Voice-Enabled Government Scheme Recommendation Agent

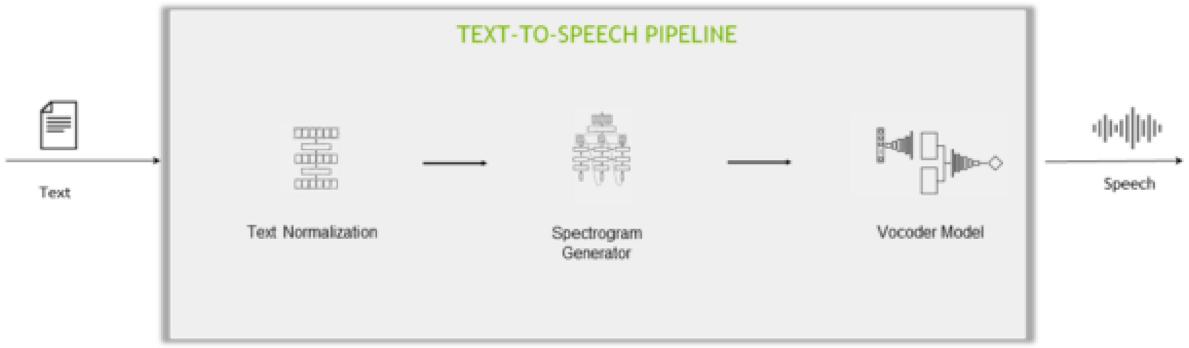
1. System Overview

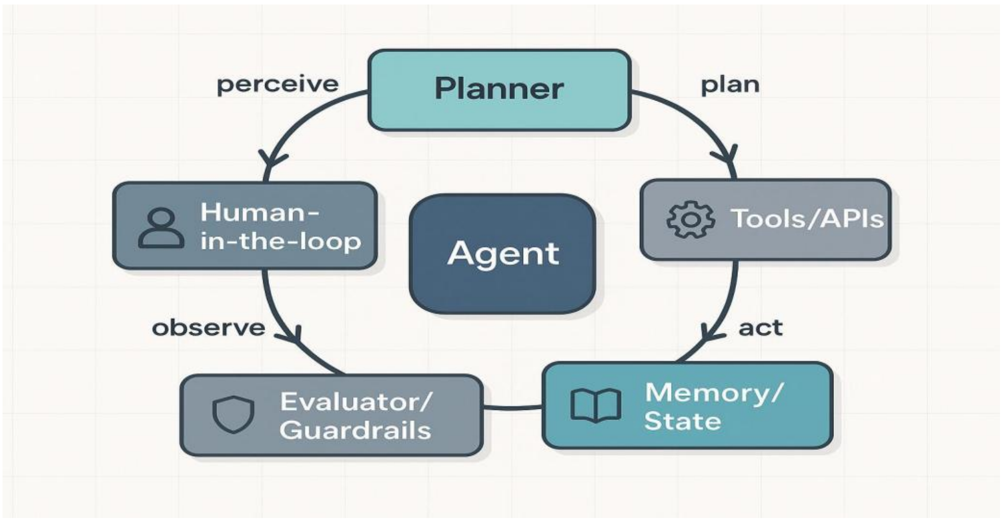
The system is a **voice-driven AI agent** that interacts with a user, collects required personal information step by step, confirms inputs to reduce speech errors, and determines eligibility for government welfare schemes.

Core goals:

- Voice-first interaction (STT + TTS)
- Robust handling of speech errors
- Deterministic decision-making
- Transparent eligibility reasoning

2. High-Level Architecture

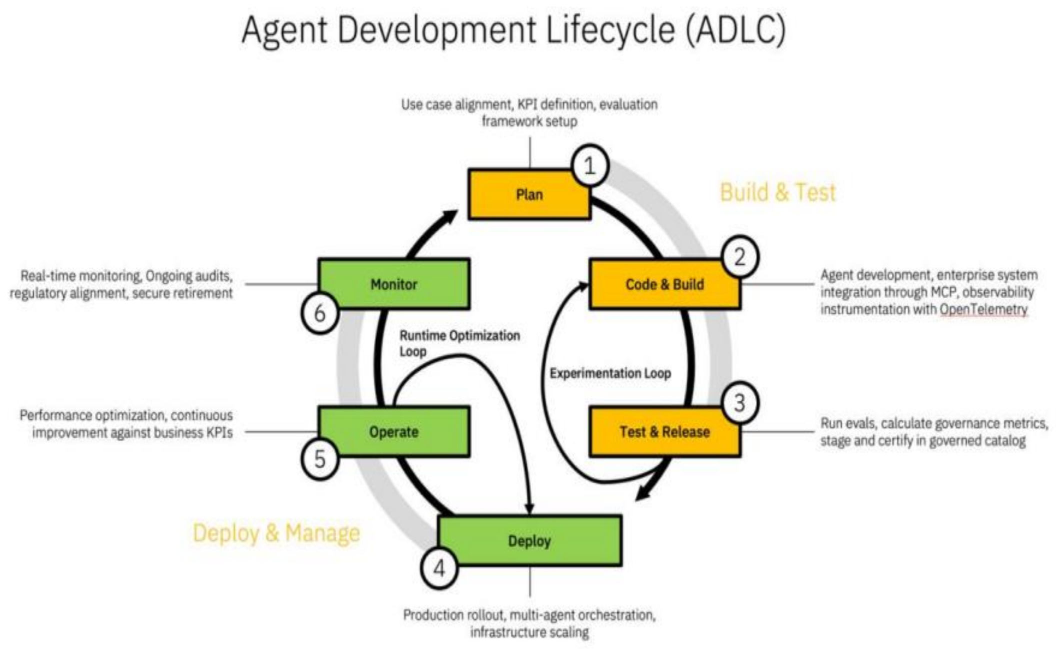


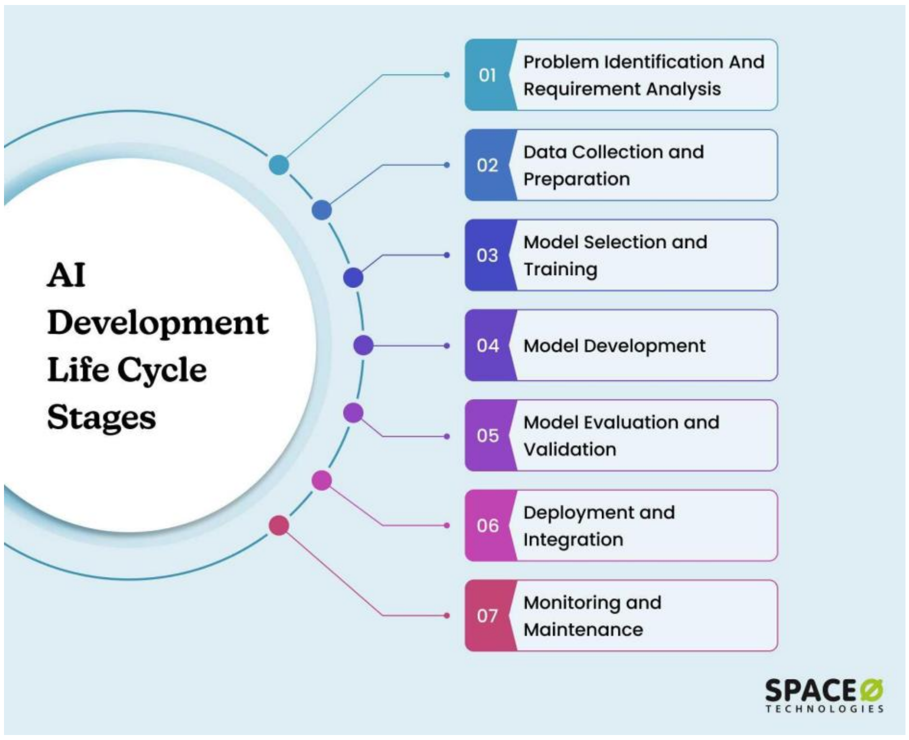


Components

Layer	Component	Responsibility
Interface	STT (Whisper)	Converts user speech → text
Interface	TTS (gTTS)	Converts agent text → speech
Agent Core	Planner	Decides next action
Agent Core	Executor	Executes the decided action
Agent Core	Evaluator	Validates output & decides state
Memory	Agent Memory	Stores user profile + state
Tools	Scheme Retriever	Loads scheme data
Tools	Eligibility Checker	Matches user vs scheme rules

3. Agent Lifecycle


Agent Development Lifecycle (ADLC)
Use case alignment, KPI definition, evaluation framework setup
1
Plan
Build & Test
2
Code & Build
Agent development, enterprise system integration through MCP, observability instrumentation with OpenTelemetry
Experimentation Loop
3
Test & Release
Run evals, calculate governance metrics, stage and certify in governed catalog
4
Deploy
Production rollout, multi-agent orchestration, infrastructure scaling
Deploy & Manage
5
Operate
Performance optimization, continuous improvement against business KPIs
Runtime Optimization Loop
6
Monitor
Real-time monitoring, Ongoing audits, regulatory alignment, secure retirement

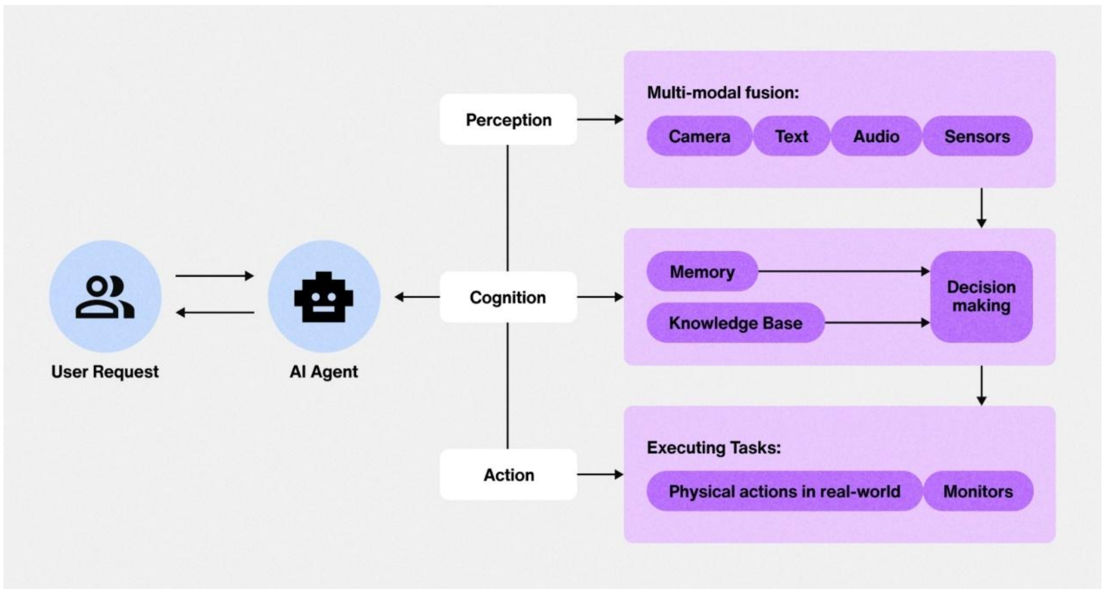

AI Development Life Cycle Stages
01 Problem Identification And Requirement Analysis
02 Data Collection and Preparation
03 Model Selection and Training
04 Model Development
05 Model Evaluation and Validation
06 Deployment and Integration
07 Monitoring and Maintenance
SPACE TECHNOLOGIES

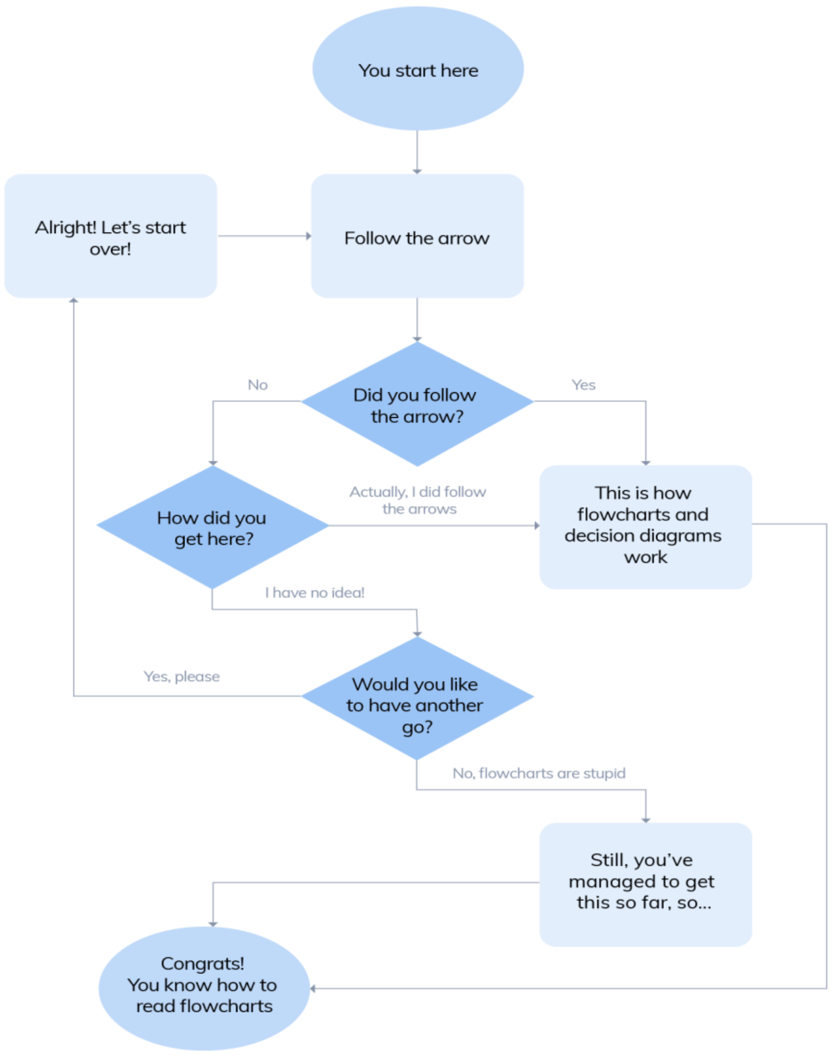
Lifecycle Steps

1. **Greeting**
2. **User speaks**
3. **Speech → Text (STT)**
4. **Confirmation loop**
5. **Planner decision**
6. **Executor action**
7. **Evaluator validation**
8. **Repeat OR Complete**

This loop continues until all required information is collected and eligibility is computed.

4. Detailed Decision Flow




You start here
Follow the arrow
Alright! Let's start over!
Did you follow the arrow?
No
Yes
How did you get here?
Actually, I did follow the arrows
This is how flowcharts and decision diagrams work
I have no idea!
Would you like to have another go?
Yes, please
No, flowcharts are stupid
Still, you've managed to get this so far, so...
Congrats! You know how to read flowcharts

Decision Logic (Planner)

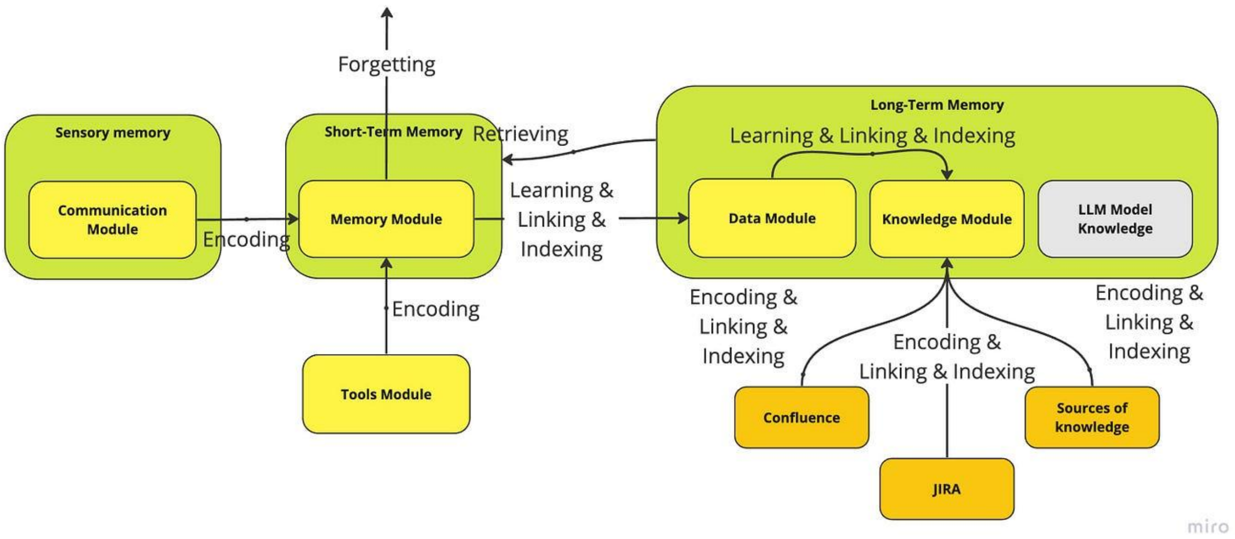
The planner always selects **exactly one** action:

```
IF contradictions exist
  → HANDLE_CONTRADICTION
ELSE IF any required field missing
  → ASK_MISSING_INFO:<field>
ELSE
  → CHECK_ELIGIBILITY
```

This guarantees:

- Deterministic behavior
- No hallucinated actions
- Predictable execution

5. Agent Memory Design



Memory Structure

```
{
  "name": "string",
  "age": "int",
  "income": "int",
  "gender": "male/female",
  "bpl": "boolean",
  "housing_status": "homeless/owned/rented"
}
```

Additional Memory State

- pending_confirmation → for ASR correction
- contradictions[] → conflict tracking
- history[] → conversation log
- state → collecting_info / completed

6. Prompt Architecture (Planner)

Planner Prompt (Design Principle)

The planner **never generates user-facing text**.
It only returns **one of three commands**.

Prompt Template

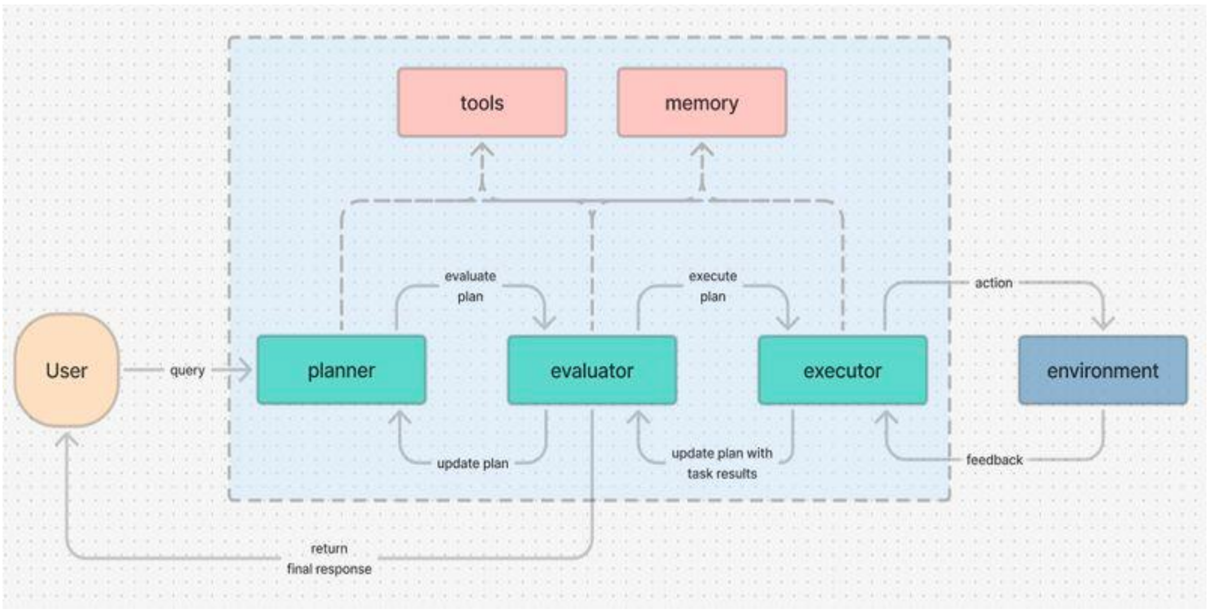
```
You are an internal planner for an AI agent.

User text:
{user_text}

Current memory:
{memory}

Contradictions:
{contradictions}

Choose exactly ONE:
1. ASK_MISSING_INFO:<field>
2. CHECK_ELIGIBILITY
```

3. HANDLE_CONTRADICTION

Why This Works

- Prevents hallucinations
- Keeps logic auditable
- Makes agent explainable

7. Executor–Evaluator Pattern



Executor

- Translates planner action → real operation
- Calls tools
- Fetches schemes
- Checks eligibility

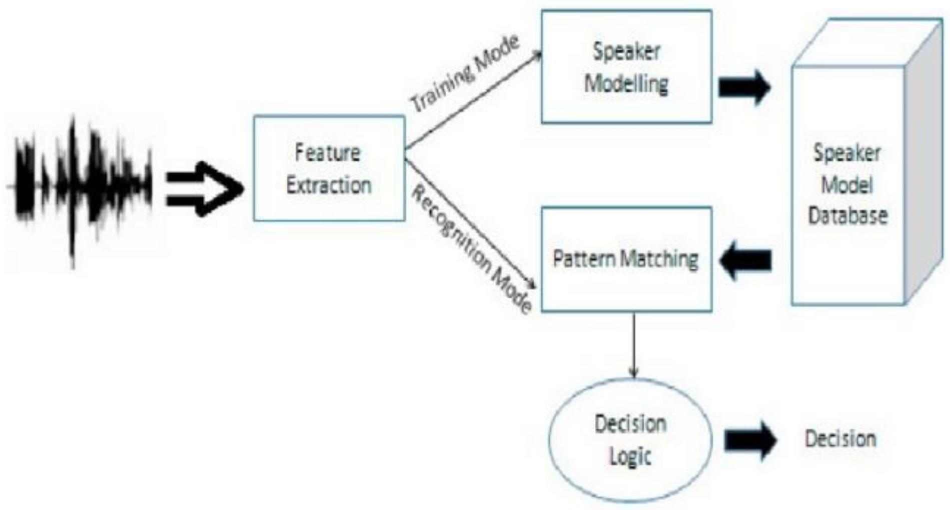
Evaluator

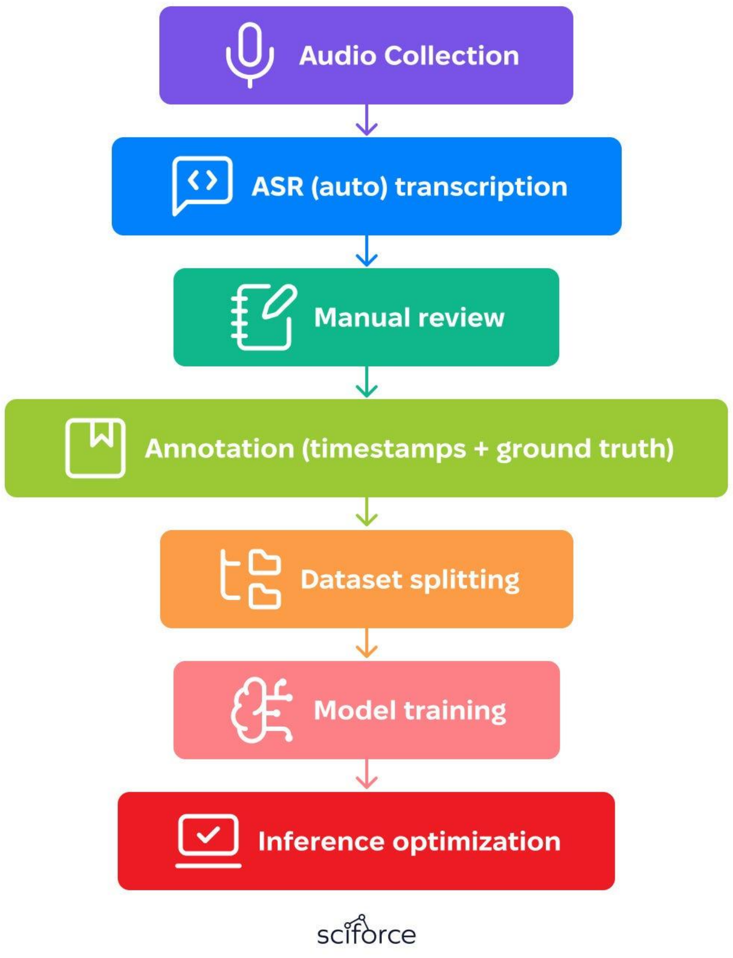
- Validates executor output
- Decides:
    - WAITING_FOR_USER
    - COMPLETED
    - FAILED

This separation mirrors **production-grade agent systems**.

8. Speech Error Handling (Key Design Strength)

Confirmation-Based Voice Pattern





Flow:

1. User speaks
2. STT transcribes
3. Agent repeats text
4. User confirms (Yes / No)
5. Only confirmed data is stored

Benefit

- Works even with noisy STT
- Transparent to user
- Highly reliable for real-world usage

9. Scheme Eligibility Engine

Rule-Based Matching

Each scheme defines explicit criteria:

```
{
  "income_max": 300000,
  "housing_status": "homeless"
}
```

Eligibility is checked using **deterministic rules**, not LLM guesses.

Advantages

- Explainable decisions
- No hallucination risk
- Easy to extend

10. End-to-End Flow Summary

```
User Voice
  ↓
Speech-to-Text (Whisper)
  ↓
Confirmation Loop
```

↓

Planner (LLM – decision only)

↓

Executor (tools + rules)

↓

Evaluator (state decision)

↓

Text-to-Speech (Odia response)