VoiceTranslate Pro - Development Roadmap

> **Purpose:** Detailed implementation guide for future development phases
> **Last Updated:** 2026-02-21
> **Current Version:** v2.0.0+delta-time
> **Next Phase:** Phase 4 - Meeting Minutes & Speaker Recognition

📋 Quick Reference

Phase	Status	Priority	Est. Time
PoC	📋 For Review	Critical	1 week
Phase 4	📋 Planned	High	5 weeks
Phase 5	📋 Planned	Medium	2 weeks

Current Status: 🔄 **REVIEW PHASE - Not Ready for Implementation**

Action Required: Review and approve Proof of Concept plan before proceeding.

🔬 Proof of Concept (PoC) - CRITICAL FIRST STEP

> **⚠ DO NOT START IMPLEMENTATION UNTIL PoC IS COMPLETE AND REVIEWED**

PoC Objective

Validate technical feasibility and compatibility before committing to full implementation.

PoC Folder Structure (Isolated)

All PoC code must be in a **separate, isolated directory** to avoid polluting the main codebase:

```
project-root/
├── src/                          # ⛔ DO NOT MODIFY - Main codebase
│   ├── gui/
│   ├── core/
│   └── ...
├── tests/                        # ⛔ DO NOT MODIFY
├── docs/
├── poc/                          # ✅ PoC WORK DIRECTORY
│   ├── README.md                 # PoC setup and run instructions
│   ├── requirements.txt          # PoC-specific dependencies
│   ├── poc1_fluent_widgets/      # PoC 1: UI Compatibility
│   │   ├── test_fluent_theme.py
│   │   ├── test_threading.py
│   │   └── results.md
│   ├── poc2_speaker_diarization/ # PoC 2: Speaker Detection
```

```
│   │   ├── test_diarization.py
│   │   ├── test_latency.py
│   │   └── results.md
│   ├── poc3_data_model/          # PoC 3: Model Coexistence
│   │   ├── test_coexistence.py
│   │   └── results.md
│   └── poc4_model_download/      # PoC 4: Model Management (NEW)
│       ├── test_downloader.py
│       ├── test_resume.py
│       └── results.md
```

PoC Rules:

1. **Isolate:** All PoC code in `poc/` folder only
2. **Don't modify src/:** Main codebase remains untouched during PoC
3. **Copy if needed:** Copy relevant files from `src/` to `poc/` for testing
4. **Document:** Each PoC must have `results.md` with findings
5. **Clean:** PoC folder can be deleted after implementation starts

PoC 1: PyQt-Fluent-Widgets Compatibility Test

Duration: 2-3 days
Location: `poc/poc1_fluent_widgets/`
Goal: Verify PyQt-Fluent-Widgets works with existing PySide6 codebase

Test Scenarios:

1. **Basic Integration** (`test_fluent_theme.py`)

```python
# In poc/poc1_fluent_widgets/test_fluent_theme.py
import sys
sys.path.insert(0, '../../src')  # Access src without modifying

from qfluentwidgets import FluentWindow, setTheme, Theme
from PySide6.QtWidgets import QApplication

app = QApplication([])
setTheme(Theme.DARK)
window = FluentWindow()
# Test with copied TranslationDisplay widget from src/
```

2. **Audio Threading Compatibility** (`test_threading.py`)
   - Copy existing audio capture code from `src/audio/`
   - Test if Fluent widgets work with QThread-based audio
   - Verify no UI freezing with real-time updates
   - Check signal/slot behavior with Fluent components

3. **Performance Impact**

    - Measure startup time difference (Fluent vs standard Qt)
    - Check memory usage increase
    - Verify no regression in ASR latency

Deliverables:

- `poc/poc1_fluent_widgets/test_fluent_theme.py` - Basic theme test
- `poc/poc1_fluent_widgets/test_threading.py` - Audio threading test
- `poc/poc1_fluent_widgets/results.md` - Findings and benchmarks

Success Criteria:

- ☐ Existing translation pipeline works with Fluent theme
- ☐ No audio dropouts or UI freezing
- ☐ Startup time increase < 20%
- ☐ Memory increase < 50MB

Decision Points:

- ✅ **PASS:** Proceed with PyQt-Fluent-Widgets for Meeting Mode
- ❌ **FAIL:** Use custom QSS theme instead (fallback option in Section 4.3)

PoC 2: Speaker Diarization Integration

Duration: 2-3 days
Location: `poc/poc2_speaker_diarization/`
Goal: Test turn-based speaker detection with existing pipeline

Test Setup:

```
# In poc/poc2_speaker_diarization/test_diarization.py
import sys
sys.path.insert(0, '../../src')

from src.core.pipeline.orchestrator import TranslationPipeline
# Create test version of SimpleSpeakerDiarization in poc/
from speaker_test import SimpleSpeakerDiarization

pipeline = TranslationPipeline(config)
diarization = SimpleSpeakerDiarization(max_speakers=3)

# Test: Does adding speaker detection affect ASR latency?
# Measure: Processing time with/without diarization
```

Test Scenarios:

1. **Latency Test** (`test_latency.py`)
    - Measure ASR processing time with/without diarization
    - Test with varying speaker counts (2, 3, 4)

- Verify thread safety with concurrent audio capture

2. **Integration Test** (`test_integration.py`)
   - Test diarization with draft/final streaming modes
   - Verify speaker assignment accuracy with simulated audio
   - Test speaker count changes mid-session

Deliverables:

- `poc/poc2_speaker_diarization/speaker_test.py` - Test diarization implementation
- `poc/poc2_speaker_diarization/test_latency.py` - Latency benchmarks
- `poc/poc2_speaker_diarization/test_integration.py` - Pipeline integration test
- `poc/poc2_speaker_diarization/results.md` - Findings and metrics

Success Criteria:

- ☐ Speaker detection adds < 50ms latency
- ☐ Works with 2-4 speaker scenarios
- ☐ Can coexist with draft/final streaming modes
- ☐ Thread-safe operation verified

Decision Points:

- ✅ **PASS:** Proceed with turn-based approach
- ❌ **FAIL:** Delay speaker recognition to V2, focus on minutes format only

PoC 3: Meeting Data Model Coexistence

Duration: 1-2 days
Location: `poc/poc3_data_model/`
Goal: Verify MeetingEntry model works alongside existing TranslationEntry

Test Scenarios:

1. **Model Coexistence** (`test_coexistence.py`)
   - Create MeetingEntry and TranslationEntry in same test session
   - Verify no conflicts between data models
   - Test memory usage with both models active

2. **Export Compatibility** (`test_export.py`)
   - Test export functions for both formats
   - Verify no data loss when converting between modes
   - Test simultaneous export of both formats

Deliverables:

- `poc/poc3_data_model/test_coexistence.py` - Model coexistence test
- `poc/poc3_data_model/test_export.py` - Export functionality test
- `poc/poc3_data_model/results.md` - Findings

Success Criteria:

- [ ] Both models can coexist without conflicts
- [ ] Export functions work independently
- [ ] No data loss when switching modes
- [ ] Memory usage acceptable (<100MB overhead)

Decision Points:

- ✅ **PASS:** Proceed with dual-mode architecture
- ❌ **FAIL:** Consider separate applications for each mode

PoC 4: Model Download & Management (NEW - Critical)

Duration: 2 days
Location: `poc/poc4_model_download/`
Goal: Test robust model downloading without installer

Test Scenarios:

1. **Basic Download** (`test_downloader.py`)
   - Download small test file (10MB) from HuggingFace
   - Verify progress bar updates correctly
   - Test checksum verification
2. **Resume & Retry** (`test_resume.py`)
   - Simulate interrupted download (kill mid-download)
   - Test resume capability
   - Test retry logic on network failure
   - Test mirror fallback
3. **Permission Test** (`test_permissions.py`)
   - Test log directory creation in user home
   - Test model directory creation
   - Verify graceful handling of permission denied

Deliverables:

- `poc/poc4_model_download/model_manager_test.py` - Test ModelManager implementation
- `poc/poc4_model_download/test_downloader.py` - Download functionality
- `poc/poc4_model_download/test_resume.py` - Resume/retry logic
- `poc/poc4_model_download/test_permissions.py` - Directory permissions
- `poc/poc4_model_download/results.md` - Findings

Success Criteria:

- [ ] Downloads work on both Windows and macOS
- [ ] Progress bars show accurate % and speed
- [ ] Resume interrupted downloads

- ☐ Retry on network failure (3 attempts)
- ☐ Graceful handling of permission errors
- ☐ Checksum verification catches corrupt downloads

Decision Points:

- ✅ **PASS:** Proceed with portable distribution + model download
- ❌ **FAIL:** Require manual model download by users

Current Focus: Phase 4 - Meeting Minutes with Speaker Recognition

---

🛠️ Technology Stack

Component	Technology	Reason
Language	Python 3.10+	Best ecosystem for AI/ML
GUI Framework	PySide6 + PyQt-Fluent-Widgets	Modern UI. Critical: Verify compatibility in PoC
Speech-to-Text	faster-whisper	Optimized for Edge (CPU/GPU), low latency
Speaker Diarization	SimpleSpeakerDiarization (V1)	Turn-based detection. Low resource usage for MVP
Audio Input	sounddevice	Cross-platform microphone access
Logging	loguru	Easy rotation, formatting, and privacy modes
Distribution	Portable Executable / Script	No installer wizard. Users run directly

⚠ Critical Technical Risks (Revised)

Risk 1: PySide6 vs PyQt-Fluent-Widgets Compatibility 🔴 HIGH

Issue: PyQt-Fluent-Widgets may conflict with existing PySide6 code, causing UI freezing or threading issues.

Mitigation:

- **PoC 1 is CRITICAL:** Must verify compatibility before proceeding
- If PoC fails, revert to standard Qt Styling (QSS) immediately
- Test specifically with audio threading and real-time updates

Impact: If failed, modern UI goal fails. Have fallback ready.

Risk 2: Model Management (No Installer) 🟡 MEDIUM

Issue: Without installer, cannot guarantee models exist on first run. Must handle downloads robustly.

Mitigation:

- **ModelManager class:** Dedicated downloader with retries, progress bars, error handling
- **Checksum verification:** Verify SHA256 before loading to catch corrupt downloads
- **Mirror support:** Allow switching to secondary mirror if HuggingFace is blocked/slow
- **Offline mode:** Allow app launch even if models fail (show error state, don't block)

Risk 3: Portable App Permissions 🟡 MEDIUM

Issue: Without installer, macOS/Windows may flag the app. Users need manual permission guidance.

Mitigation:

- **First Run Wizard:** Clear guidance for microphone access in OS settings
- **Permission check:** Detect denied permissions and show helpful dialog
- **User-writable paths:** Write logs to ~/Documents or ~/AppData, not Program Files

Risk 4: Diarization Latency 🟡 MEDIUM

Issue: Speaker detection could block ASR thread if not implemented correctly.

Mitigation:

- **V1 Turn-Based:** Must be proven <50ms latency in PoC
- **Offload to QRunnable:** If upgrading to V2 embeddings, use separate thread pool
- **Asynchronous processing:** Never block UI or audio capture threads

Risk 5: Logging Permissions 🟢 LOW

Issue: Portable app must handle log directory creation gracefully.

Mitigation:

- **Auto-create directories:** `~/.voicetranslate/logs` created on first run
- **Graceful fallback:** If permission denied, log to console only
- **Pathlib usage:** Use `Path.home()` for all user data paths

🎯 New Design Goals

Based on user requirements for a professional meeting transcription application:

Approach: Option A - Incremental Integration

> ⚠ **Design Decision:** Meeting Mode will be added as a **new view/window** alongside the existing Translation Mode, NOT a complete replacement.

Rationale:

- Existing GUI (`src/gui/main.py`, 54KB+) has mature translation functionality
- Interview Mode and Documentary Mode serve different use cases
- Gradual migration reduces risk and maintains backward compatibility
- Users can choose between Translation Mode and Meeting Mode

Core Features

1. **Meeting Mode** - New window/view for meeting transcription with minutes format
2. **Speaker Recognition** - Identify and label different speakers (with user-configurable count)
3. **Modern GUI Theme (Optional)** - PyQt-Fluent-Widgets for new Meeting Mode only
4. **Audio Test Function** - Real-time level meter for any mode
5. **Debug Logging** - Comprehensive logging for troubleshooting

Architecture Principles

- **Backward Compatible:** Existing Translation Mode remains unchanged
- **Shared Components:** Reuse ASR pipeline, add speaker diarization as optional addon
- **Gradual Theme Migration:** New Meeting Mode gets modern theme first
- **Data Separation:** Meeting minutes data model alongside existing translation entries

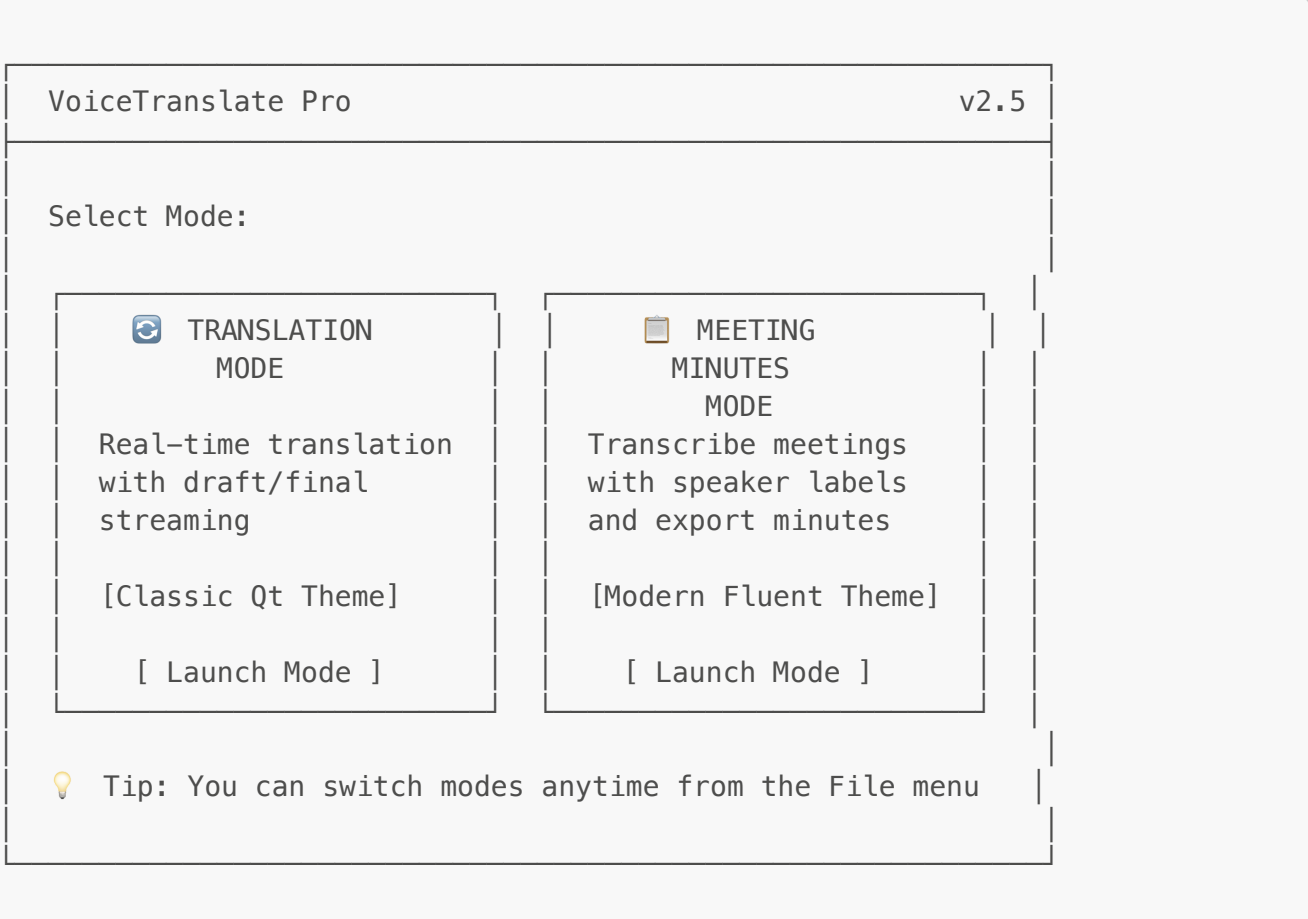
---

🎨 GUI Design Specification

> **Target Audience:** Customers, Stakeholders, Development Team
> **Design Philosophy:** Professional, Minimalist, Focus on Content
> **Architecture:** New Meeting Mode window coexists with existing Translation Mode

Mode Selection (New)

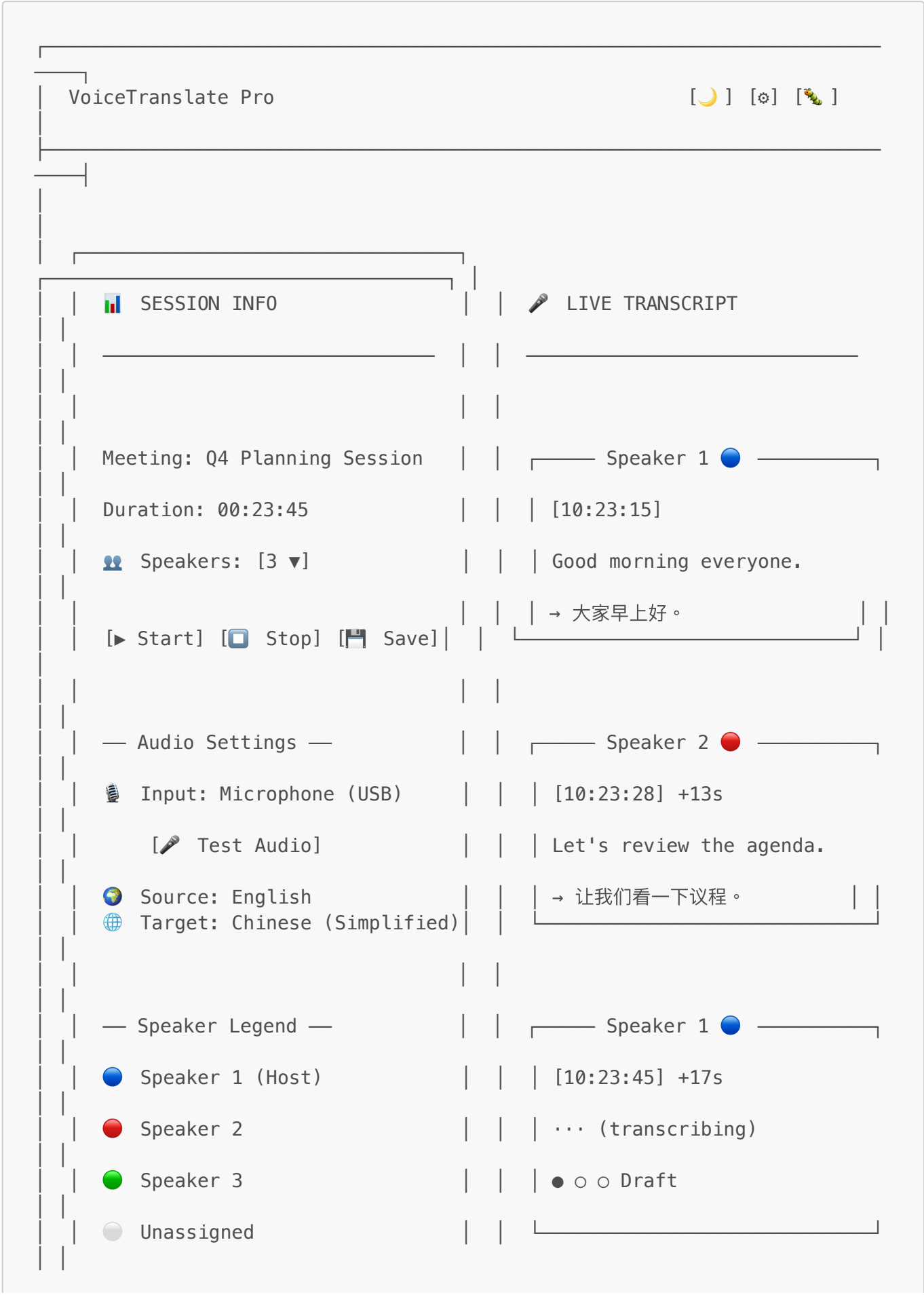
On launch, user selects mode:

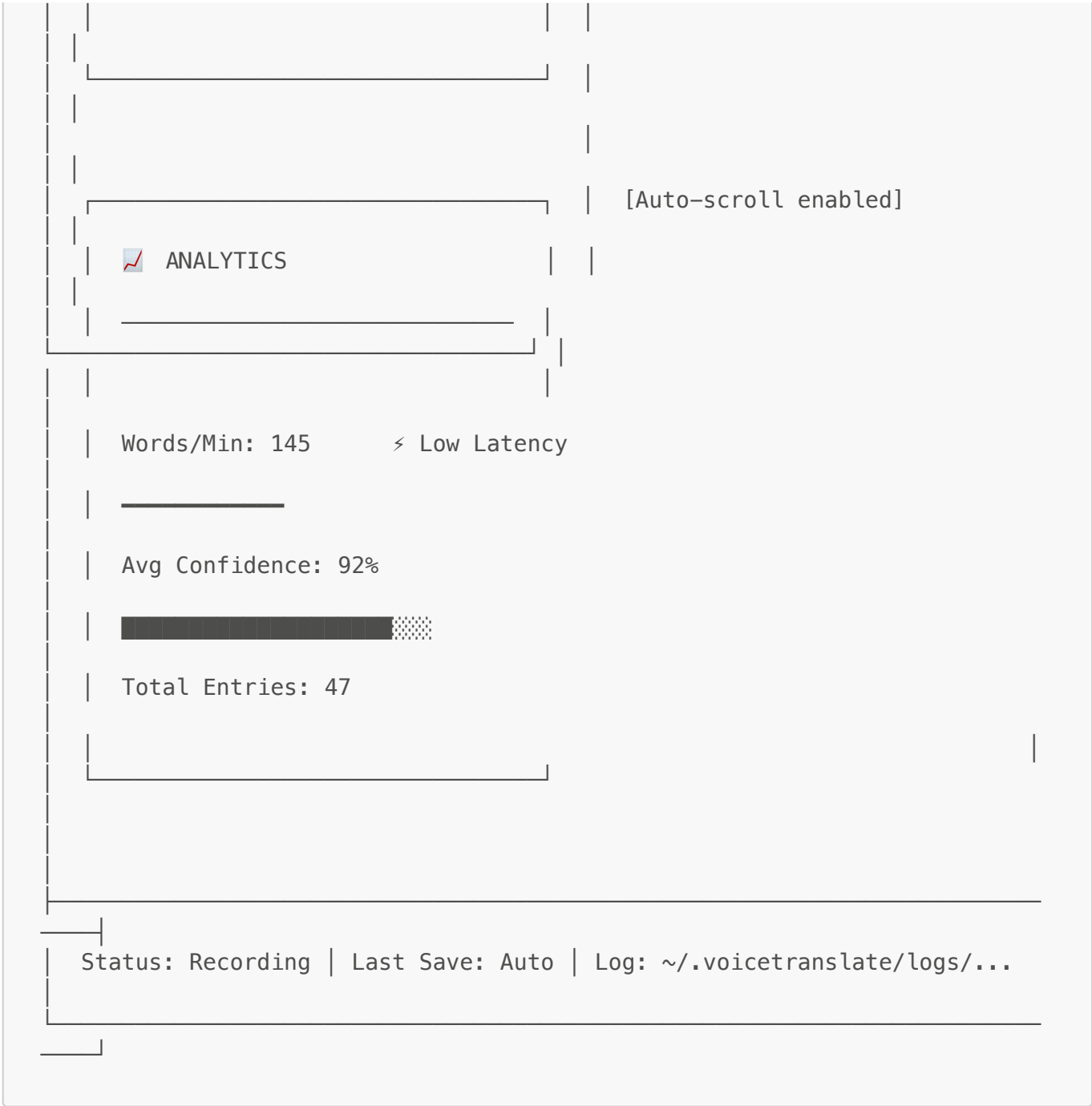


Meeting Mode Layout (New Window)

> **Note:** This is a NEW window class `MeetingWindow`, not a replacement for existing GUI.

Layout Overview





Key Differences from Existing Translation Mode:

Feature	Translation Mode (Existing)	Meeting Mode (New)
Primary Output	Real-time translation	Meeting minutes with speakers
Speaker Support	None	2-8 speakers with color coding
Export Formats	TXT, SRT, VTT	Markdown, JSON
UI Theme	Classic PySide6/Qt	PyQt-Fluent-Widgets (if PoC passes)
Display Style	Sequential text	Speaker bubbles
Target Use Case	Live translation	Meeting transcription

> Both modes share: Audio device selection, ASR pipeline, debug logging, CPU/RAM indicator

Component Breakdown

1. Header Bar

Element	Description
App Title	"VoiceTranslate Pro" with version badge
Theme Toggle	🌙 /☀️ Dark/Light mode switch
Settings	⚙️ Preferences (language defaults, audio device)
Debug	🐞 Open log folder / Generate crash report

2. Left Panel: Control & Info (30% width)

Session Info Card

- Editable meeting title
- Live duration timer
- **Speaker Count Selector:** Input expected number of speakers (2-8)
- Transport controls (Start/Stop/Save)
- Audio test button

Audio Settings Card

- Input device dropdown with levels
- Source language selector
- Target language selector (optional)
- Quality/Mode toggle (Speed vs Accuracy)

Speaker Legend Card

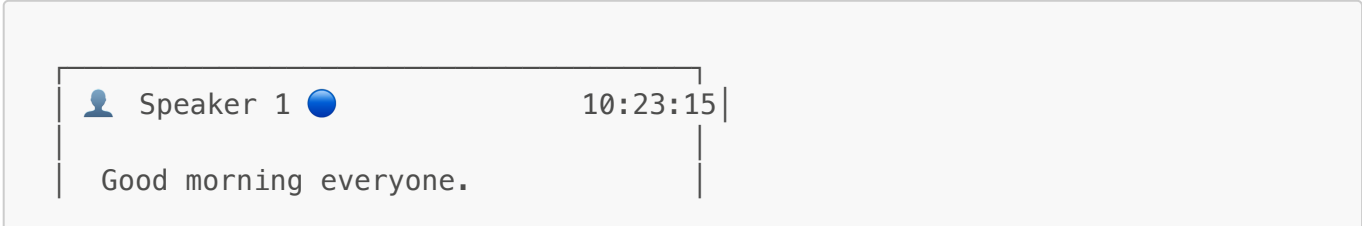
- Color-coded speaker list
- Click to rename speakers (updates all previous entries retroactively)
- Drag to merge speakers

Analytics Card (collapsible)

- Real-time WPM (Words Per Minute)
- Confidence score bar
- Session statistics

3. Right Panel: Live Transcript (70% width)

Speaker Bubble Design



```
│  Let's start with the Q4 review.     │
│                                      │
│  → 大家早上好。让我们从Q4回顾开始。   │
│                            [✓ Final] │
└──────────────────────────────────────┘
```

Bubble States:

- 🟡 **Draft:** Pale yellow background, "Draft" label, pulsing indicator
- 🟢 **Final:** White/dark background, checkmark, timestamp locked
- 🔵 **Translating:** Blue tint, spinner, "Translating..." text

Visual Indicators:

- **Timestamp:** Top-right corner `[HH:MM:SS]`
- **Delta Time:** Small text below timestamp `+12s`
- **Speaker Avatar:** Colored circle with first letter
- **Translation:** Italic, slightly muted, preceded by arrow →
- **Confidence:** Subtle border color (green=high, yellow=medium, red=low)

4. Status Bar

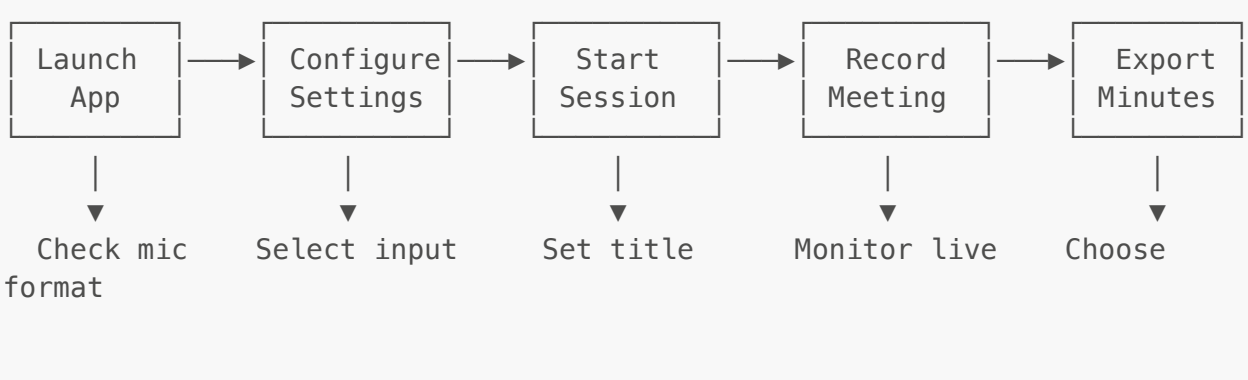
- **Left:** Connection/Recording status with colored dot
- **Center:**
  - Last auto-save time
  - CPU/RAM usage indicator (hover for details)
- **Right:** Log file location (click to open)

CPU/RAM Indicator:

```
⚡ CPU: 12% | RAM: 45% [████████▒▒▒▒▒▒▒▒▒▒▒▒]
```

- Shows real-time resource usage
- Warns if CPU >80% or RAM >85%
- Click to open detailed performance panel

User Flow



```
  permissions  & languages      (optional)     transcript
(MD/JSON/TXT)
```

First Run Wizard

Shown on first launch or when microphone permission is denied:

```
┌──────────────────────────────────────────────────────────┐
│  Welcome to VoiceTranslate Pro                [Skip] [Next]│
├──────────────────────────────────────────────────────────┤
│                                                          │
│  Step 1/3: Microphone Access                             │
│                                                          │
│  This app needs microphone access to transcribe meetings.│
│                                                          │
│  [🔴 Permission Denied]                                  │
│                                                          │
│  Please enable microphone access:                        │
│  1. Open System Settings → Privacy & Security → Microphone│
│  2. Enable "VoiceTranslate Pro"                          │
│  3. Restart the app                                      │
│                                                          │
│  [Open System Settings]           [Test Microphone]      │
│                                                          │
└──────────────────────────────────────────────────────────┘
```

Wizard Steps:

1. **Microphone Permission** - Check and guide user to enable access
2. **Audio Test** - Visual level meter to confirm mic is working
3. **Model Download** - Download ASR models if not bundled (~100MB)
4. **Default Languages** - Set preferred source/target languages
5. **Privacy Settings** - Enable/disable transcript logging

4.4 Audio Test Function

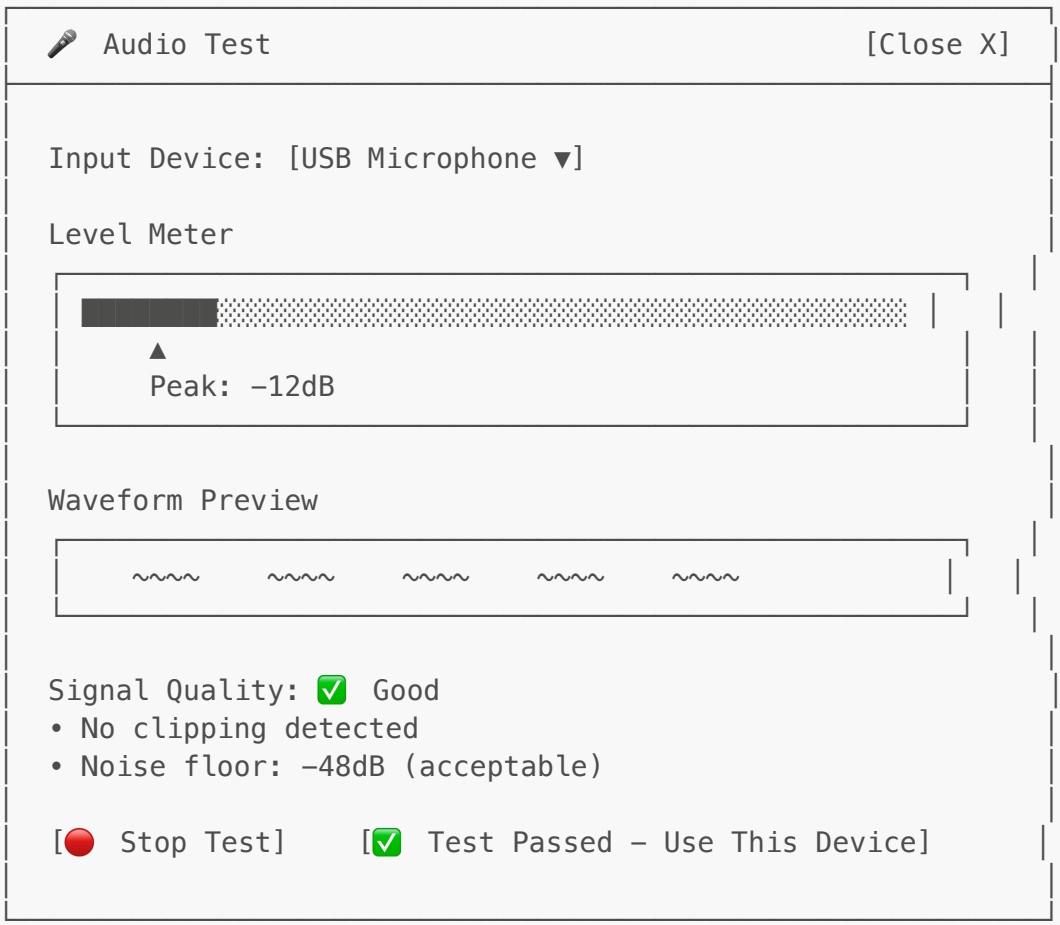
File: `src/gui/audio_test.py`

Purpose: Real-time microphone testing with visual feedback before starting transcription.

Features:

- Visual level meter (0-100%)
- Peak level indicator
- Signal quality assessment (clipping detection)
- Real-time waveform display
- Noise floor detection

UI Design:



Implementation:

```
"""
Audio test dialog with real-time level meter and waveform display.
"""

from PySide6.QtWidgets import (
    QDialog, QVBoxLayout, QHBoxLayout, QLabel,
    QPushButton, QComboBox, QProgressBar, QWidget
)
from PySide6.QtCore import Qt, QTimer, Signal
from PySide6.QtGui import QPainter, QColor, QPen
import numpy as np
import sounddevice as sd

class LevelMeter(QWidget):
    """Custom level meter widget."""

    def __init__(self, parent=None):
        super().__init__(parent)
        self.setMinimumHeight(40)
        self.level = 0.0  # 0.0 to 1.0
```

```
        self.peak = 0.0
        self.peak_hold_frames = 0

    def set_level(self, level: float):
        """Update level (0.0 to 1.0)."""
        self.level = min(1.0, max(0.0, level))
        if self.level > self.peak:
            self.peak = self.level
            self.peak_hold_frames = 30  # Hold for 30 frames (~500ms)
        self.update()

    def paintEvent(self, event):
        painter = QPainter(self)

        # Background
        painter.fillRect(self.rect(), QColor("#2D2D44"))

        # Level bar
        width = int(self.width() * self.level)
        height = self.height() - 4

        # Color gradient based on level
        if self.level > 0.9:
            color = QColor("#F44336")  # Red (clipping)
        elif self.level > 0.7:
            color = QColor("#FFC107")  # Yellow
        else:
            color = QColor("#4CAF50")  # Green

        painter.fillRect(2, 2, width, height, color)

        # Peak indicator
        if self.peak > 0:
            peak_x = int(self.width() * self.peak)
            painter.fillRect(peak_x - 2, 2, 4, height, QColor("#FFFFFF"))

        # Decay peak
        if self.peak_hold_frames > 0:
            self.peak_hold_frames -= 1
        else:
            self.peak *= 0.95  # Decay

class WaveformWidget(QWidget):
    """Real-time waveform display."""

    def __init__(self, parent=None, history_size=1000):
        super().__init__(parent)
        self.setMinimumHeight(60)
        self.history = np.zeros(history_size)
        self.write_pos = 0

    def add_samples(self, samples: np.ndarray):
        """Add audio samples to display buffer."""
        samples_to_write = min(len(samples), len(self.history))
```

```
            self.history[self.write_pos:self.write_pos + samples_to_write] =
samples[:samples_to_write]
            self.write_pos = (self.write_pos + samples_to_write) %
len(self.history)
            self.update()

        def paintEvent(self, event):
            painter = QPainter(self)
            painter.fillRect(self.rect(), QColor("#1E1E2E"))

            # Draw waveform
            pen = QPen(QColor("#6C5DD3"))
            pen.setWidth(2)
            painter.setPen(pen)

            width = self.width()
            height = self.height()
            center_y = height // 2

            # Draw samples
            points = []
            for x in range(0, width, 2):
                idx = (self.write_pos + int(x * len(self.history) / width)) %
len(self.history)
                sample = self.history[idx]
                y = center_y + int(sample * (height // 3))
                points.append((x, y))

            for i in range(len(points) - 1):
                painter.drawLine(points[i][0], points[i][1],
                                 points[i+1][0], points[i+1][1])

    class AudioTestDialog(QDialog):
        """Audio test dialog with level meter and waveform."""

        device_selected = Signal(int, str)  # device_index, device_name

        def __init__(self, parent=None):
            super().__init__(parent)
            self.setWindowTitle("Audio Test")
            self.setMinimumSize(500, 400)

            self.stream = None
            self.is_testing = False

            self._setup_ui()
            self._populate_devices()

        def _setup_ui(self):
            layout = QVBoxLayout(self)

            # Device selection
            device_layout = QHBoxLayout()
            device_layout.addWidget(QLabel("Input Device:"))
```

```
        self.device_combo = QComboBox()

self.device_combo.currentIndexChanged.connect(self._on_device_changed)
        device_layout.addWidget(self.device_combo, 1)
        layout.addLayout(device_layout)

        # Level meter
        layout.addWidget(QLabel("Level Meter:"))
        self.level_meter = LevelMeter()
        layout.addWidget(self.level_meter)

        self.peak_label = QLabel("Peak: -- dB")
        layout.addWidget(self.peak_label)

        # Waveform
        layout.addWidget(QLabel("Waveform Preview:"))
        self.waveform = WaveformWidget()
        layout.addWidget(self.waveform)

        # Quality assessment
        self.quality_label = QLabel("Signal Quality: Click 'Start Test' to
begin")
        layout.addWidget(self.quality_label)

        # Buttons
        button_layout = QHBoxLayout()
        self.test_button = QPushButton("🎤 Start Test")
        self.test_button.clicked.connect(self._toggle_test)
        button_layout.addWidget(self.test_button)

        self.save_audio_button = QPushButton("💾 Save Test Audio")
        self.save_audio_button.setEnabled(False)
        self.save_audio_button.clicked.connect(self._save_test_audio)
        self.save_audio_button.setToolTip("Save 5-second sample for
debugging")
        button_layout.addWidget(self.save_audio_button)

        self.use_button = QPushButton("✅ Use This Device")
        self.use_button.setEnabled(False)
        self.use_button.clicked.connect(self._accept_device)
        button_layout.addWidget(self.use_button)

        layout.addLayout(button_layout)

    def _populate_devices(self):
        """Populate audio device dropdown."""
        devices = sd.query_devices()
        for i, device in enumerate(devices):
            if device['max_input_channels'] > 0:
                self.device_combo.addItem(
                    f"{device['name']}",
                    userData=i
                )
```

```
    def _toggle_test(self):
        """Start or stop audio test."""
        if self.is_testing:
            self._stop_test()
        else:
            self._start_test()

    def _start_test(self):
        """Start audio capture for testing."""
        device_id = self.device_combo.currentData()

        def audio_callback(indata, frames, time_info, status):
            """Process incoming audio."""
            # Calculate RMS level
            rms = np.sqrt(np.mean(indata**2))
            level = min(1.0, rms * 10)  # Scale to 0-1

            # Update UI from main thread
            self.level_meter.set_level(level)
            self.waveform.add_samples(indata[:, 0])

            # Calculate dB
            db = 20 * np.log10(rms + 1e-10)
            self.peak_label.setText(f"Peak: {db:.1f} dB")

            # Quality assessment
            self._assess_quality(rms, np.max(np.abs(indata)))

        self.stream = sd.InputStream(
            device=device_id,
            channels=1,
            samplerate=16000,
            callback=audio_callback
        )
        self.stream.start()

        self.is_testing = True
        self.test_button.setText("🔴 Stop Test")
        self.use_button.setEnabled(True)

    def _stop_test(self):
        """Stop audio capture."""
        if self.stream:
            self.stream.stop()
            self.stream.close()
            self.stream = None

        self.is_testing = False
        self.test_button.setText("🎤 Start Test")
        self.use_button.setEnabled(False)

    def _assess_quality(self, rms: float, peak: float):
        """Assess audio quality."""
        if peak > 0.95:
```

```
            quality = "⚠ Clipping detected - Lower your input volume"
        elif rms < 0.01:
            quality = "⚠ Signal too quiet - Check microphone"
        else:
            noise_floor = -48  # Placeholder
            quality = f"✅ Good (Noise floor: {noise_floor}dB)"

        self.quality_label.setText(f"Signal Quality: {quality}")

    def _on_device_changed(self):
        """Handle device change."""
        if self.is_testing:
            self._stop_test()
            self._start_test()

    def _accept_device(self):
        """Accept selected device."""
        device_id = self.device_combo.currentData()
        device_name = self.device_combo.currentText()
        self.device_selected.emit(device_id, device_name)
        self.accept()

    def _save_test_audio(self):
        """Save 5-second test audio sample for debugging.

        Useful for support: if user reports 'Mic not working,'
        they can send this recorded sample.
        """
        from PySide6.QtWidgets import QFileDialog
        import wave
        import io

        # Get save location
        filepath, _ = QFileDialog.getSaveFileName(
            self,
            "Save Test Audio",
            "test_audio.wav",
            "WAV files (*.wav)"
        )

        if not filepath:
            return

        try:
            # Record 5 seconds of audio
            device_id = self.device_combo.currentData()
            sample_rate = 16000
            duration = 5

            recording = sd.rec(
                int(duration * sample_rate),
                samplerate=sample_rate,
                channels=1,
                dtype=np.int16,
```

```
                device=device_id
            )
            sd.wait()  # Wait until recording is finished

            # Save as WAV
            with wave.open(filepath, 'wb') as wf:
                wf.setnchannels(1)
                wf.setsampwidth(2)  # 16-bit
                wf.setframerate(sample_rate)
                wf.writeframes(recording.tobytes())

            logger.info(f"Test audio saved: {filepath}")

        except Exception as e:
            logger.error(f"Failed to save test audio: {e}")

    def closeEvent(self, event):
        """Clean up on close."""
        self._stop_test()
        super().closeEvent(event)
```

Integration Points:

1. **First Run Wizard:** Step 2 uses AudioTestDialog
2. **Settings Menu:** "Test Audio" button opens dialog
3. **Main Window:** Audio device selector includes "Test" button

Color Scheme (Dark Mode)

Element	Color	Hex
Background	Deep Navy	#1E1E2E
Surface	Dark Purple-Gray	#2D2D44
Primary	Purple	#6C5DD3
Secondary	Teal	#00D4AA
Accent	Coral	#FF6B6B
Text Primary	White	#FFFFFF
Text Secondary	Gray	#B4B4BE
Speaker 1	Blue	#4A90D9
Speaker 2	Coral	#E85D75
Speaker 3	Green	#50C878
Speaker 4	Orange	#F5A623

Responsive Behavior

Screen Size	Layout
Desktop (>1200px)	Full 2-panel layout
Tablet (800-1200px)	Collapsible left panel
Small (<800px)	Single panel, tabs for controls

Keyboard Shortcuts

Shortcut	Action
`Space`	Start/Stop recording
`Ctrl+S`	Save session
`Ctrl+E`	Export minutes
`Ctrl+,`	Open preferences
`Ctrl+D`	Toggle debug panel
`Esc`	Stop recording / Close dialog

Accessibility Features

- **High Contrast Mode:** Pure black/white theme
- **Font Scaling:** 3 sizes (Small/Medium/Large)
- **Screen Reader:** Full ARIA labels on all controls
- **Keyboard Navigation:** Tab order optimized for workflow

Phase 4: Meeting Minutes & Speaker Recognition

4.1 Meeting Minutes Format

Purpose: Transform translation output into professional meeting minutes with timestamps and speaker labels.

4.1.1 Meeting Minutes Data Model

File: `src/gui/meeting/minutes_model.py`

```
"""
Meeting minutes data model with speaker and timing information.
"""

from dataclasses import dataclass, field
from typing import List, Optional
from datetime import datetime
from enum import Enum

class SpeakerRole(Enum):
```

```
    """Speaker role in meeting."""
    UNKNOWN = "unknown"
    HOST = "host"
    PARTICIPANT = "participant"
    SPEAKER_A = "A"
    SPEAKER_B = "B"
    SPEAKER_C = "C"

@dataclass
class MeetingEntry:
    """Single entry in meeting minutes."""
    entry_id: int
    timestamp_utc: datetime  # Stored as UTC internally
    speaker_id: str  # "Speaker 1", "Speaker 2", etc.
    speaker_role: SpeakerRole
    original_text: str
    translated_text: Optional[str] = None
    confidence: float = 0.0
    duration_ms: int = 0

    def get_local_timestamp(self) -> datetime:
        """Convert UTC timestamp to local time for display."""
        from datetime import timezone
        return self.timestamp_utc.astimezone(timezone.utc).astimezone()

    def to_minutes_format(self) -> str:
        """Format as meeting minutes line."""
        time_str = self.get_local_timestamp().strftime("%H:%M:%S")
        if self.translated_text:
            return f"[{time_str}] {self.speaker_id}:
{self.original_text}\n              → {self.translated_text}"
        return f"[{time_str}] {self.speaker_id}: {self.original_text}"

@dataclass
class MeetingSession:
    """Complete meeting session."""
    session_id: str
    start_time: datetime
    title: str = "Meeting Transcription"
    entries: List[MeetingEntry] = field(default_factory=list)

    def add_entry(self, entry: MeetingEntry):
        """Add entry and auto-assign speaker if not set."""
        entry.entry_id = len(self.entries) + 1
        self.entries.append(entry)

    def export_as_minutes(self) -> str:
        """Export as formatted meeting minutes."""
        lines = [
            f"# {self.title}",
            f"Date: {self.start_time.strftime('%Y-%m-%d %H:%M')}",
            f"Duration: {self.get_duration()} minutes",
            f"Total Entries: {len(self.entries)}",
            "",
```

```
            "## Meeting Minutes",
            ""
        ]

        current_speaker = None
        for entry in self.entries:
            if entry.speaker_id != current_speaker:
                lines.append(f"\n**{entry.speaker_id}:**")
                current_speaker = entry.speaker_id
            lines.append(f"[{entry.timestamp.strftime('%H:%M:%S')}]
{entry.original_text}")
            if entry.translated_text:
                lines.append(f"  → {entry.translated_text}")

        return "\n".join(lines)

    def get_duration(self) -> int:
        """Get meeting duration in minutes."""
        if not self.entries:
            return 0
        duration = self.entries[-1].timestamp - self.start_time
        return int(duration.total_seconds() / 60)
```

4.1.2 Meeting Minutes Display

File: src/gui/meeting/minutes_display.py

```
"""
Meeting minutes display widget with speaker timeline.
"""

from PySide6.QtWidgets import (
    QWidget, QVBoxLayout, QHBoxLayout, QLabel,
    QScrollArea, QFrame, QPushButton
)
from PySide6.QtCore import Qt, Signal
from PySide6.QtGui import QColor, QFont

class SpeakerBubble(QFrame):
    """Visual bubble for speaker entry with confidence-based styling."""

    # Speaker color mapping
    SPEAKER_COLORS = {
        "Speaker 1": "#4A90D9",  # Blue
        "Speaker 2": "#E85D75",  # Coral
        "Speaker 3": "#50C878",  # Green
        "Speaker 4": "#F5A623",  # Orange
        "Unknown": "#9B9B9B",    # Gray
    }

    # Confidence styling: border color indicates quality
```

```
    CONFIDENCE_HIGH = 0.8   # Green border
    CONFIDENCE_MED = 0.6    # Yellow border
    CONFIDENCE_LOW = 0.4    # Red border

    def __init__(self, entry: MeetingEntry, parent=None):
        super().__init__(parent)
        self.entry = entry
        self._setup_ui()

    def _get_confidence_style(self) -> str:
        """Get border color based on confidence score."""
        conf = self.entry.confidence
        if conf >= self.CONFIDENCE_HIGH:
            return "border: 2px solid #4CAF50;"  # Green
        elif conf >= self.CONFIDENCE_MED:
            return "border: 2px solid #FFC107;"  # Yellow
        elif conf >= self.CONFIDENCE_LOW:
            return "border: 2px solid #FF9800;"  # Orange
        else:
            return "border: 2px solid #F44336;"  # Red

    def _get_text_opacity(self) -> str:
        """Get text opacity based on confidence (low confidence =
fainter)."""
        conf = self.entry.confidence
        opacity = max(0.5, conf)  # Minimum 50% opacity
        return f"color: rgba(255, 255, 255, {opacity});"

    def _setup_ui(self):
        layout = QHBoxLayout(self)
        layout.setContentsMargins(10, 8, 10, 8)

        # Apply confidence-based border
        confidence_border = self._get_confidence_style()
        self.setStyleSheet(f"""
            SpeakerBubble {{
                background-color: #2D2D44;
                border-radius: 12px;
                {confidence_border}
            }}
        """)

        # Speaker avatar circle
        avatar = QLabel(self.entry.speaker_id[0])  # First letter
        avatar.setFixedSize(32, 32)
        avatar.setAlignment(Qt.AlignCenter)
        color = self.SPEAKER_COLORS.get(self.entry.speaker_id, "#9B9B9B")
        avatar.setStyleSheet(f"""
            background-color: {color};
            color: white;
            border-radius: 16px;
            font-weight: bold;
            font-size: 14px;
        """)
```

```
        layout.addWidget(avatar)

        # Content
        content = QVBoxLayout()

        # Header: Speaker name + time
        header = QHBoxLayout()
        speaker_label = QLabel(f"<b>{self.entry.speaker_id}</b>")
        time_label = QLabel(f"<span style='color: gray;'>
{self.entry.timestamp.strftime('%H:%M:%S')}</span>")
        header.addWidget(speaker_label)
        header.addWidget(time_label)
        header.addStretch()
        content.addLayout(header)

        # Original text
        orig_label = QLabel(self.entry.original_text)
        orig_label.setWordWrap(True)
        content.addWidget(orig_label)

        # Translation (if available)
        if self.entry.translated_text:
            trans_label = QLabel(f"→ {self.entry.translated_text}")
            trans_label.setWordWrap(True)
            trans_label.setStyleSheet("color: #666; font-style: italic;")
            content.addWidget(trans_label)

        layout.addLayout(content, 1)

class MinutesDisplayWidget(QScrollArea):
    """Scrollable meeting minutes display."""

    entry_selected = Signal(MeetingEntry)

    def __init__(self, parent=None):
        super().__init__(parent)
        self._entries: List[MeetingEntry] = []
        self._setup_ui()

    def _setup_ui(self):
        self.setWidgetResizable(True)

        container = QWidget()
        self._layout = QVBoxLayout(container)
        self._layout.setSpacing(8)
        self._layout.addStretch()

        self.setWidget(container)

    def add_entry(self, entry: MeetingEntry):
        """Add new meeting entry."""
        self._entries.append(entry)
        bubble = SpeakerBubble(entry)
```

```
        # Insert before stretch
        self._layout.insertWidget(self._layout.count() - 1, bubble)

        # Auto-scroll to bottom
        self.verticalScrollBar().setValue(
            self.verticalScrollBar().maximum()
        )

    def clear(self):
        """Clear all entries."""
        self._entries.clear()
        # Remove all widgets except stretch
        while self._layout.count() > 1:
            item = self._layout.takeAt(0)
            if item.widget():
                item.widget().deleteLater()

    def export_as_markdown(self, filepath: str) -> bool:
        """Export minutes as Markdown."""
        if not self._entries:
            return False

        session = MeetingSession(
            session_id="export",
            start_time=self._entries[0].timestamp,
            entries=self._entries
        )

        with open(filepath, 'w', encoding='utf-8') as f:
            f.write(session.export_as_minutes())

        return True
```

4.2 Speaker Recognition

Purpose: Identify and differentiate speakers in real-time.

> ⚠ **V1 Implementation Note:** Use `SimpleSpeakerDiarization` (turn-based) for MVP to ensure performance. AI-based diarization (embeddings) reserved for V2.

4.2.1 Speaker Diarization Engine (V1: Turn-Based)

File: `src/core/speaker/diarization.py`

V1 Strategy - Turn-Based Detection:

- Detect speaker turns using pause detection
- Assume alternating speakers for structured meetings
- Computationally free, no ML model required
- Works well for 2-4 person meetings with clear turn-taking

User-Configurable Speaker Count:

- End-user inputs expected number of speakers (2-8) before meeting starts
- System pre-allocates speaker slots with color coding
- Turn-based algorithm cycles through known speaker count
- Reduces ambiguity: "Which of the 3 speakers is talking?" vs "Is this a new speaker?"
- Allows manual correction: user can drag misattributed segments to correct speaker

V2 Strategy - AI Embeddings (Future):

- Use pyannote.audio or speechbrain ECAPA-TDNN
- Run in separate low-priority thread post-transcription
- Higher accuracy but resource-intensive
- Benefits from known speaker count constraint

```
"""
Speaker diarization using edge-based approach with user-configurable
speaker count.
Uses turn-based detection for V1 (MVP) - computationally efficient.
"""

import numpy as np
from typing import List, Dict, Optional, Tuple
from dataclasses import dataclass
from collections import deque
import hashlib

@dataclass
class SpeakerSegment:
    """Segment attributed to a speaker."""
    speaker_id: str
    start_time: float
    end_time: float
    embedding: np.ndarray
    confidence: float

class SpeakerDiarization:
    """
    Real-time speaker diarization for meeting transcription.

    Approach:
    1. Extract voice embedding from audio segment
    2. Compare with known speaker embeddings (cosine similarity)
    3. Assign to existing speaker or create new one
    4. Maintain rolling window for speaker consistency
    """

    def __init__(
        self,
        max_speakers: int = 4,
        similarity_threshold: float = 0.75,
        embedding_dim: int = 256
```

```
    ):
        self.max_speakers = max_speakers
        self.similarity_threshold = similarity_threshold
        self.embedding_dim = embedding_dim

        # Known speakers: speaker_id -> list of embeddings
        self._speakers: Dict[str, List[np.ndarray]] = {}
        self._speaker_counter = 0

        # Rolling window for recent assignments
        self._recent_segments: deque = deque(maxlen=10)

    def process_segment(
        self,
        audio_segment: np.ndarray,
        start_time: float,
        end_time: float
    ) -> SpeakerSegment:
        """
        Process audio segment and identify speaker.

        Args:
            audio_segment: Audio samples (numpy array)
            start_time: Segment start time in seconds
            end_time: Segment end time in seconds

        Returns:
            SpeakerSegment with identified speaker
        """
        # Extract embedding (simplified - use actual model in production)
        embedding = self._extract_embedding(audio_segment)

        # Find best matching speaker
        speaker_id, confidence = self._identify_speaker(embedding)

        # Store embedding for speaker
        if speaker_id not in self._speakers:
            self._speakers[speaker_id] = []
        self._speakers[speaker_id].append(embedding)

        # Keep only recent embeddings per speaker
        self._speakers[speaker_id] = self._speakers[speaker_id][-10:]

        segment = SpeakerSegment(
            speaker_id=speaker_id,
            start_time=start_time,
            end_time=end_time,
            embedding=embedding,
            confidence=confidence
        )

        self._recent_segments.append(segment)
        return segment
```

```
    def _extract_embedding(self, audio: np.ndarray) -> np.ndarray:
        """
        Extract voice embedding from audio.

        Note: In production, use a proper speaker embedding model like:
        - speechbrain/ecapa-tdnn
        - pyannote/embedding
        - or lightweight edge model
        """
        # Placeholder: Use audio features as embedding
        # In production, replace with actual speaker embedding model
        features = self._extract_audio_features(audio)

        # Normalize
        norm = np.linalg.norm(features)
        if norm > 0:
            features = features / norm

        return features

    def _extract_audio_features(self, audio: np.ndarray) -> np.ndarray:
        """Extract basic audio features for embedding."""
        # Simple feature extraction (placeholder)
        # Real implementation would use MFCC, pitch, etc.

        # Pad or truncate to fixed length
        target_length = 16000  # 1 second at 16kHz
        if len(audio) < target_length:
            audio = np.pad(audio, (0, target_length - len(audio)))
        else:
            audio = audio[:target_length]

        # Compute spectrogram-like features
        fft = np.fft.rfft(audio)
        magnitude = np.abs(fft)

        # Downsample to embedding dimension
        embedding = np.interp(
            np.linspace(0, len(magnitude), self.embedding_dim),
            np.arange(len(magnitude)),
            magnitude
        )

        return embedding

    def _identify_speaker(self, embedding: np.ndarray) -> Tuple[str,
float]:
        """
        Identify speaker from embedding.

        Returns:
            (speaker_id, confidence)
        """
        if not self._speakers:
```

```
            return self._create_new_speaker(embedding), 1.0

        best_match = None
        best_similarity = 0.0

        for speaker_id, embeddings in self._speakers.items():
            # Compare with average embedding for speaker
            avg_embedding = np.mean(embeddings, axis=0)
            similarity = self._cosine_similarity(embedding, avg_embedding)

            if similarity > best_similarity:
                best_similarity = similarity
                best_match = speaker_id

        # Check if similarity exceeds threshold
        if best_similarity >= self.similarity_threshold:
            return best_match, best_similarity

        # Create new speaker if under limit
        if len(self._speakers) < self.max_speakers:
            return self._create_new_speaker(embedding), 1.0

        # Otherwise, assign to closest match
        return best_match, best_similarity

    def _create_new_speaker(self, embedding: np.ndarray) -> str:
        """Create new speaker entry."""
        self._speaker_counter += 1
        speaker_id = f"Speaker {self._speaker_counter}"
        self._speakers[speaker_id] = [embedding]
        return speaker_id

    def _cosine_similarity(self, a: np.ndarray, b: np.ndarray) -> float:
        """Compute cosine similarity between two vectors."""
        return np.dot(a, b) / (np.linalg.norm(a) * np.linalg.norm(b))

    def get_speaker_stats(self) -> Dict[str, dict]:
        """Get statistics for each speaker."""
        stats = {}
        for speaker_id in self._speakers:
            speaker_segments = [
                s for s in self._recent_segments
                if s.speaker_id == speaker_id
            ]
            total_time = sum(
                s.end_time - s.start_time for s in speaker_segments
            )
            stats[speaker_id] = {
                "segment_count": len(speaker_segments),
                "total_time_seconds": total_time
            }
        return stats

    def reset(self):
```

```
        """Reset all speaker data."""
        self._speakers.clear()
        self._speaker_counter = 0
        self._recent_segments.clear()


class SimpleSpeakerDiarization(SpeakerDiarization):
    """
    Simplified speaker diarization using turn-based detection.

    Alternative approach: Detect speaker turns using pause detection
    and assume alternating speakers for dialogue.
    """

    def __init__(self, max_speakers: int = 2, **kwargs):
        super().__init__(max_speakers=max_speakers, **kwargs)
        self._last_speaker = None
        self._turn_counter = 0

    def process_segment(
        self,
        audio_segment: np.ndarray,
        start_time: float,
        end_time: float,
        is_new_turn: bool = False
    ) -> SpeakerSegment:
        """
        Process with turn detection.

        Args:
            is_new_turn: True if this is detected as a new speaker turn
        """
        if is_new_turn or self._last_speaker is None:
            self._turn_counter += 1
            speaker_id = f"Speaker {(self._turn_counter %
self.max_speakers) + 1}"
        else:
            speaker_id = self._last_speaker

        self._last_speaker = speaker_id

        return SpeakerSegment(
            speaker_id=speaker_id,
            start_time=start_time,
            end_time=end_time,
            embedding=np.array([]),
            confidence=0.8 if is_new_turn else 0.9
        )
```

4.2.2 Speaker-Aware Pipeline Integration

File: `src/core/pipeline/meeting_pipeline.py`

```
"""
Meeting transcription pipeline with speaker diarization.
"""

from typing import Optional, Callable
from dataclasses import dataclass
from datetime import datetime

from src.core.pipeline.orchestrator import TranslationPipeline,
PipelineConfig
from src.core.speaker.diarization import SpeakerDiarization,
SpeakerSegment
from src.gui.meeting.minutes_model import MeetingEntry, MeetingSession

@dataclass
class MeetingConfig:
    """Configuration for meeting transcription."""
    enable_speaker_diarization: bool = True
    max_speakers: int = 4
    enable_translation: bool = True
    source_language: str = "auto"
    target_language: Optional[str] = None
    session_title: str = "Meeting Transcription"

class MeetingPipeline:
    """
    End-to-end meeting transcription pipeline.

    Combines:
    - Audio capture
    - Voice Activity Detection
    - Speaker Diarization
    - Speech Recognition (ASR)
    - Translation (optional)
    - Meeting minutes generation
    """

    def __init__(
        self,
        config: MeetingConfig,
        asr_pipeline_config: PipelineConfig
    ):
        self.config = config
        self.asr_config = asr_pipeline_config

        # Initialize components
        self.asr_pipeline = TranslationPipeline(asr_pipeline_config)
        self.diarization =
SpeakerDiarization(max_speakers=config.max_speakers)

        # Session state
        self.session: Optional[MeetingSession] = None
```

```
        self._on_entry_callback: Optional[Callable[[MeetingEntry], None]]
= None
        self._is_running = False

    def start_session(self, title: Optional[str] = None):
        """Start new meeting session."""
        import uuid

        self.session = MeetingSession(
            session_id=str(uuid.uuid4()),
            start_time=datetime.now(),
            title=title or self.config.session_title
        )

        self.diarization.reset()
        self._is_running = True

        # Start ASR pipeline with custom callback
        self.asr_pipeline.set_output_callback(self._on_asr_output)
        self.asr_pipeline.start()

    def _on_asr_output(self, text: str, is_final: bool = True):
        """Handle ASR output with speaker attribution."""
        if not self.session or not text.strip():
            return

        # Get current audio timing
        current_time = datetime.now()

        # Perform speaker diarization
        # Note: In real implementation, pass actual audio segment
        speaker_segment = self.diarization.process_segment(
            audio_segment=np.array([]),  # Placeholder
            start_time=current_time.timestamp(),
            end_time=current_time.timestamp()
        )

        # Translate if enabled
        translated = None
        if self.config.enable_translation and self.config.target_language:
            translated = self._translate_text(text)

        # Create meeting entry
        entry = MeetingEntry(
            entry_id=0,  # Will be auto-assigned
            timestamp=current_time,
            speaker_id=speaker_segment.speaker_id,
            speaker_role=self._infer_speaker_role(speaker_segment),
            original_text=text,
            translated_text=translated,
            confidence=speaker_segment.confidence,
            duration_ms=0  # Calculate from audio segment
        )
```

```
        self.session.add_entry(entry)

        # Notify callback
        if self._on_entry_callback:
            self._on_entry_callback(entry)

    def _translate_text(self, text: str) -> str:
        """Translate text using pipeline translator."""
        # Use existing translation pipeline
        # Implementation depends on existing translation system
        return text  # Placeholder

    def _infer_speaker_role(self, segment: SpeakerSegment) -> SpeakerRole:
        """Infer speaker role from segment."""
        from src.gui.meeting.minutes_model import SpeakerRole

        # Simple mapping based on speaker ID
        if segment.speaker_id == "Speaker 1":
            return SpeakerRole.HOST
        return SpeakerRole.PARTICIPANT

    def set_entry_callback(self, callback: Callable[[MeetingEntry],
None]):
        """Set callback for new entries."""
        self._on_entry_callback = callback

    def stop_session(self) -> MeetingSession:
        """Stop session and return complete minutes."""
        self._is_running = False
        self.asr_pipeline.stop()
        return self.session

    def export_session(self, filepath: str, format: str = "markdown"):
        """Export session to file."""
        if not self.session:
            return False

        if format == "markdown":
            content = self.session.export_as_minutes()
            with open(filepath, 'w', encoding='utf-8') as f:
                f.write(content)
        elif format == "json":
            import json
            # Convert to JSON
            data = {
                "session_id": self.session.session_id,
                "title": self.session.title,
                "start_time": self.session.start_time.isoformat(),
                "entries": [
                    {
                        "timestamp": e.timestamp.isoformat(),
                        "speaker": e.speaker_id,
                        "text": e.original_text,
                        "translation": e.translated_text,
```

```
                        "confidence": e.confidence
                    }
                    for e in self.session.entries
                ]
            }
            with open(filepath, 'w', encoding='utf-8') as f:
                json.dump(data, f, indent=2, ensure_ascii=False)

        return True
```

4.3 Modern GUI Theme

Purpose: Replace default Qt6 styling with a modern, professional theme.

4.3.1 Modern Theme System

File: `src/gui/theme/modern_theme.py`

```
"""
Modern UI theme for VoiceTranslate Pro.
Clean, professional design with dark/light modes.
"""

from PySide6.QtWidgets import QApplication
from PySide6.QtCore import Qt
from PySide6.QtGui import QColor, QPalette, QFont
from enum import Enum

class ThemeMode(Enum):
    DARK = "dark"
    LIGHT = "light"
    AUTO = "auto"

class ModernTheme:
    """
    Modern theme system with customizable colors and fonts.
    """

    # Color palette
    COLORS = {
        ThemeMode.DARK: {
            "background": "#1E1E2E",
            "surface": "#2D2D44",
            "surface_hover": "#3D3D5C",
            "primary": "#6C5DD3",
            "primary_hover": "#5B4EC2",
            "secondary": "#00D4AA",
            "accent": "#FF6B6B",
            "text_primary": "#FFFFFF",
            "text_secondary": "#B4B4BE",
```

```
            "border": "#3D3D5C",
            "success": "#4CAF50",
            "warning": "#FFC107",
            "error": "#F44336",
        },
        ThemeMode.LIGHT: {
            "background": "#F5F5F7",
            "surface": "#FFFFFF",
            "surface_hover": "#F0F0F5",
            "primary": "#6C5DD3",
            "primary_hover": "#5B4EC2",
            "secondary": "#00B894",
            "accent": "#FF6B6B",
            "text_primary": "#1A1A2E",
            "text_secondary": "#6B6B7B",
            "border": "#E0E0E8",
            "success": "#4CAF50",
            "warning": "#FFC107",
            "error": "#F44336",
        }
    }

    # Typography
    FONTS = {
        "heading": QFont("Inter", 24, QFont.Bold),
        "subheading": QFont("Inter", 18, QFont.DemiBold),
        "body": QFont("Inter", 13),
        "body_small": QFont("Inter", 11),
        "caption": QFont("Inter", 10),
        "monospace": QFont("JetBrains Mono", 12),
    }

    def __init__(self, mode: ThemeMode = ThemeMode.DARK):
        self.mode = mode
        self.colors = self.COLORS[mode]

    def apply_to_app(self, app: QApplication):
        """Apply theme to entire application."""
        app.setStyleSheet(self.get_stylesheet())

        # Set default font
        app.setFont(self.FONTS["body"])

    def get_stylesheet(self) -> str:
        """Generate QSS stylesheet."""
        c = self.colors

        return f"""
        /* Main Window */
        QMainWindow {{
            background-color: {c['background']};
        }}

        /* Central Widget */
```

```
        QWidget {{
            background-color: {c['background']};
            color: {c['text_primary']};
            font-family: 'Inter', -apple-system, BlinkMacSystemFont, sans-
serif;
        }}

        /* Buttons */
        QPushButton {{
            background-color: {c['surface']};
            color: {c['text_primary']};
            border: 1px solid {c['border']};
            border-radius: 8px;
            padding: 10px 20px;
            font-weight: 500;
            min-width: 80px;
        }}

        QPushButton:hover {{
            background-color: {c['surface_hover']};
        }}

        QPushButton:pressed {{
            background-color: {c['primary']};
        }}

        QPushButton:disabled {{
            background-color: {c['surface']};
            color: {c['text_secondary']};
            border-color: {c['border']};
        }}

        QPushButton#primary {{
            background-color: {c['primary']};
            color: white;
            border: none;
        }}

        QPushButton#primary:hover {{
            background-color: {c['primary_hover']};
        }}

        /* Combo Box */
        QComboBox {{
            background-color: {c['surface']};
            color: {c['text_primary']};
            border: 1px solid {c['border']};
            border-radius: 6px;
            padding: 8px 12px;
            min-width: 120px;
        }}

        QComboBox:hover {{
            border-color: {c['primary']};
```

```
        }}

        QComboBox::drop-down {{
            border: none;
            width: 24px;
        }}

        QComboBox QAbstractItemView {{
            background-color: {c['surface']};
            color: {c['text_primary']};
            selection-background-color: {c['primary']};
            border: 1px solid {c['border']};
            border-radius: 6px;
        }}

        /* Text Edit */
        QTextEdit {{
            background-color: {c['surface']};
            color: {c['text_primary']};
            border: 1px solid {c['border']};
            border-radius: 8px;
            padding: 12px;
            font-size: 14px;
            line-height: 1.5;
        }}

        QTextEdit:focus {{
            border-color: {c['primary']};
        }}

        /* Group Box */
        QGroupBox {{
            background-color: {c['surface']};
            border: 1px solid {c['border']};
            border-radius: 12px;
            margin-top: 16px;
            padding-top: 16px;
            font-weight: 600;
        }}

        QGroupBox::title {{
            subcontrol-origin: margin;
            left: 16px;
            padding: 0 8px;
            color: {c['text_secondary']};
        }}

        /* Labels */
        QLabel {{
            color: {c['text_primary']};
        }}

        QLabel#heading {{
            font-size: 24px;
```

```
            font-weight: bold;
            color: {c['text_primary']};
        }}

        QLabel#subheading {{
            font-size: 16px;
            font-weight: 600;
            color: {c['text_secondary']};
        }}

        QLabel#caption {{
            font-size: 12px;
            color: {c['text_secondary']};
        }}

        /* Scroll Area */
        QScrollArea {{
            border: none;
            background-color: transparent;
        }}

        QScrollBar:vertical {{
            background-color: transparent;
            width: 8px;
            margin: 0;
        }}

        QScrollBar::handle:vertical {{
            background-color: {c['border']};
            border-radius: 4px;
            min-height: 32px;
        }}

        QScrollBar::handle:vertical:hover {{
            background-color: {c['text_secondary']};
        }}

        /* Status Bar */
        QStatusBar {{
            background-color: {c['surface']};
            color: {c['text_secondary']};
            border-top: 1px solid {c['border']};
        }}

        /* Progress Bar */
        QProgressBar {{
            background-color: {c['surface']};
            border: none;
            border-radius: 4px;
            height: 6px;
            text-align: center;
        }}

        QProgressBar::chunk {{
```

```
            background-color: {c['primary']};
            border-radius: 4px;
        }}

        /* Check Box */
        QCheckBox {{
            color: {c['text_primary']};
            spacing: 8px;
        }}

        QCheckBox::indicator {{
            width: 18px;
            height: 18px;
            border-radius: 4px;
            border: 2px solid {c['border']};
            background-color: {c['surface']};
        }}

        QCheckBox::indicator:checked {{
            background-color: {c['primary']};
            border-color: {c['primary']};
        }}

        /* Line Edit */
        QLineEdit {{
            background-color: {c['surface']};
            color: {c['text_primary']};
            border: 1px solid {c['border']};
            border-radius: 6px;
            padding: 8px 12px;
        }}

        QLineEdit:focus {{
            border-color: {c['primary']};
        }}
        """

    def get_card_style(self) -> str:
        """Get style for card containers."""
        c = self.colors
        return f"""
            background-color: {c['surface']};
            border-radius: 12px;
            border: 1px solid {c['border']};
        """


class ThemeManager:
    """Manages theme application-wide."""

    _instance = None

    def __new__(cls):
        if cls._instance is None:
```

```
            cls._instance = super().__new__(cls)
            cls._instance._theme = ModernTheme(ThemeMode.DARK)
        return cls._instance

    def set_theme(self, mode: ThemeMode):
        """Change theme."""
        self._theme = ModernTheme(mode)

        # Reapply to active application
        app = QApplication.instance()
        if app:
            self._theme.apply_to_app(app)

    def get_theme(self) -> ModernTheme:
        """Get current theme."""
        return self._theme
```

Phase 5: Debug Logging System

Purpose: Comprehensive logging for troubleshooting user issues.

6.1 Debug Logger

File: `src/core/utils/debug_logger.py`

```
"""
Comprehensive debug logging system using loguru.
Generates detailed logs for troubleshooting.
"""

import sys
import os
import platform
import traceback
from datetime import datetime
from pathlib import Path
from typing import Optional
from dataclasses import dataclass, asdict
import json

from loguru import logger

@dataclass
class SystemInfo:
    """System information for debugging."""
    platform: str
    platform_version: str
    python_version: str
    cpu_count: int
    memory_gb: float
```

```
    gpu_info: str = "Unknown"

class DebugLogger:
    """
    Comprehensive debug logging for VoiceTranslate Pro using loguru.

    Features:
    - Rotating log files
    - System info capture
    - Component-level logging
    - Crash dump generation
    - Privacy mode (disable text logging)
    - Runtime log level switching
    """

    LOG_DIR = Path.home() / ".voicetranslate" / "logs"
    MAX_BYTES = "10 MB"
    BACKUP_COUNT = 5

    def __init__(self, name: str = "voicetranslate"):
        self.name = name
        self.log_file = None

        # Privacy mode - when True, don't log transcript text
        self.privacy_mode = False

        self._setup_handlers()
        self._log_system_info()

    def _setup_handlers(self):
        """Setup loguru handlers."""
        self.LOG_DIR.mkdir(parents=True, exist_ok=True)

        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        self.log_file = self.LOG_DIR / f"{self.name}_{timestamp}.log"

        # Remove default handler
        logger.remove()

        # File handler with rotation
        logger.add(
            self.log_file,
            rotation=self.MAX_BYTES,
            retention=self.BACKUP_COUNT,
            encoding='utf-8',
            level="DEBUG",
            format="{time:YYYY-MM-DD HH:mm:ss} | {level:<8} | {name} |
{function}:{line} | {message}"
        )

        # Console handler
        logger.add(
            sys.stdout,
            level="INFO",
```

```
            format="{level}: {message}"
        )

    def set_log_level(self, level: str):
        """Change log level at runtime (INFO or DEBUG)."""
        # loguru requires re-adding handlers to change levels
        logger.remove()
        self._setup_handlers()
        logger.info(f"Log level changed to {level}")

    def set_privacy_mode(self, enabled: bool):
        """Enable privacy mode - disables transcript text in logs."""
        self.privacy_mode = enabled
        self.logger.info(f"Privacy mode {'enabled' if enabled else
'disabled'}")

    def _log_system_info(self):
        """Log system information."""
        import psutil

        info = SystemInfo(
            platform=platform.system(),
            platform_version=platform.version(),
            python_version=platform.python_version(),
            cpu_count=os.cpu_count(),
            memory_gb=psutil.virtual_memory().total / (1024**3),
        )

        # Try to get GPU info
        try:
            import torch
            if torch.cuda.is_available():
                info.gpu_info = f"CUDA: {torch.cuda.get_device_name(0)}"
            elif torch.backends.mps.is_available():
                info.gpu_info = "Apple MPS (Metal)"
        except:
            pass

        logger.info("=" * 60)
        logger.info("VoiceTranslate Pro - Debug Log")
        logger.info(f"Log File: {self.log_file}")
        logger.info("=" * 60)
        logger.info("System Information:")
        for key, value in asdict(info).items():
            logger.info(f"  {key}: {value}")
        logger.info("=" * 60)

    def log_component_init(self, component: str, config: dict = None):
        """Log component initialization."""
        logger.info(f"[INIT] {component}")
        if config:
            logger.debug(f"  Config: {json.dumps(config, indent=2)}")

    def log_audio_device(self, device_info: dict):
```

```
        """Log audio device selection."""
        logger.info(f"[AUDIO] Device: {device_info.get('name',
'Unknown')}")
        logger.debug(f"  Info: {json.dumps(device_info, indent=2)}")

    def log_asr_result(self, text: str, confidence: float, latency_ms:
float):
        """Log ASR result. Respects privacy mode."""
        logger.info(f"[ASR] Confidence: {confidence:.2f}, Latency:
{latency_ms:.0f}ms")
        if not self.privacy_mode:
            logger.debug(f"  Text: {text[:100]}...")
        else:
            logger.debug(f"  Text: [REDACTED - Privacy Mode]")

    def log_translation(self, source: str, target: str, latency_ms:
float):
        """Log translation result. Respects privacy mode."""
        logger.info(f"[TRANSLATE] Latency: {latency_ms:.0f}ms")
        if not self.privacy_mode:
            logger.debug(f"  Source: {source[:100]}...")
            logger.debug(f"  Target: {target[:100]}...")
        else:
            logger.debug(f"  Source/Target: [REDACTED - Privacy Mode]")

    def log_error(self, error: Exception, context: str = ""):
        """Log error with traceback."""
        logger.error(f"[ERROR] {context}: {str(error)}")
        logger.error(traceback.format_exc())

    def log_performance(self, metric_name: str, value: float, unit: str =
"ms"):
        """Log performance metric."""
        logger.info(f"[PERF] {metric_name}: {value:.2f}{unit}")

    def generate_crash_dump(self, exc_info: tuple = None) -> Path:
        """Generate crash dump file."""
        dump_file = self.LOG_DIR /
f"crash_{datetime.now().strftime('%Y%m%d_%H%M%S')}.txt"

        with open(dump_file, 'w') as f:
            f.write("VoiceTranslate Pro - Crash Report\n")
            f.write("=" * 60 + "\n")
            f.write(f"Time: {datetime.now().isoformat()}\n")
            f.write(f"Log File: {self.log_file}\n")
            f.write("\nException:\n")

            if exc_info:
                f.write(''.join(traceback.format_exception(*exc_info)))
            else:
                f.write(traceback.format_exc())

            f.write("\nRecent Log Entries:\n")
            # Include last 50 log lines if possible
```

```
        logger.critical(f"Crash dump generated: {dump_file}")
        return dump_file

    def get_log_location(self) -> str:
        """Get log file location for user."""
        return str(self.log_file)


# Global logger instance
_debug_logger: Optional[DebugLogger] = None

def get_debug_logger() -> DebugLogger:
    """Get or create global debug logger."""
    global _debug_logger
    if _debug_logger is None:
        _debug_logger = DebugLogger()
    return _debug_logger
```

5.1.2 ModelManager (NEW - Critical for Portable Distribution)

File: `src/core/utils/model_manager.py`

Purpose: Handle model downloads on first run without installer.

```
"""
Model download and management system.
Handles large model downloads with progress, retries, and verification.
"""

import hashlib
import requests
from pathlib import Path
from typing import Optional, Callable
from dataclasses import dataclass
import json

@dataclass
class ModelConfig:
    """Configuration for a downloadable model."""
    name: str
    url: str
    mirror_url: Optional[str] = None
    checksum: str = ""  # SHA256
    size_mb: int = 0
    local_path: Path = Path.home() / ".voicetranslate" / "models"

class ModelManager:
    """
    Manages AI model downloads and verification.
```

```
    Features:
    - Progress tracking with callbacks
    - Automatic retry on failure
    - Checksum verification
    - Mirror fallback
    - Resume partial downloads
    """

    def __init__(self, progress_callback: Optional[Callable[[float, int],
None]] = None):
        """
        Args:
            progress_callback: Function(percent, bytes_per_sec) called
during download
        """
        self.progress_callback = progress_callback
        self.models_dir = Path.home() / ".voicetranslate" / "models"
        self.models_dir.mkdir(parents=True, exist_ok=True)

    def download_model(self, config: ModelConfig, force: bool = False) ->
bool:
        """
        Download model with progress tracking and verification.

        Args:
            config: Model configuration
            force: Re-download even if exists

        Returns:
            True if successful
        """
        local_file = self.models_dir / config.name

        # Check if already downloaded
        if local_file.exists() and not force:
            if self._verify_checksum(local_file, config.checksum):
                logger.info(f"Model {config.name} already exists and
verified")
                return True
            else:
                logger.warning(f"Model {config.name} checksum mismatch,
re-downloading")

        # Try primary URL first, then mirror
        urls = [config.url]
        if config.mirror_url:
            urls.append(config.mirror_url)

        for url in urls:
            try:
                if self._download_with_progress(url, local_file,
config.size_mb):
                    if self._verify_checksum(local_file, config.checksum):
```

```
                            logger.info(f"Model {config.name} downloaded and
verified")
                            return True
                        else:
                            logger.error(f"Checksum verification failed for
{config.name}")
                            local_file.unlink(missing_ok=True)
                except Exception as e:
                    logger.error(f"Failed to download from {url}: {e}")
                    continue

        return False

    def _download_with_progress(self, url: str, dest: Path, expected_mb:
int) -> bool:
        """Download file with progress tracking."""
        response = requests.get(url, stream=True, timeout=30)
        response.raise_for_status()

        total_size = int(response.headers.get('content-length', 0))
        downloaded = 0

        with open(dest, 'wb') as f:
            for chunk in response.iter_content(chunk_size=8192):
                if chunk:
                    f.write(chunk)
                    downloaded += len(chunk)

                    if self.progress_callback and total_size > 0:
                        percent = (downloaded / total_size) * 100
                        # Calculate speed (simplified)
                        self.progress_callback(percent, 0)

        return True

    def _verify_checksum(self, filepath: Path, expected: str) -> bool:
        """Verify SHA256 checksum of file."""
        if not expected:
            return True  # No checksum provided, skip verification

        sha256_hash = hashlib.sha256()
        with open(filepath, "rb") as f:
            for byte_block in iter(lambda: f.read(4096), b""):
                sha256_hash.update(byte_block)

        return sha256_hash.hexdigest() == expected

    def get_model_path(self, name: str) -> Optional[Path]:
        """Get path to model if it exists and is valid."""
        path = self.models_dir / name
        if path.exists():
            return path
        return None
```

```
    def is_model_available(self, name: str) -> bool:
        """Check if model is downloaded and ready to use."""
        return self.get_model_path(name) is not None
```

Integration with First Run Wizard:

```
# In First Run Wizard, Step 3: Model Download
class ModelDownloadPage(QWidget):
    def __init__(self):
        self.progress_bar = QProgressBar()
        self.status_label = QLabel("Ready to download")
        self.retry_button = QPushButton("Retry")
        self.retry_button.hide()

    def start_download(self):
        self.manager =
ModelManager(progress_callback=self._update_progress)

        # Download Whisper model
        config = ModelConfig(
            name="whisper-base.pt",
            url="https://huggingface.co/.../model.pt",
            mirror_url="https://mirror.example.com/.../model.pt",
            checksum="sha256_hash_here",
            size_mb=150
        )

        success = self.manager.download_model(config)
        if success:
            self.status_label.setText("✅ Download complete!")
        else:
            self.status_label.setText("❌ Download failed. Check
connection.")
            self.retry_button.show()
```

Implementation Timeline (REVIEW PENDING)

> ⚠ **IMPORTANT:** This timeline is **TENTATIVE** and subject to change based on PoC results. **DO NOT START** until PoC is complete and approved.

Phase PoC: Proof of Concept (Week 0)

Status: 📋 **For Review** - Must complete before implementation
Location: All work in `poc/` folder (isolated from `src/`)

Day	Task	Location	Deliverable

Day	Task	Location	Deliverable
1-2	PoC 1: PyQt-Fluent-Widgets compatibility	`poc/poc1_fluent_widgets/`	Test report with go/no-go decision
3-4	PoC 2: Speaker diarization integration	`poc/poc2_speaker_diarization/`	Latency benchmarks
5	PoC 3: Data model coexistence	`poc/poc3_data_model/`	Integration validation
6	PoC 4: Model download & management	`poc/poc4_model_download/`	Download robustness test
7	PoC review and decision	N/A	Approved/revised roadmap

PoC Output Requirements: Each PoC folder must contain:

- `README.md` - How to run the tests
- `results.md` - Findings, benchmarks, and recommendations
- Test scripts (`.py` files)
- Any copied code from `src/` (do not modify original)

Decision Gates:

- ✅ **All PoCs Pass:** Proceed with full roadmap (8 weeks)
- ⚠ **Partial Pass:** Revise scope (6-8 weeks with reduced features)
- ❌ **PoC Fails:** Redesign approach (timeline TBD)

Phase 4: Meeting Mode Implementation (Weeks 1-6)

Prerequisite: PoC approved

Week 1-2: Foundation

- Day 1-3: Meeting minutes data model (UTC timestamps) - **separate from existing model**
- Day 4-5: Create `MeetingWindow` class (new window, existing app)
- Day 6-7: Basic meeting display widget (speaker bubbles)
- Day 8-10: Export functionality (Markdown, JSON)

Week 3-4: Speaker Recognition

- Day 1-2: Implement `SimpleSpeakerDiarization` with user-configurable speaker count
- Day 3-4: Speaker legend with rename functionality
- Day 5-6: Integration with existing pipeline via QThread
- Day 7-10: Speaker visualization in UI

Week 5: UI Polish (Conditional on PoC Result)

- **If PoC 1 PASSED:** Apply PyQt-Fluent-Widgets to Meeting Mode only
- **If PoC 1 FAILED:** Implement custom QSS theme
- Day 1-2: Theme system
- Day 3-4: Apply theme to Meeting Mode components
- Day 5: First Run Wizard (Meeting Mode only)
- Day 6-7: Audio test function with visual level meter

Week 6: Integration

- Day 1-2: Mode switcher (Translation Mode ↔ Meeting Mode)
- Day 3-4: Shared audio device selection with test
- Day 5-7: Integration testing, bug fixes

Phase 5: Debug Logging, Model Management & Polish (Weeks 7-8)

> **Scope Change:** Without installer, focus shifts to Model Management and Stability

Week 7: Debug Logging & Robust Model Downloader

- Day 1-2: Debug logging system with privacy controls
  - Auto-create `~/.voicetranslate/logs` directory
  - Handle PermissionError gracefully
  - Privacy mode for transcript redaction
- Day 3-5: **ModelManager class (NEW PRIORITY)**
  - Download progress bars (% and MB/s)
  - Retry logic for failed downloads
  - Checksum verification (SHA256)
  - Mirror support for blocked regions
  - Offline mode (launch without models)

Week 8: Performance Tuning & Portable Distribution Test

- Day 1-2: Performance tuning
  - Memory leak checks
  - CPU usage optimization
  - Audio buffer tuning
- Day 3-4: **Portable Distribution Test**
  - Verify app runs on clean machines without installation
  - Test model download on slow connections
  - Verify log directory creation
- Day 5-6: Accessibility testing (WCAG AA contrast ratios)
- Day 7: Final integration testing, bug fixes

Key Deliverables:

- ☐ Model downloads work reliably without installer
- ☐ App handles network failures gracefully
- ☐ Portable executable runs on clean Windows/Mac

- [ ] Debug logs write to user-writable directories

Go/No-Go Checklist

Before starting implementation, verify:

PoC Phase (Week 0) - Isolated in `poc/` Folder

- [ ] **PoC 1 completed:** PyQt-Fluent-Widgets compatibility (`poc/poc1_fluent_widgets/`)
- [ ] **PoC 2 completed:** Speaker diarization latency acceptable (`poc/poc2_speaker_diarization/`)
- [ ] **PoC 3 completed:** Data models can coexist (`poc/poc3_data_model/`)
- [ ] **PoC 4 completed:** Model download logic tested (`poc/poc4_model_download/`)
  - [ ] Resume interrupted downloads
  - [ ] Retry on network failure
  - [ ] Checksum verification
  - [ ] Permission handling
- [ ] All PoC results documented in respective `results.md` files

Strategic Approval

- [ ] Stakeholder approval on incremental approach (Option A)
- [ ] Resource allocation confirmed (8 weeks)
- [ ] Fallback plan documented (if PoC partially fails)
  - PyQt-Fluent-Widgets fails → Use QSS theme
  - Speaker diarization fails → Delay to V2
  - Model download fails → Require manual download

Pre-Implementation

- [ ] ModelManager class design reviewed
- [ ] First Run Wizard flow approved
- [ ] Audio Test "Save Sample" feature approved

File Structure

Implementation (After PoC Approval)

```
src/                              # ⛔ DO NOT MODIFY during PoC
├── gui/
│   ├── theme/
│   │   ├── __init__.py
│   │   └── modern_theme.py       # Modern UI theme (Meeting Mode only)
│   ├── meeting/
│   │   ├── __init__.py
│   │   ├── minutes_model.py      # Meeting data model (NEW)
│   │   ├── minutes_display.py    # Meeting display widget (NEW)
│   │   └── meeting_window.py     # Meeting Mode window (NEW)
```

```
│   ├── audio_test.py             # Audio test dialog (shared)
│   └── main.py                   # EXISTING - Translation Mode
(unchanged)
├── core/
│   ├── speaker/
│   │   ├── __init__.py
│   │   ├── diarization.py        # Speaker recognition (NEW)
│   │   └── embedding.py          # Voice embedding model (V2)
│   ├── pipeline/
│   │   ├── orchestrator.py       # EXISTING (unchanged)
│   │   └── meeting_pipeline.py   # Meeting-aware pipeline (NEW)
│   └── utils/
│       ├── debug_logger.py       # Debug logging (shared)
│       └── model_manager.py      # Model download & management (NEW)
```

PoC Phase (Before Implementation)

```
poc/                              # ✅ PoC WORK DIRECTORY
├── README.md                     # PoC setup instructions
├── requirements.txt              # PoC dependencies
├── poc1_fluent_widgets/          # PoC 1: UI Compatibility
│   ├── test_fluent_theme.py
│   ├── test_threading.py
│   └── results.md
├── poc2_speaker_diarization/     # PoC 2: Speaker Detection
│   ├── speaker_test.py
│   ├── test_latency.py
│   └── results.md
├── poc3_data_model/              # PoC 3: Model Coexistence
│   ├── test_coexistence.py
│   └── results.md
└── poc4_model_download/          # PoC 4: Model Management
    ├── model_manager_test.py
    ├── test_downloader.py
    └── results.md
```

File Status Legend:

- **NEW:** Created for this feature
- **EXISTING:** Unchanged from current version
- **(shared):** Used by both Translation and Meeting modes

📋 Review Checklist for Stakeholders

Before approving this roadmap, please confirm:

✅ Executive Summary

Status: Approved for PoC Phase
Impact of Change: Removing installer eliminates code signing, notarization, and complex build scripts. Phase 5 workload reduced by ~40%.
New Focus: Model Management (robust downloading) and Performance Optimization (portable app stability).

Strategic Decisions

- [ ] **Option A (Incremental Integration)** is the correct approach
- [ ] Coexistence of Translation Mode and Meeting Mode is acceptable
- [ ] 8-week timeline (including PoC) is feasible
- [ ] **Portable executable distribution** (no installer) is acceptable

Technical Decisions

- [ ] PyQt-Fluent-Widgets dependency is approved (contingent on PoC)
- [ ] Turn-based speaker diarization is sufficient for V1
- [ ] User-configurable speaker count (2-8) meets requirements
- [ ] loguru logging library is approved
- [ ] **Model download on first run** (not bundled) is acceptable

Revised Critical Risks Acknowledged

- [ ] **Risk 1:** PyQt-Fluent-Widgets compatibility (🔴 High - PoC 1 critical)
- [ ] **Risk 2:** Model Management without installer (🟡 Medium - robust download required)
- [ ] **Risk 3:** Portable app permissions (🟡 Medium - user guidance needed)
- [ ] **Risk 4:** Diarization latency (🟡 Medium - must not block ASR)
- [ ] **Risk 5:** Logging permissions (🟢 Low - user-writable paths)

PoC Scope (Week 0) - REVISED

Location: All PoC work isolated in `poc/` folder (do not modify `src/`)

- [ ] **PoC 1:** PyQt-Fluent-Widgets compatibility (2-3 days) - **CRITICAL**
  - Location: `poc/poc1_fluent_widgets/`
  - Output: `results.md` with go/no-go decision
- [ ] **PoC 2:** Speaker diarization integration (2-3 days)
  - Location: `poc/poc2_speaker_diarization/`
  - Output: Latency benchmarks in `results.md`
- [ ] **PoC 3:** Data model coexistence (1-2 days)
  - Location: `poc/poc3_data_model/`
  - Output: Integration validation in `results.md`
- [ ] **PoC 4:** Model download & management (2 days) - **NEW**
  - Location: `poc/poc4_model_download/`
  - Output: Download robustness test results
  - Must test: Resume, retry, checksum, permissions
- [ ] **PoC Isolation Verified:** No modifications to `src/` during PoC phase

Design & UX Features Approved

- [ ] **First Run Wizard** with model download progress bars and retry logic
- [ ] **Audio Test** with "Save Test Audio" button for debugging
- [ ] **Offline Mode** - app launches even if model download fails
- [ ] **CPU/RAM indicator** in status bar
- [ ] **Debug Logging** with privacy mode and auto-directory creation

Fallback Acceptance

- [ ] If PyQt-Fluent-Widgets PoC fails → Use QSS theme immediately
- [ ] If speaker diarization PoC fails → Delay to V2 (minutes without speaker ID)
- [ ] If model download fails → Allow manual download or offline mode
- [ ] Partial PoC success may require scope reduction

Pre-Implementation Requirements

- [ ] ModelManager class design reviewed
- [ ] First Run Wizard flow approved
- [ ] Audio Test "Save Sample" feature approved
- [ ] Portable path handling strategy confirmed (Path.home() usage)

Immediate Next Steps (Upon Approval)

1. **Create `poc/` folder** and set up isolated environment
2. **Start PoC 1 immediately** - PyQt-Fluent-Widgets is the biggest technical unknown
   - Work in `poc/poc1_fluent_widgets/` only
   - Copy needed code from `src/`, do not modify originals
3. **Draft ModelManager** in `poc/poc4_model_download/`
4. Freeze UI design - Meeting Mode layout approved for prototyping
5. Allocate resources for PoC phase (1 week)
6. Schedule PoC review meeting (review all `results.md` files)

Status: 🔄 **APPROVED FOR PoC** → Implementation pending PoC success
Last Updated: 2026-02-21
Version: 2.1-review-ready

End of Roadmap Document