Program Workflow Guide

Overview

This is a real-time speech-to-text pipeline with conditional AI integration. The system transcribes speech, detects speakers, and intelligently routes questions to a Gemma AI model for conversational responses.

Architecture Overview

```
   Audio Input        Speech Pipeline      Text Output





  Speaker Detect      Conversation         Gemma AI
                  Manager
```

Core Components

1. Main Pipeline (`program_pipeline.py`)

Purpose: Orchestrates the entire workflow and manages the main program loop.

Key Functions: - `main()`: Entry point that initializes all components and runs the main loop - `handle_gemma_conversation()`: Manages AI conversation mode - `handle_listening_mode()`: Manages passive listening mode

Workflow: 1. Initializes Vosk speech recognition model 2. Sets up audio stream (16kHz, mono, 16-bit) 3. Runs continuous loop processing audio frames 4. Routes transcribed text to appropriate handlers 5. Manages program shutdown

2. Speech Processing (`speech_processor.py`)

SpeechProcessor Class **Purpose**: Handles Voice Activity Detection (VAD) and basic speech processing.

Key Methods: - `process_frame(audio_data)`: Analyzes audio frame for speech activity - Tracks speech/silence states - Manages VAD thresholds and timing

How it works: - Uses WebRTC VAD to detect speech vs. silence - Maintains state of speaking/silent periods - Provides boolean output for each audio frame

SpeakerDetector Class **Purpose**: Identifies different speakers and tracks speaker changes.

Key Methods: - `extract_voice_features(audio_data)`: Extracts voice characteristics - `detect_speaker_change(features)`: Determines if speaker has changed - `update_speaker_count(audio_data, silence_frames)`: Updates speaker detection

Voice Features Extracted: - Energy levels (mean, variance, RMS) - Pitch estimates (standard deviation) - Zero crossings (frequency content) - Spectral centroid (frequency distribution) - Peak amplitude

Speaker Change Detection: 1. **Silence-based**: Detects changes after silence periods 2. **Pattern-based**: Compares recent vs. older energy patterns 3. **Feature comparison**: Analyzes energy, pitch, and RMS differences

3. Conversation Management (`conversation_manager.py`)

Purpose: Manages conversation state and history.

Key Methods: - `should_enter_gemma_mode(text)`: Determines if text should trigger AI mode - `should_exit_gemma_mode(text)`: Checks for exit keywords - `add_to_history(text, is_user)`: Adds messages to conversation history - `get_conversation_context()`: Formats conversation for AI context - `reset_conversation()`: Clears conversation state

Conversation States: - **Listening Mode**: Passive transcription only - **Gemma Mode**: Active AI conversation

Entry Triggers: - Questions (detected by `utils.is_question()`) - Keywords: "hey gemma", "gemma", "ai", "assistant", "help"

Exit Triggers: - Keywords: "exit", "quit", "stop", "bye", "goodbye", "end conversation"

4. Gemma Integration (`gemma_client.py`)

Purpose: Handles all interactions with the Gemma AI model.

Key Methods: - `generate_response(prompt, context, timeout)`: Sends requests to Gemma - `is_server_available()`: Checks if Ollama server is running

API Integration: - Connects to Ollama server at `http://localhost:11434` - Uses REST API for text generation - Handles timeouts and error conditions - Formats prompts with conversation context

5. Conditional Routing (`conditional_gemma_input.py`)

Purpose: Determines when and how to route text to AI.

Key Methods: - `check_conditions(text, speaker, emotion, confidence)`: Evaluates routing criteria - `process_transcription(text, speaker,` `emotion, confidence)`: Main routing logic - `generate_response(text, speaker, emotion)`: Creates AI responses

Routing Conditions: - **Questions**: Automatically routed to AI - **Keywords**: Specific trigger words - **Length**: Min/max text length requirements - **Speaker**: Specific speaker routing (optional) - **Confidence**: Transcription confidence threshold

Pre-configured Condition Sets: - `questions_only`: Routes only questions to AI - `keywords_only`: Routes based on keywords only - `route_all`: Routes everything to AI

6. Utility Functions (`utils.py`)

Purpose: Shared helper functions used across multiple modules.

Key Functions: - `is_question(text)`: Detects if text is a question - `contains_keywords(text, keywords)`: Checks for keyword presence - `truncate_history(history, max_items)`: Limits history size - `format_conversation_context(history, max_messages)`: Formats conversation for AI

Question Detection Logic: 1. Checks for question marks (?) 2. Identifies question words: what, how, why, when, where, who, which 3. Detects question prefixes: is, are, do, does, can, will

Complete Workflow

1. Initialization Phase

```
1. Load Vosk speech recognition model
2. Initialize audio stream (PyAudio)
3. Create component instances:
   - SpeechProcessor (VAD)
   - SpeakerDetector (speaker identification)
   - ConversationManager (state management)
   - GemmaClient (AI integration)
   - ConditionalGemmaPipeline (routing logic)
4. Start audio capture loop
```

2. Audio Processing Loop

```
For each audio frame (2048 samples):
1. Read audio data from microphone
2. Process through SpeechProcessor (VAD)
3. If speech detected:
   - Update SpeakerDetector with audio features
   - Track speaker changes and count
4. Feed audio to Vosk recognizer
```

```
5. If transcription available:
   - Display transcribed text
   - Show speaker information
   - Route to conversation handler
```

3. Text Processing Workflow

```
When text is transcribed:
1. Check if in Gemma mode or listening mode
2. If in listening mode:
   - Check if should enter Gemma mode
   - If yes: Switch to Gemma mode and get initial AI response
   - If no: Continue listening
3. If in Gemma mode:
   - Add user text to conversation history
   - Check if should exit Gemma mode
   - If yes: Return to listening mode
   - If no: Get AI response and continue conversation
```

4. AI Integration Workflow

```
When routing to Gemma:
1. Check routing conditions (questions, keywords, etc.)
2. If conditions met:
   - Format conversation context
   - Send request to Gemma via Ollama API
   - Display AI response
   - Add response to conversation history
3. If conditions not met:
   - Skip AI processing
   - Continue with transcription only
```

5. Speaker Detection Workflow

```
For each speech frame:
1. Extract voice features (energy, pitch, spectral data)
2. Compare with previous speaker characteristics
3. If significant change detected:
   - Increment speaker count
   - Assign new speaker ID (Speaker A, B, C, etc.)
   - Update speaker profiles
4. Estimate total number of unique speakers
5. Display current speaker and count
```

Key Features

Real-time Processing

- Continuous audio stream processing
- Low-latency transcription
- Immediate speaker detection
- Instant AI responses

Intelligent Routing

- Question-based AI activation
- Keyword-triggered responses
- Configurable routing conditions
- Context-aware conversations

Multi-speaker Support

- Automatic speaker identification
- Speaker change detection
- Voice characteristic analysis
- Speaker count estimation

Conversation Management

- State-based conversation flow
- History management
- Context preservation
- Easy mode switching

Error Handling

- Graceful audio error recovery
- API timeout management
- Server availability checking
- Clean shutdown procedures

Configuration Options

Audio Settings

- Sample rate: 16kHz
- Channels: Mono
- Bit depth: 16-bit
- Frame size: 2048 samples

VAD Settings

- Aggressiveness: Level 2

- Silence threshold: 3 frames
- Speech detection sensitivity: Configurable

AI Settings

- Model: gemma3n:e4b (default)
- Timeout: 30 seconds
- Context length: 6 messages
- History limit: 10 messages

Routing Conditions

- Question detection: Enabled
- Keywords: Configurable list
- Min/max text length: 5-500 characters
- Confidence threshold: 0.7

Usage Examples

Basic Usage

```
python program_pipeline.py
```

Custom Conditions

```python
from conditional_gemma_input import ConditionalGemmaPipeline

# Route only questions
pipeline = ConditionalGemmaPipeline(
    model="gemma3n:e4b",
    conditions=CONDITIONS['questions_only']
)

# Route everything
pipeline = ConditionalGemmaPipeline(
    model="gemma3n:e4b",
    conditions=CONDITIONS['route_all']
)
```

Standalone Gemma Chat

```
python gemma_runner.py --model gemma3n:e4b
```

Troubleshooting

Common Issues

1. **No audio input**: Check microphone permissions and connections

2. **Vosk model not found**: Ensure model is in correct directory
3. **Ollama not responding**: Check if Ollama server is running
4. **Poor transcription**: Adjust microphone position and reduce background noise

Performance Optimization

- Use SSD storage for Vosk model
- Ensure sufficient RAM for audio processing
- Close unnecessary applications
- Use wired microphone for better quality

This refactored system provides a clean, modular, and efficient pipeline for real-time speech processing with intelligent AI integration.