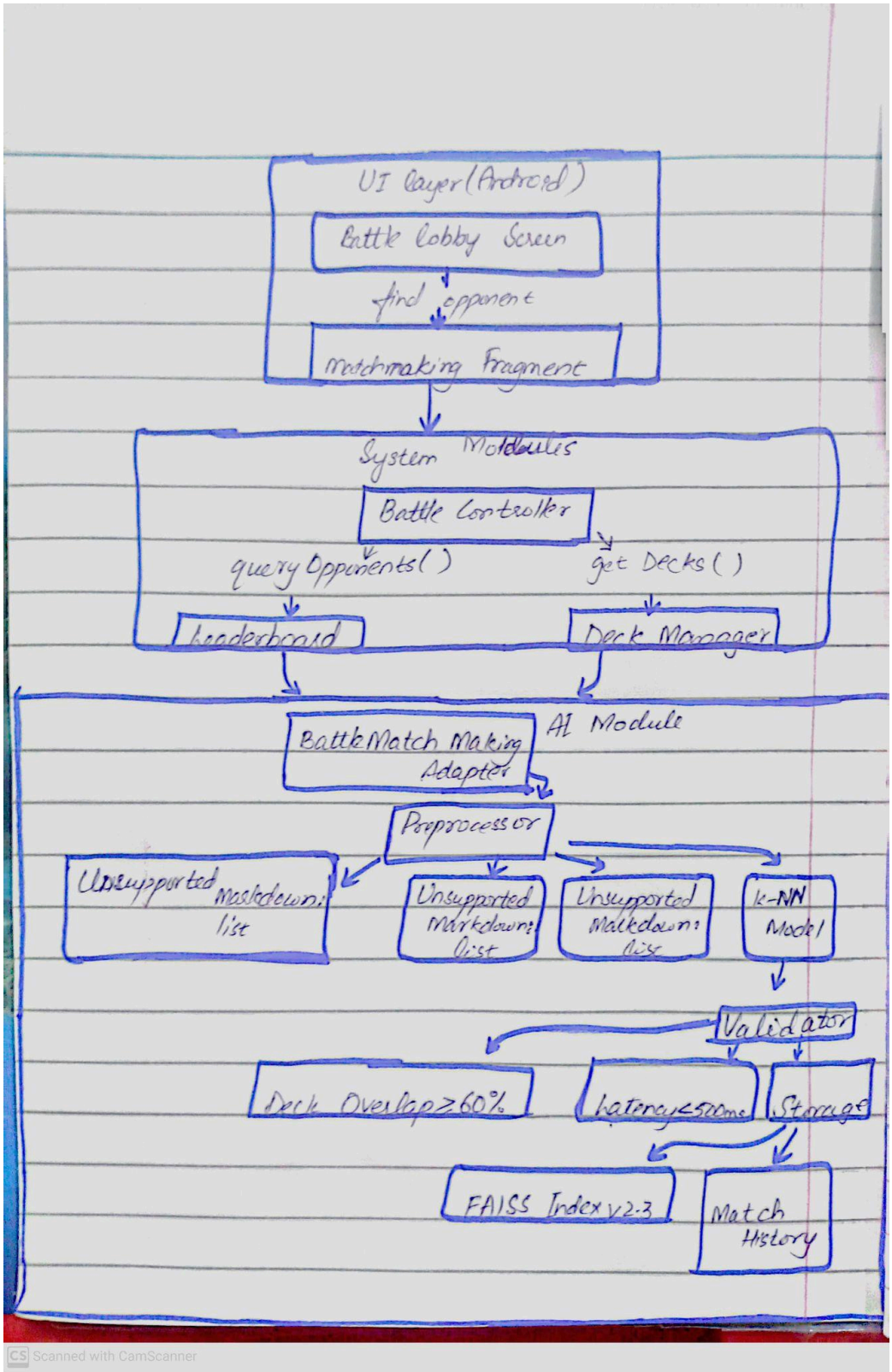
AI Design Diagrams

1. AI based top down view of system

Weakness-based recommendation engine

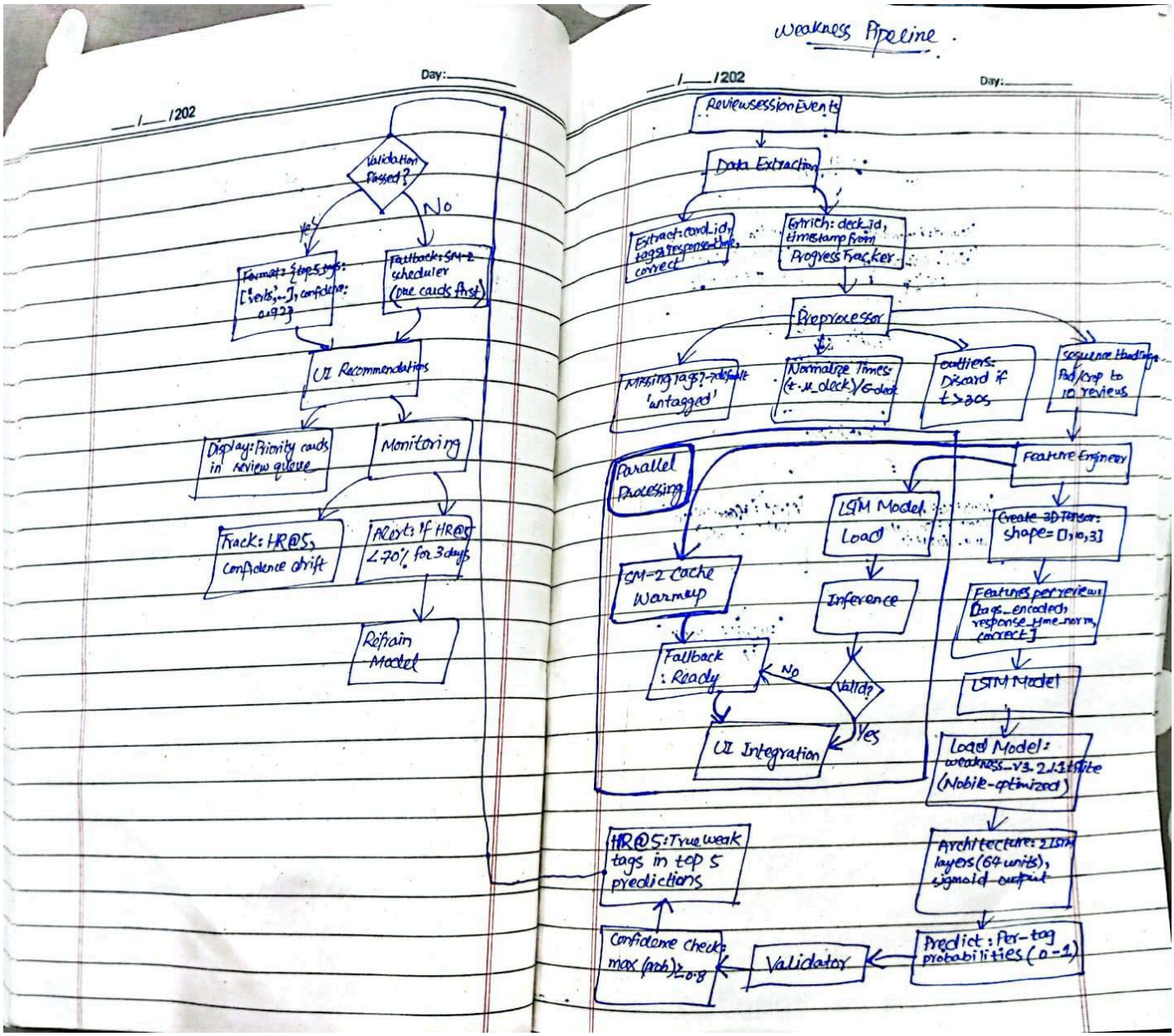


Battle opponent matchmaker

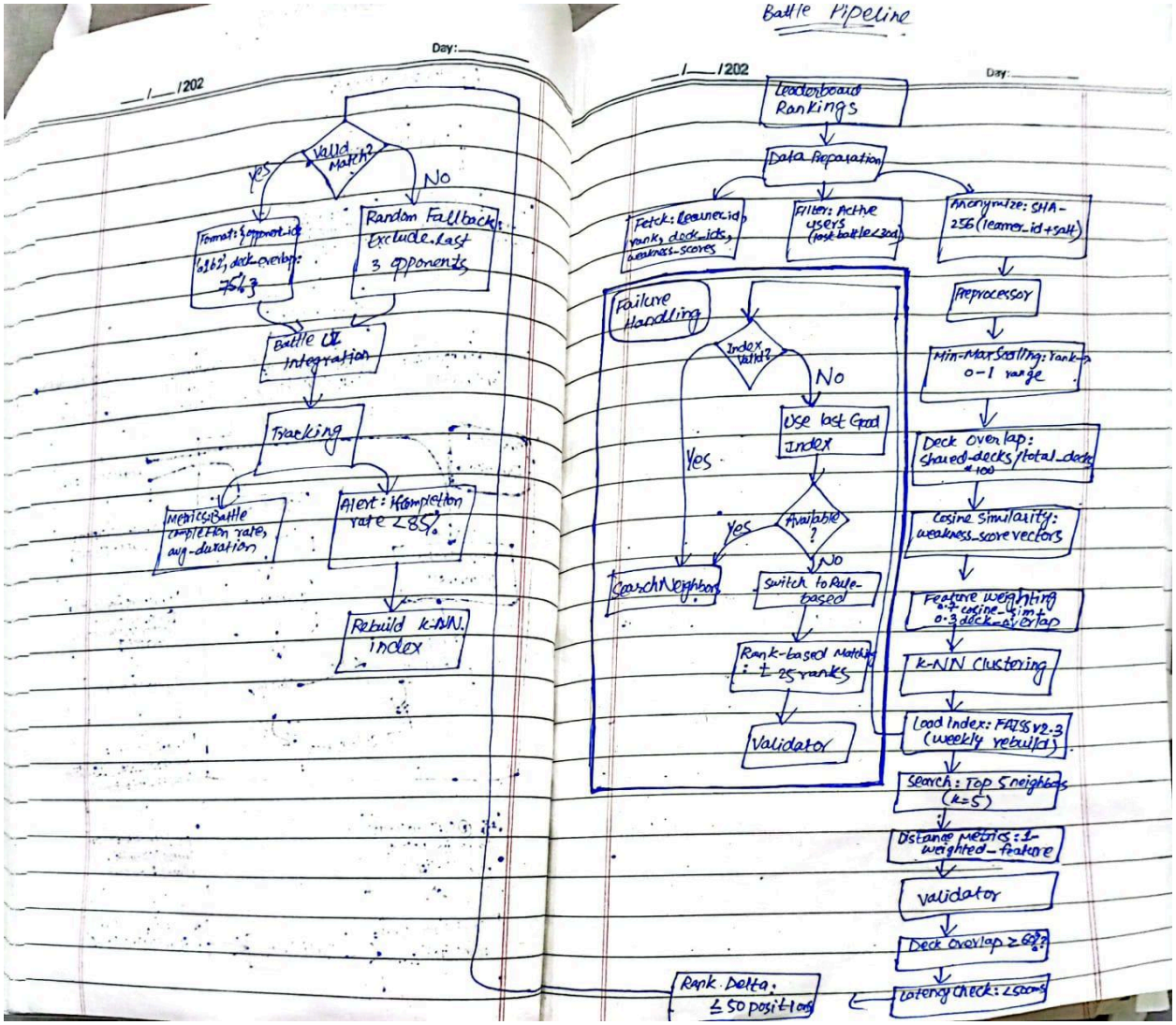


1. Inference flow(lifecycle/pipeline)

Weakness-based recommendation engine

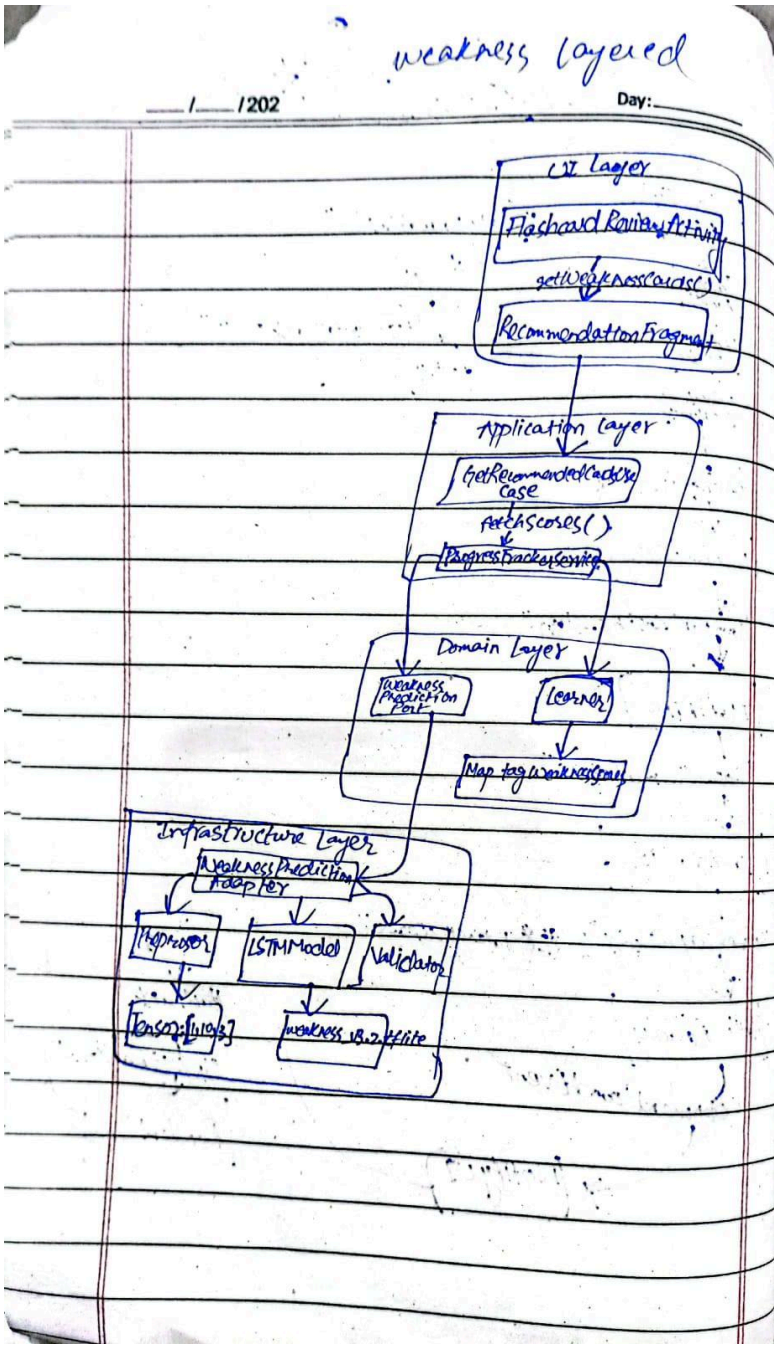


Battle opponent matchmaker

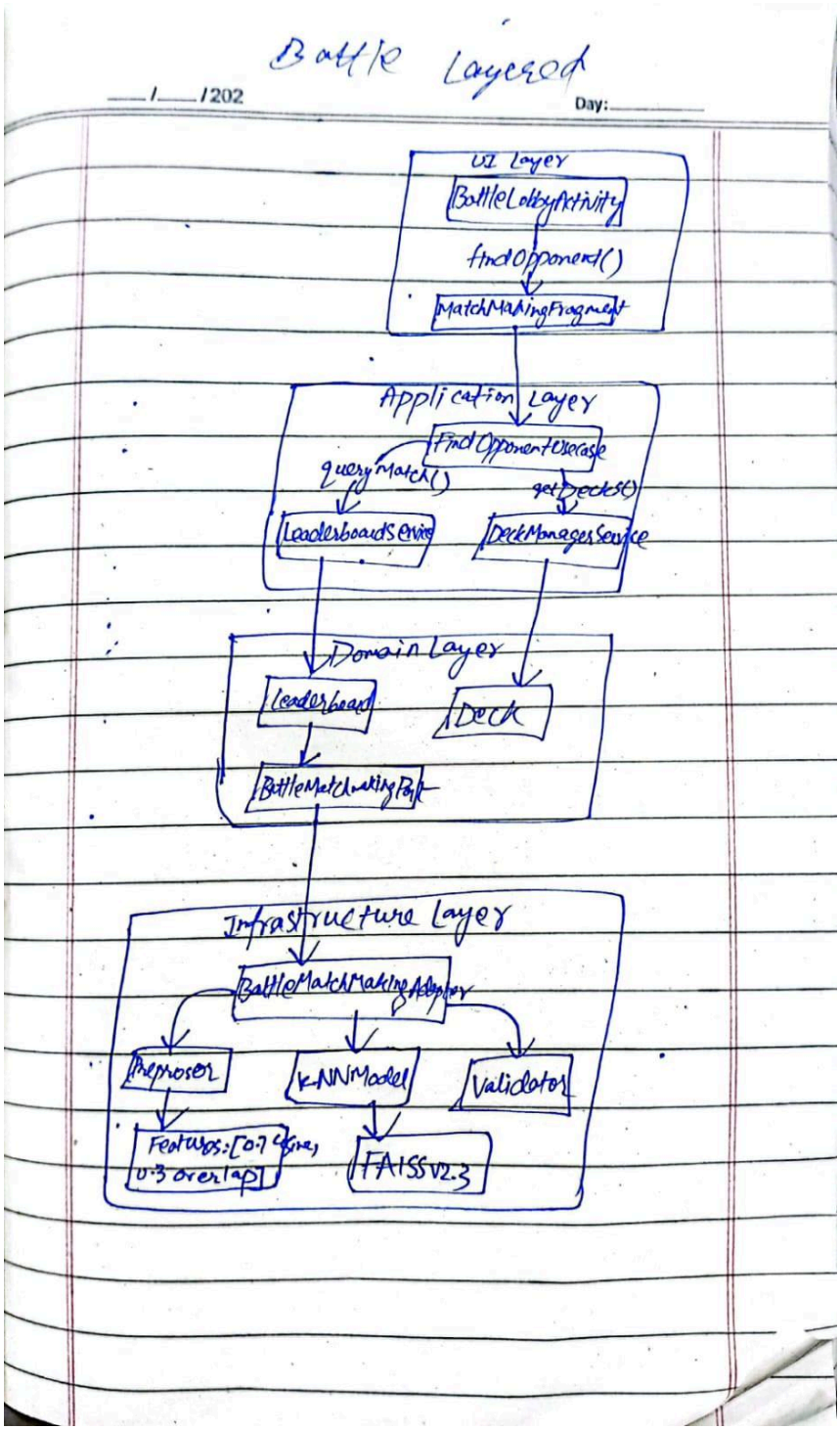


1. Layered Architecture with AI Black Box

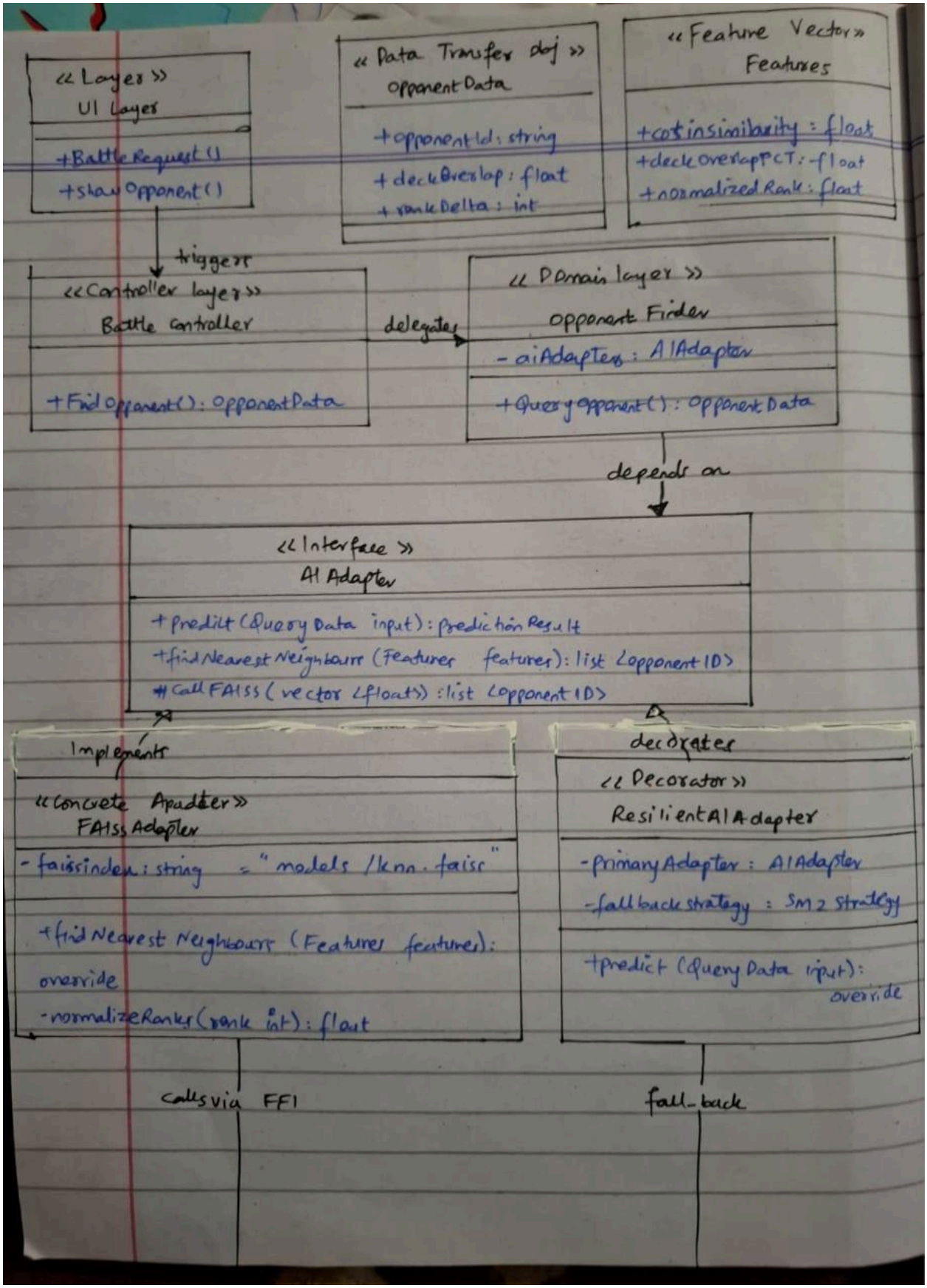
Weakness-based recommendation engine

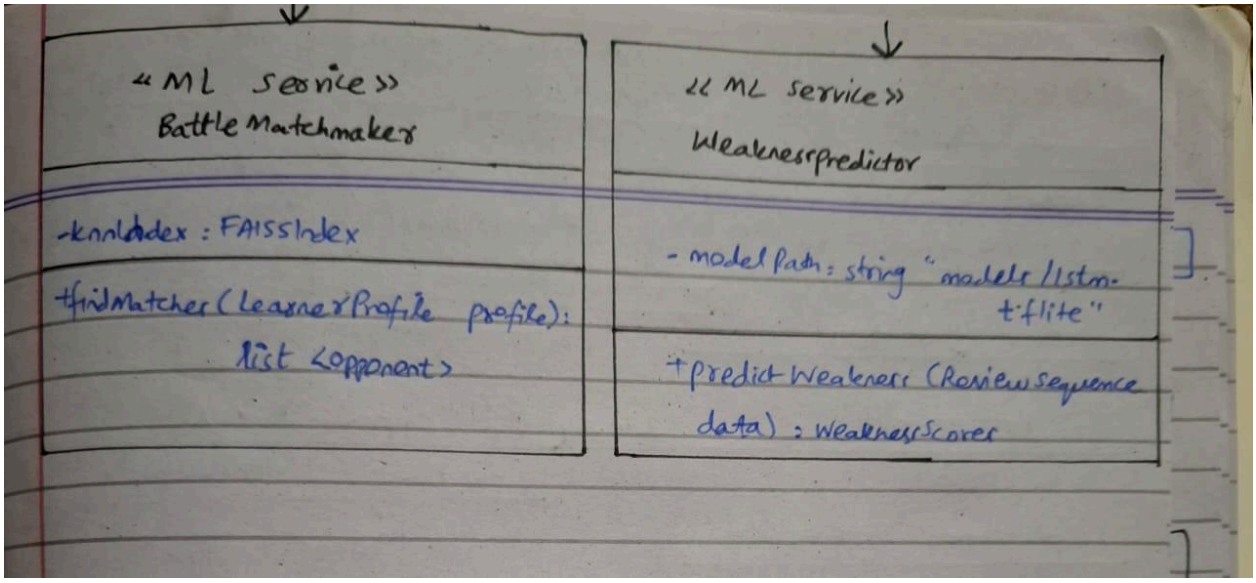


Battle opponent matchmaker



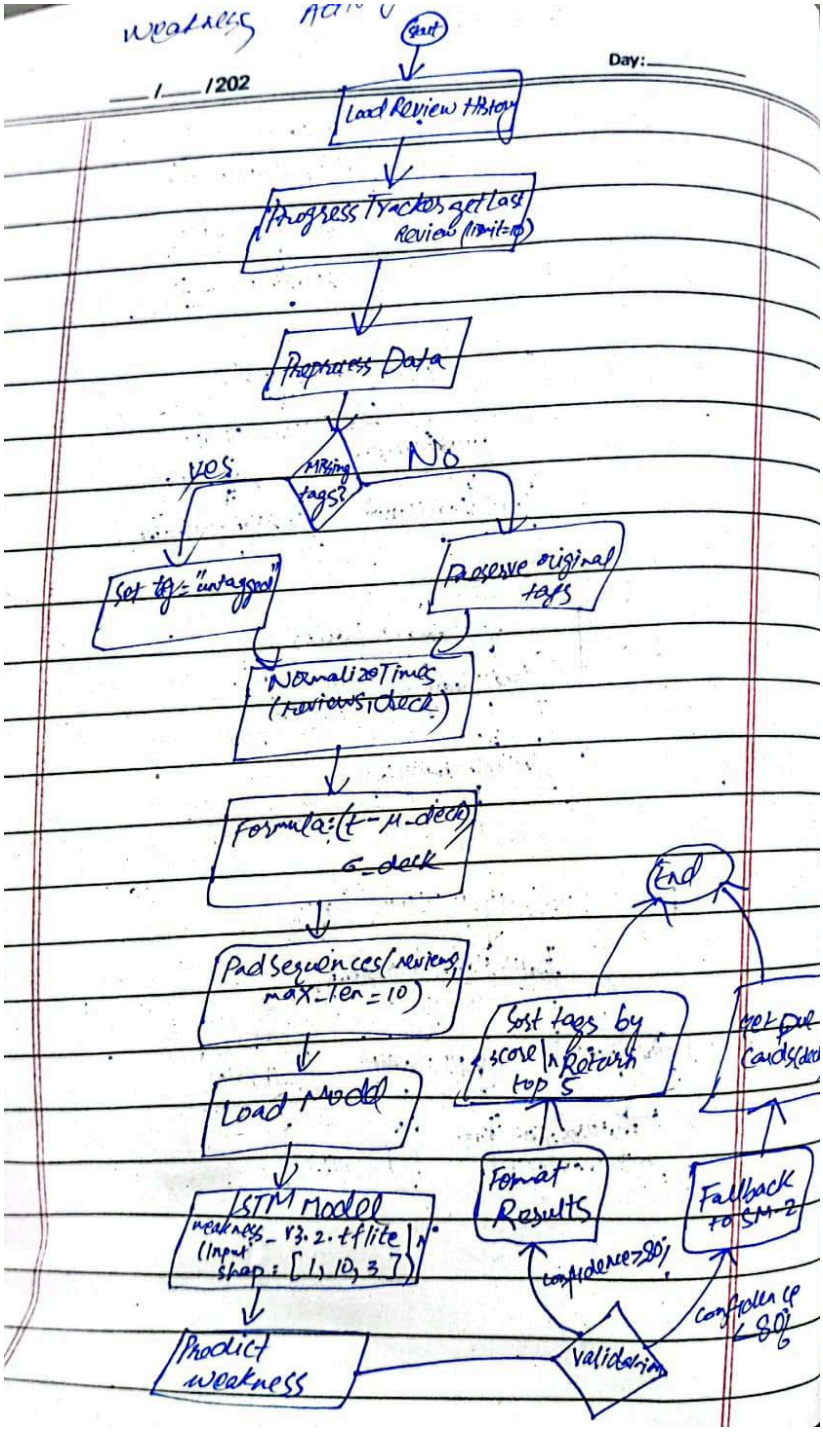
2. Class Diagram with ML Adapter


«Layer»
UI Layer
+BattleRequest()
+showOpponent()
trigger
«Data Transfer obj»
opponentData
+opponentId: string
+deckOverlap: float
+rankDelta: int
«Feature Vector»
Features
+cosinsimilarity: float
+deckOverlapPCT: float
+normalizedRank: float
«Controller layer»
Battle controller
+FindOpponent(): opponentData
delegates
«Domain layer»
opponent Finder
-aiAdapter: AIAdapter
+QueryOpponent(): opponentData
depends on
«Interface»
AI Adapter
+predict(QueryData input): predictionResult
+findNearestNeighbours(Features features): list<opponentID>
#callFAISS(vector<float>): list<opponentID>
Implements
«concrete Apadter»
FAISSAdapter
-faissIndex: string = "models/knn.faisc"
+findNearestNeighbours(Features features): override
-normalizeRanks(rank int): float
decorater
«Decorator»
ResilientAIAdapter
-primaryAdapter: AIAdapter
-fallbackstrategy: SM2 Strategy
+predict(QueryData input): override
calls via FFI
fall-back

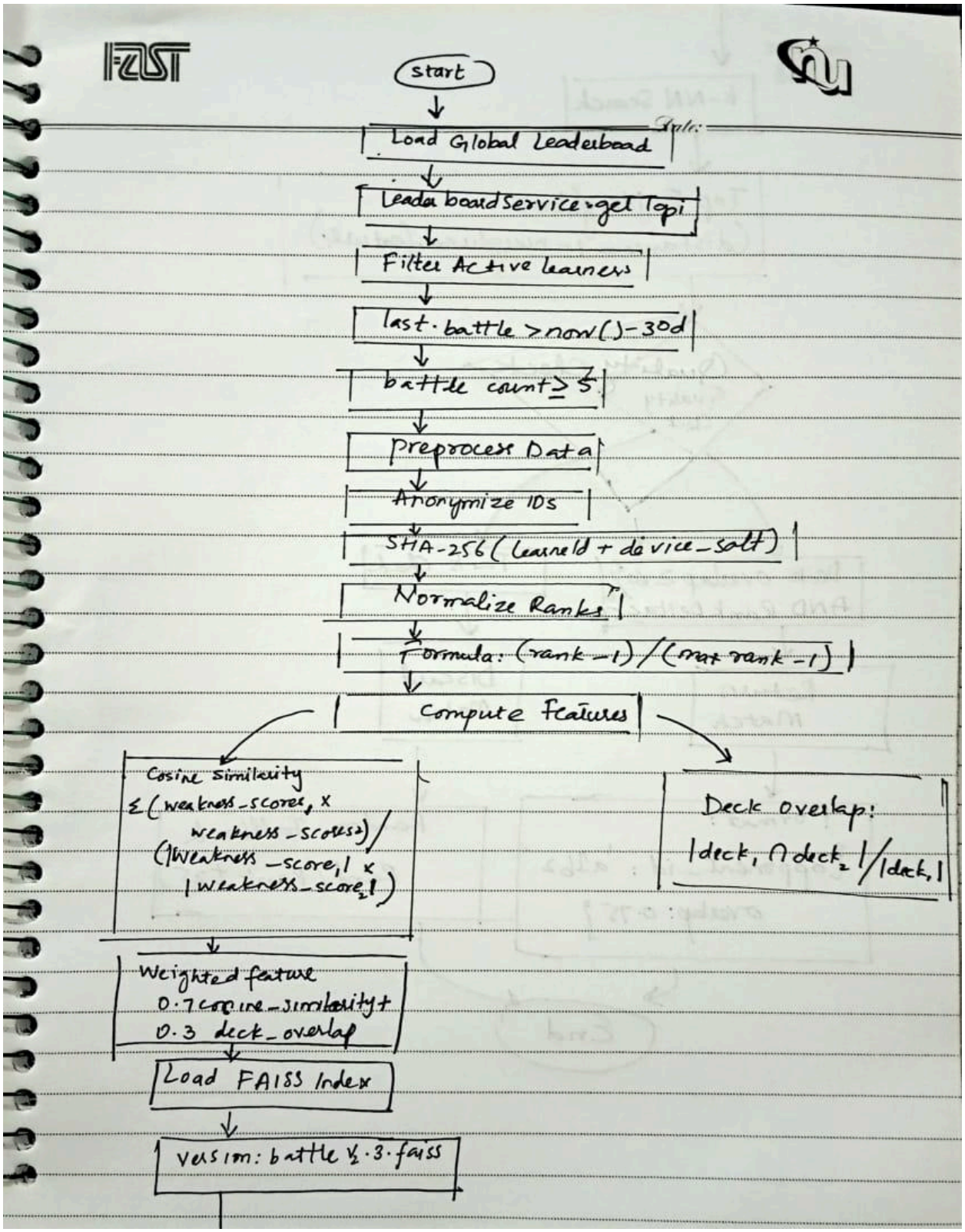

«ML service»
BattleMatchmaker
-knnIndex : FAISSIndex
+findMatches(LearnerProfile profile): list<Opponent>
«ML service»
WeaknessPredictor
- modelPath : string "models/lstm.tflite"
+predictWeaknesses(ReviewSequenceData) : WeaknessScores

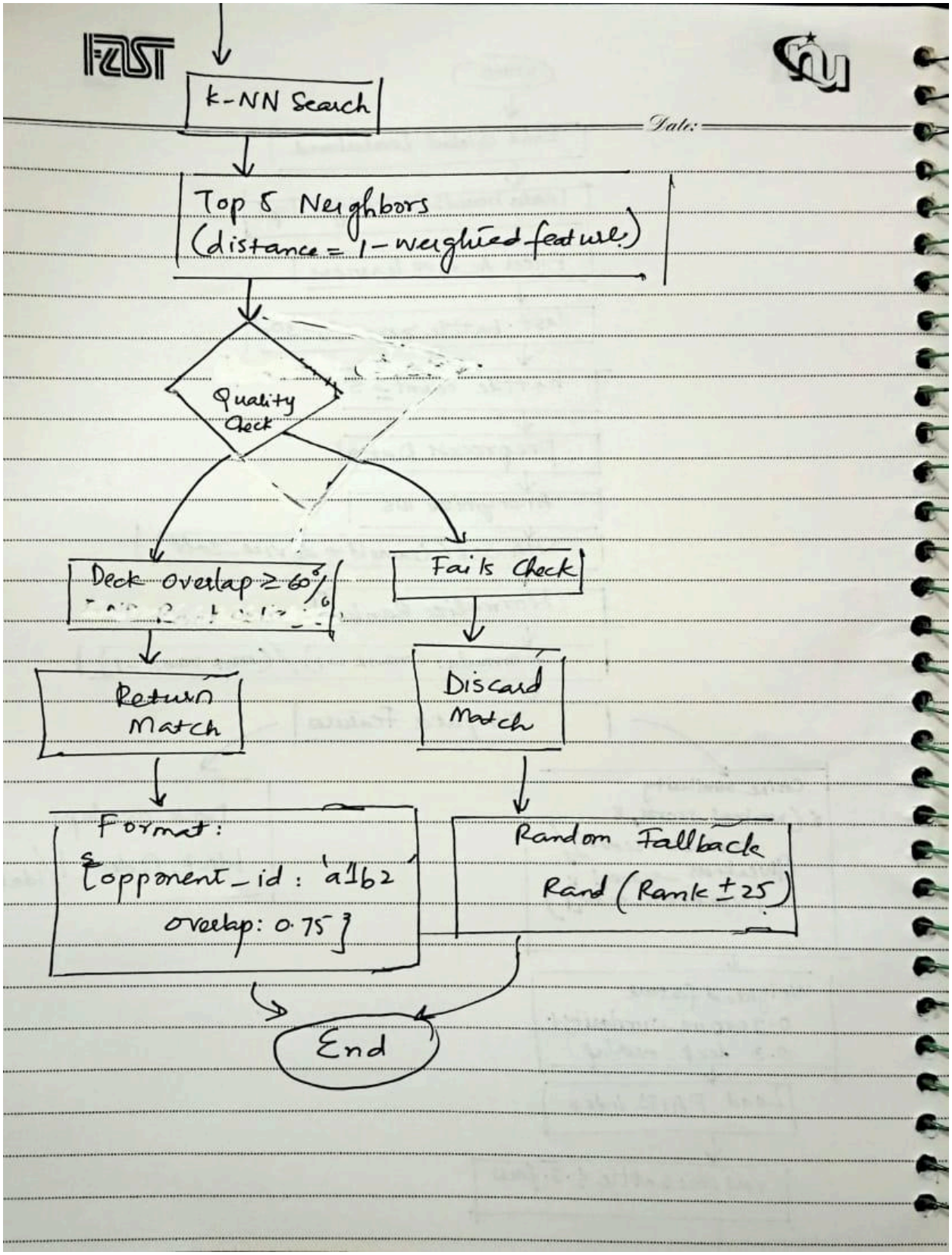
4. ML Pipeline Activity Diagram (Internal)

Weakness-based recommendation engine



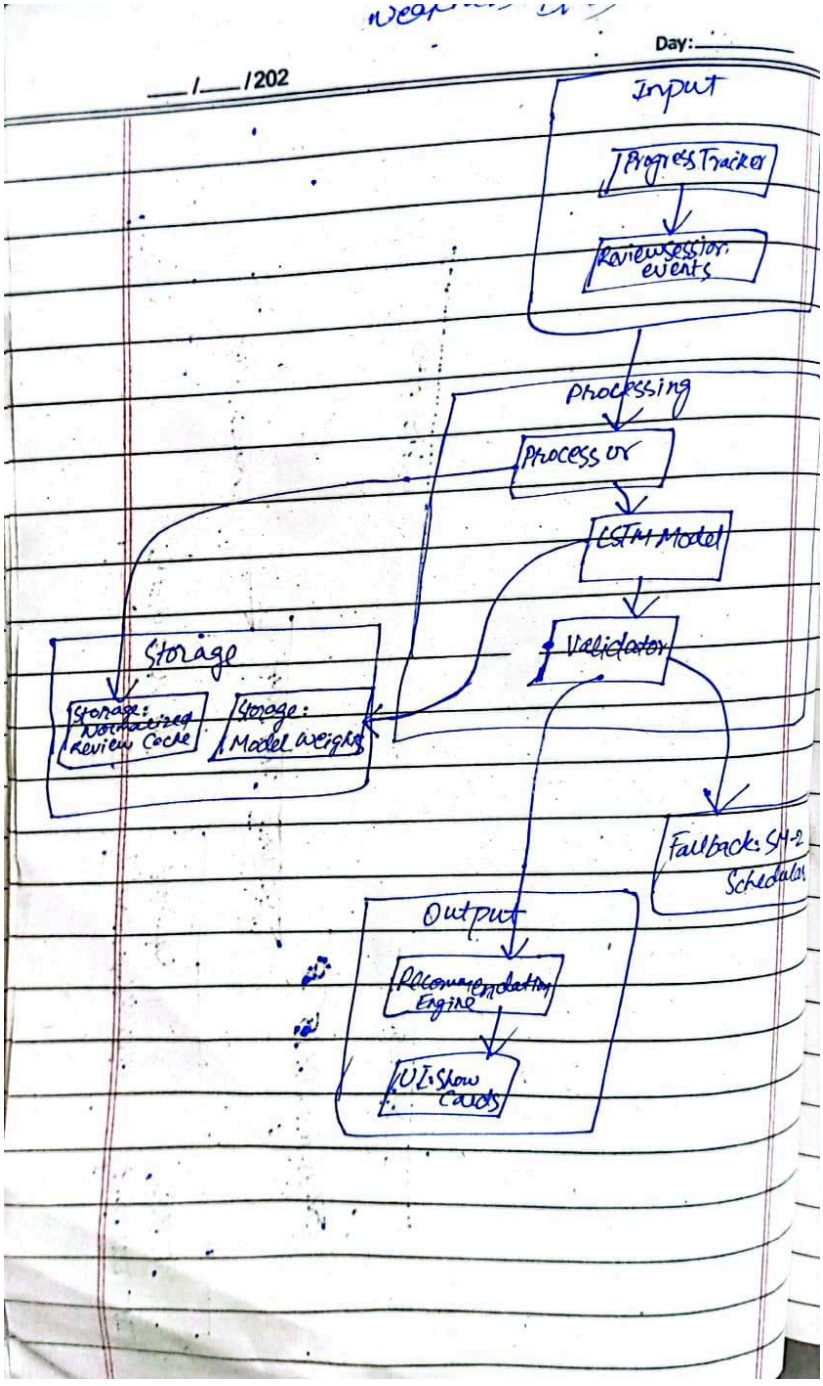
Battle opponent matchmaker



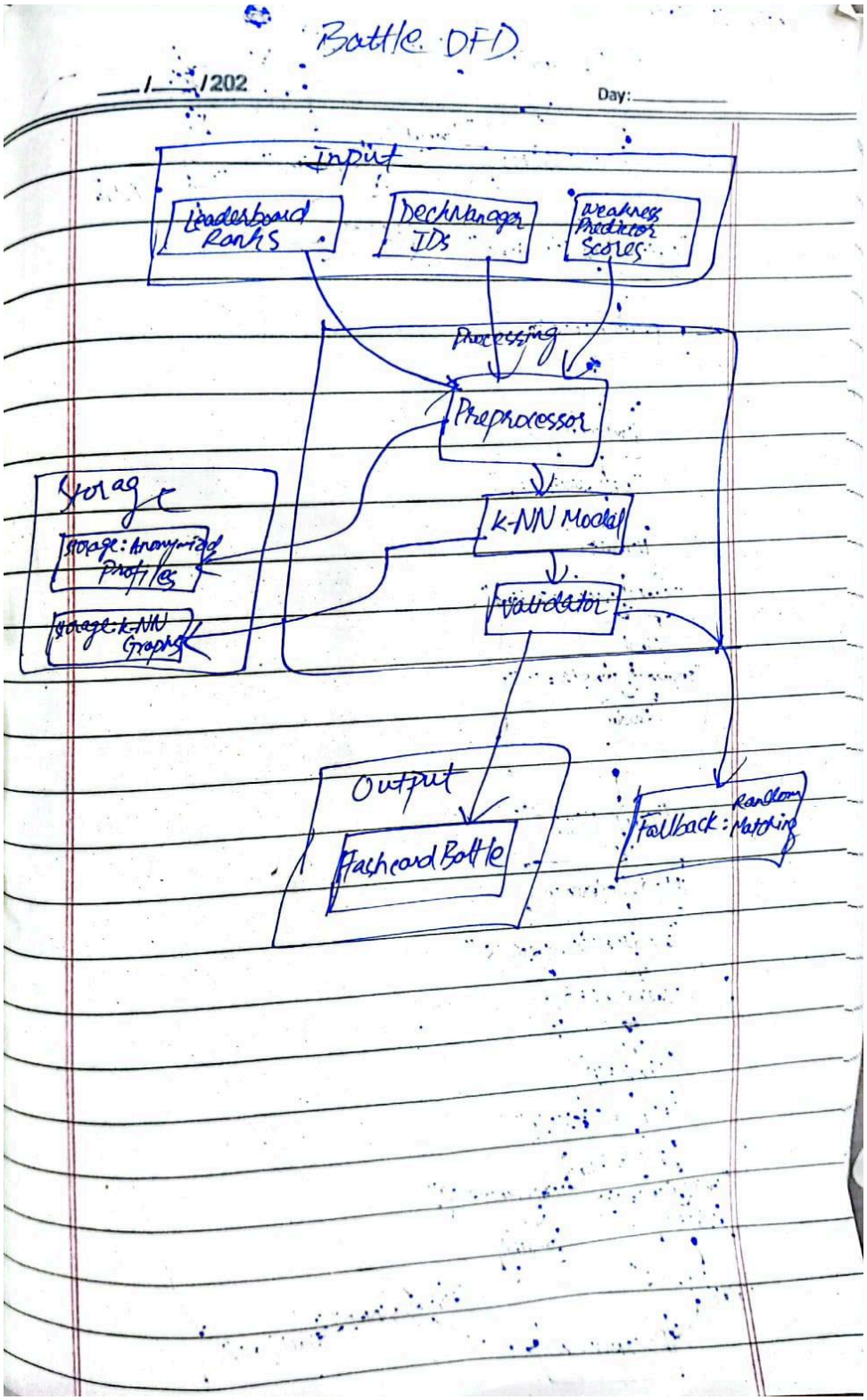

k-NN Search
Top 5 Neighbors
(distance = 1 - weighted feature)
Quality Check
Deck Overlap ≥ 60%
Fails Check
Return Match
Discard Match
Format:
{opponent_id: 'a1b2'
overlap: 0.75}
Random Fallback
Rand (Rank ± 25)
End

5. Data Flow Diagram (DFD)

Weakness-based recommendation engine

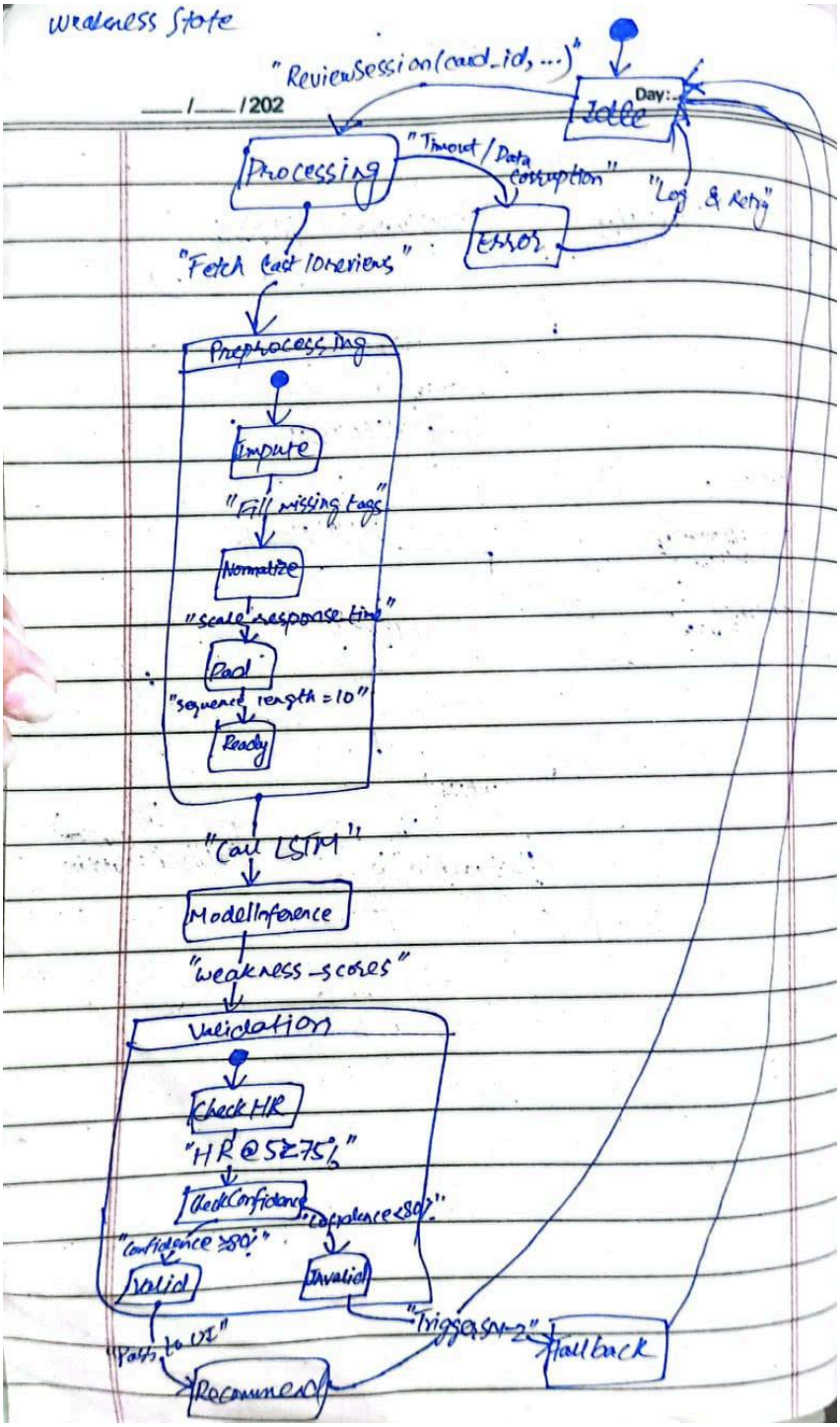


Battle opponent matchmaker

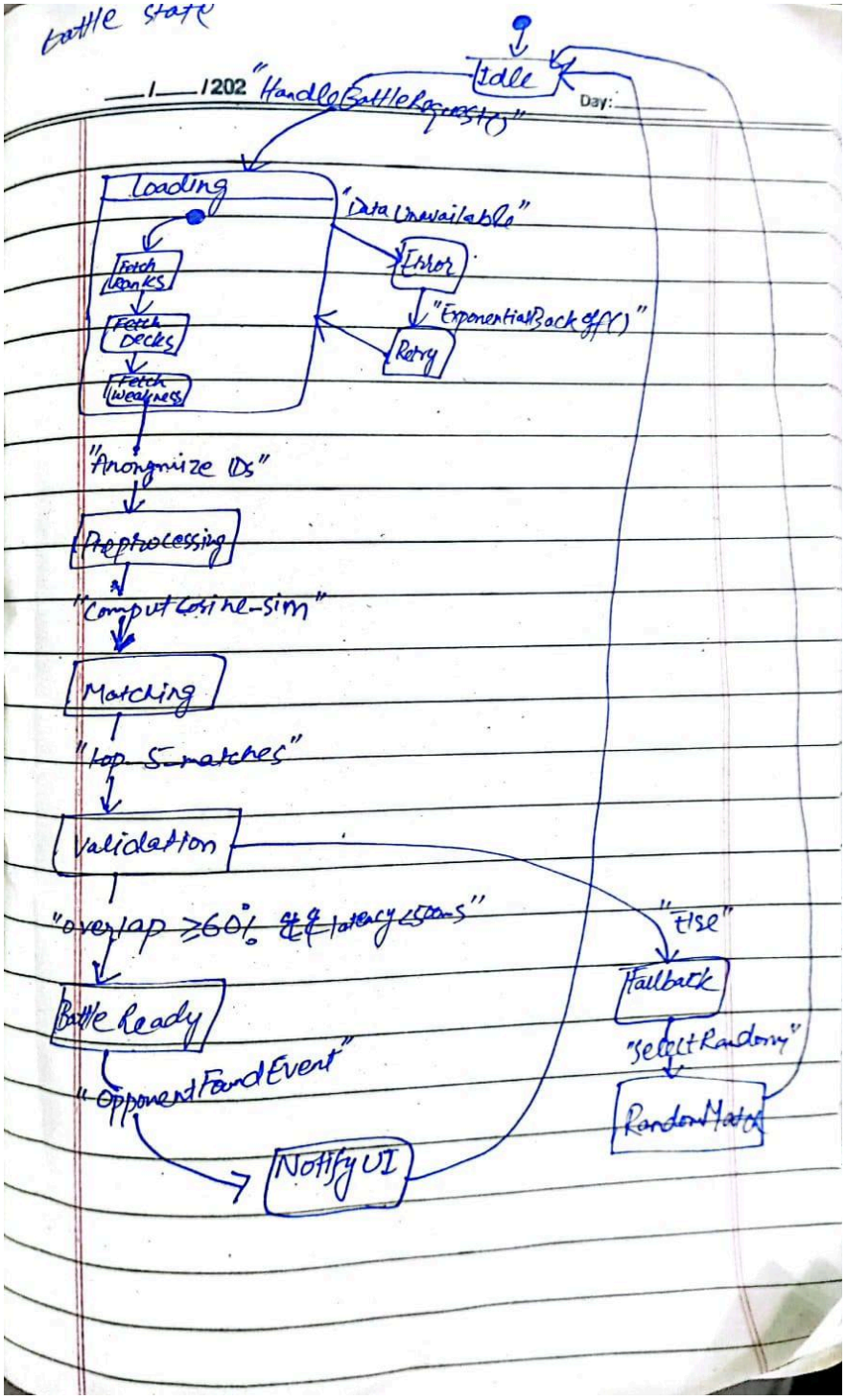


6. State Machine for ML Service

Weakness-based recommendation engine

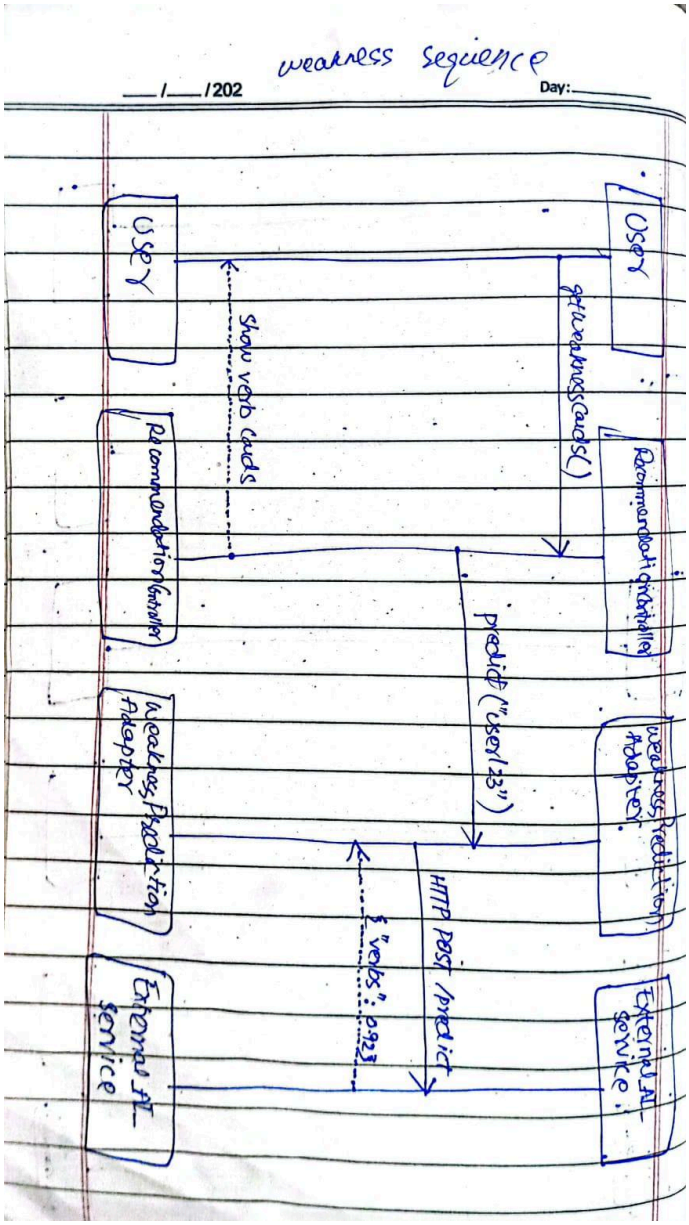


Battle opponent matchmaker



6. Sequence diagrams

Weakness-based recommendation engine



Battle opponent matchmaker