

# Octavia - Stage 0 Task Description

## Project Overview

Octavia is an AI-powered video translation platform designed to deliver high-quality, contextually accurate translations with perfect audio-visual synchronization. We have successfully completed the front-end development and researched the best translation tools. The next phase is to make the front-end fully functional by integrating the backend translation pipeline.

---

## Mission Statement

**Build Octavia like a pyramid: layer on top of layer, step by step, brick by brick.**

---

## Technology Stack

### Core Translation Engine

- **Ollama:** For running local LLMs (Llama 3.1, Mistral, etc.) for translation and reasoning
- **Kimi-K2 Thinking:** For contextually-aware AI translation with reasoning capabilities
- **Coqui TTS:** For high-quality text-to-speech voice synthesis and voice cloning
- **OpenAI WhisperX:** For accurate speech recognition with word-level timestamps
- **FFmpeg:** For video/audio processing, encoding, and synchronization

### Frontend (Completed)

- Next.js 16 with React 19
- TypeScript
- Tailwind CSS with “Liquid Glass” design system
- Framer Motion for animations
- 24 complete dashboard pages

### Backend & Storage

- **Supabase:** For user authentication, database, and file storage
  - **Supabase Storage:** For video/audio file uploads and processed outputs
  - **Supabase Auth:** For user management and OAuth integration
  - **PostgreSQL:** Via Supabase for relational data (users, jobs, subscriptions)
- 

## Target Videos for Testing

The following test videos are provided for the translation pipeline development and quality assurance:

**Google Drive Folder:** [Target Videos](#)

## Video 1: 4-Minute AI Engineering Video

- **Purpose:** Initial testing and quality validation
- **Duration:** ~4 minutes
- **Use Case:** Test all core translation features
- **Success Criteria:** Perfect sync, accurate translation, natural voice

## Video 2: 30-Minute Video

- **Purpose:** Long-form content testing and scalability validation
- **Duration:** ~30 minutes
- **Use Case:** Test sentence-based chunking, memory management, and consistency
- **Success Criteria:** No sync drift, consistent quality throughout, efficient processing

**Important Notes:** - These videos are the primary benchmarks for quality assurance - All development should target achieving perfect results on these videos - Success = both videos translated flawlessly with all QA criteria met

---

## Phase 1: Backend Integration & Translation Pipeline

### Priority 1: Video Translation Core (4-Minute Test)

**Objective:** Successfully translate a 4-minute AI Engineering video with perfect quality.

### Quality Assurance Requirements

1. **Translation Quality**
  - Contextually aware translations (not word-by-word)
  - Preserve technical terminology accurately
  - Maintain speaker's intent and tone
  - Handle idioms and cultural references appropriately
2. **Audio Synchronization**
  - Voice must be perfectly in sync with visual lip movements
  - Audio speed must match original pacing (not too fast/slow)
  - Prevent audio-video desynchronization issues
  - Handle variable-length translations gracefully
3. **Voice Quality**
  - Natural-sounding voice synthesis
  - Match original speaker's tone and emotion
  - Consistent voice characteristics throughout
  - Proper pronunciation and intonation
4. **Technical Requirements**
  - Maintain original video quality
  - Preserve background music and sound effects
  - Handle multi-speaker scenarios
  - Support 1080p video output

## Implementation Steps Step 1: Audio Extraction & Analysis

Input: 4-minute video

↓

FFmpeg audio extraction

↓

WhisperX speech recognition (word-level timestamps)

↓

Output: Timestamped transcript with speaker diarization

## Step 2: Translation with Context

Timestamped transcript

↓

Kimi-K2 contextual translation

↓

Length-constrained translation (preserve timing)

↓

Output: Translated script with timing metadata

## Step 3: Voice Synthesis

Translated script + timing metadata

↓

Coqui TTS voice cloning/synthesis

↓

Speed adjustment to match original timing

↓

Output: Translated audio track

## Step 4: Video Assembly

Original video + Translated audio

↓

FFmpeg audio replacement

↓

Sync verification

↓

Output: Final dubbed video

## Success Criteria for 4-Minute Test

- ☐ Translation is contextually accurate (manual review)
  - ☐ Audio is in sync with video ( $\pm 100\text{ms}$  tolerance)
  - ☐ Voice speed matches original (within 10%)
  - ☐ No audio cutoffs or overlaps
  - ☐ Video quality matches original
  - ☐ Processing time < 3 minutes
  - ☐ Zero critical errors during pipeline
-

## Priority 2: Long-Form Content (30-Minute Video)

**Objective:** Scale the solution to handle 30-minute videos without quality degradation.

### Additional Challenges

1. **Memory Management**
  - Process large files efficiently
  - Implement chunking for long transcripts
  - Manage temporary files properly
2. **Consistency Across Duration**
  - Maintain voice quality throughout
  - Prevent drift in synchronization
  - Handle context across long dialogues
3. **Performance Optimization**
  - Parallel processing where possible
  - Efficient caching strategies
  - Progress tracking for user feedback

**Implementation Approach** **Chunking Strategy** - **Primary Method:** Split audio by sentence boundaries using WhisperX word-level timestamps - **Fallback Method:** If sentences are too long, split into 6-9 second segments at natural pauses - **Key Principles:** - Never split mid-word or mid-phrase - Preserve semantic coherence within each chunk - Maintain context by including previous sentence in translation prompt - Process chunks sequentially to maintain voice consistency - **Chunk Overlap:** Include 2-3 seconds of overlap between chunks for smooth transitions - **Benefits:** - Better sync accuracy (shorter segments easier to match) - More precise timing control - Reduced risk of desynchronization - Natural speech rhythm preservation

**Quality Checks** - Automated sync verification every 5 minutes - Voice consistency analysis - Translation quality spot-checks - Final end-to-end review

### Success Criteria for 30-Minute Test

- ☐ All 4-minute test criteria met
- ☐ No sync drift over duration
- ☐ Consistent voice quality throughout
- ☐ Successful handling of scene transitions
- ☐ Memory usage < 8GB peak
- ☐ Sentence-based chunking works reliably
- ☐ 6-9 second fallback chunks maintain sync
- ☐ No audio gaps or overlaps between chunks

---

## Phase 2: Frontend-Backend Integration

### API Development

#### Required Endpoints

1. **POST** /api/translate/video

- Upload video file
- Select source/target languages
- Configure voice settings
- Return job ID
- 2. **GET /api/translate/status/:jobId**
  - Real-time progress updates
  - Current pipeline stage
  - Estimated completion time
  - Error handling
- 3. **GET /api/translate/download/:jobId**
  - Download translated video
  - Provide multiple format options
  - Include metadata/statistics
- 4. **POST /api/translate/audio**
  - Audio-only translation endpoint
  - Voice cloning options
- 5. **POST /api/subtitles/generate**
  - Auto-generate subtitles
  - Multiple format exports (SRT, VTT, ASS)
- 6. **POST /api/subtitles/translate**
  - Translate subtitle files
  - Context-aware translation

## Frontend Updates

### Connect Existing Pages to Backend

1. **Video Translation Pages**
  - /dashboard/video → File upload with API integration
  - /dashboard/video/progress → Real-time status updates via WebSocket/polling
  - /dashboard/video/review → Load translated video from API
2. **Audio Translation Pages**
  - /dashboard/audio → Audio upload and processing
  - /dashboard/audio/subtitle-to-audio → Convert subtitles to speech
3. **Subtitle Pages**
  - /dashboard/subtitles → Generate subtitles API
  - /dashboard/subtitles/translate → Translate subtitles API

## State Management

Implement proper state management for: - File upload progress - Job status tracking - Error handling and retry logic - User notifications

---

## Phase 3: Authentication & Account Management

### Authentication System

#### Required Features

1. **Sign In / Sign Up**
  - Email/password authentication
  - Form validation
  - Password strength requirements
  - Remember me functionality
2. **Google OAuth**
  - Google Sign-In integration
  - Profile data sync
  - Session management
3. **Email Verification**
  - Send verification email on signup
  - Verification link expiration (24 hours)
  - Resend verification email option
  - Account activation flow
4. **Password Management**
  - Change password (authenticated users)
  - Forgot password flow
  - Reset password via email link
  - Password reset link expiration (1 hour)

## Account Pages

**Profile & Security** (/dashboard/profile) - [x] UI completed - [ ] Connect to user API - [ ] Implement profile updates - [ ] Implement password change - [ ] Add 2FA setup (optional)

**Team Management** (/dashboard/team) - [x] UI completed - [ ] Invite team members - [ ] Role management (Admin, Member, Viewer) - [ ] Remove team members - [ ] Team usage statistics

## Phase 4: Billing Integration with Polar.sh

### Setup

1. **Create Polar.sh Test Account**
  - Sign up at <https://polar.sh>
  - Configure test products
  - Get API keys
2. **Define Subscription Tiers**

**Free Tier** - 5 videos/month (max 5 minutes each) - Standard quality - 1 voice clone - Community support

**Pro Tier** (\$29/month) - 50 videos/month (max 30 minutes each) - HD quality - 10 voice clones - Priority support - Advanced features (Magic Mode)

**Enterprise Tier** (\$99/month) - Unlimited videos - 4K quality - Unlimited voice clones - 24/7 support - API access - Custom integrations

## Implementation

**Billing Page** (/dashboard/billing) - ☒ UI completed - ☐ Display current subscription - ☐ Show usage statistics - ☐ Upgrade/downgrade flows - ☐ Polar.sh checkout integration - ☐ Invoice history from Polar API - ☐ Payment method management

**Usage Tracking** - Track video minutes processed - Monitor voice clones created - API call counting  
- Real-time usage display

**Webhook Integration** - Handle subscription created - Handle subscription updated - Handle subscription canceled - Handle payment succeeded - Handle payment failed

---

## Phase 5: Additional Pages & Features

### Job History (/dashboard/history)

- ☒ UI completed
- ☐ Fetch jobs from API
- ☐ Filter by status, date, type
- ☐ Download completed jobs
- ☐ Re-process failed jobs
- ☐ Delete old jobs

### Projects (/dashboard/projects)

- ☒ UI completed
- ☐ Create/edit/delete projects
- ☐ Organize jobs into projects
- ☐ Share projects with team

### My Voices (/dashboard/voices)

- ☒ UI completed
- ☐ Upload voice samples
- ☐ Train voice clones
- ☐ Test voice output
- ☐ Delete voice clones

## Help & Support

- ☒ UI completed
  - ☐ Search functionality
  - ☐ Link to documentation
  - ☐ Video tutorials
  - ☐ Contact form integration
-

## Quality Assurance Framework

### Automated Testing

**Unit Tests** - API endpoint tests - Translation accuracy tests - Audio sync verification - Voice quality metrics

**Integration Tests** - End-to-end translation pipeline - File upload/download flows - Authentication flows - Payment processing

**Performance Tests** - Load testing with concurrent jobs - Memory usage monitoring - Processing time benchmarks - API response times

### Manual QA Checklist

**For Each Video Translation** - ☐ Audio is in perfect sync (visual inspection) - ☐ Translation is contextually correct (native speaker review) - ☐ Voice quality is natural and consistent - ☐ No audio artifacts or glitches - ☐ Video quality matches original - ☐ Subtitles are accurately synced (if generated) - ☐ File downloads correctly - ☐ Metadata is preserved

**For Long Videos (>10 minutes)** - ☐ No sync drift detected - ☐ Consistent voice throughout - ☐ Proper handling of scene changes - ☐ Background audio preserved correctly

### Error Handling

#### Common Issues & Solutions

1. **Audio-Video Desync**
    - Cause: Translation length variation
    - Solution: Dynamic time-stretching, smart chunking
  2. **Fast/Slow Voice Speed**
    - Cause: Incorrect TTS pacing
    - Solution: Speed normalization, reference audio matching
  3. **Poor Translation Quality**
    - Cause: Lack of context
    - Solution: Larger context windows, reference glossaries
  4. **Long Processing Times**
    - Cause: Inefficient pipeline
    - Solution: Parallel processing, GPU acceleration
- 

## Development Approach: Building the Pyramid

### Layer 1: Foundation (Week 1-2)

- Set up backend infrastructure
- Implement basic API endpoints
- Test WhisperX integration
- Test Coqui TTS integration
- Verify FFmpeg processing



## **Layer 2: Core Translation (Week 3-4)**

- Complete translation pipeline
- Implement 4-minute video test
- Quality assurance on short video
- Iterate based on QA results
- Achieve success criteria

## **Layer 3: Scalability (Week 5-6)**

- Implement chunking strategy
- Test 30-minute video
- Performance optimization
- Memory management
- Load testing

## **Layer 4: Integration (Week 7-8)**

- Connect frontend to backend
- Implement real-time progress
- File upload/download flows
- Error handling & retries
- User notifications

## **Layer 5: Authentication (Week 9-10)**

- Implement auth system
- Google OAuth integration
- Email verification
- Password management
- Session handling

## **Layer 6: Monetization (Week 11-12)**

- Polar.sh integration
- Subscription management
- Usage tracking
- Webhook handling
- Payment flows

## **Layer 7: Polish & Launch (Week 13-14)**

- Complete remaining pages
  - Final QA testing
  - Performance optimization
  - Documentation
  - Production deployment
-

## Success Metrics

### Technical Metrics

- Translation accuracy: >95%
- Audio sync tolerance:  $\pm 100\text{ms}$
- Processing speed: <3x video length
- Uptime: >99.5%
- Error rate: <1%

### User Experience Metrics

- Translation quality score: >4.5/5
- Voice naturalness score: >4.5/5
- Sync quality score: >4.5/5
- Overall satisfaction: >4.5/5

### Business Metrics

- Successful payment processing: >99%
  - Subscription retention: >80%
  - Average videos per user: >10/month
  - Support ticket volume: <5% of users
- 

## Risks & Mitigation

### Technical Risks

1. **Risk:** Audio-video sync issues
  - **Mitigation:** Implement robust sync verification, dynamic time-stretching
2. **Risk:** Poor translation quality
  - **Mitigation:** Use Kimi-K2 with proper context, implement review workflow
3. **Risk:** Slow processing times
  - **Mitigation:** GPU acceleration, parallel processing, caching
4. **Risk:** High infrastructure costs
  - **Mitigation:** Optimize resource usage, implement job queuing

### Business Risks

1. **Risk:** Low user adoption
    - **Mitigation:** Focus on quality, competitive pricing, marketing
  2. **Risk:** Competition from established players
    - **Mitigation:** Differentiate with superior quality, better UX
  3. **Risk:** Scaling challenges
    - **Mitigation:** Design for scale from day one, load testing
-

## Conclusion

This is an ambitious but achievable project. By following the pyramid approach and maintaining a strong focus on quality at every layer, we will build a world-class video translation platform.

**Remember:** Quality over speed. Each layer must be solid before moving to the next.

**Next Steps:** 1. Set up development environment with all required tools 2. Begin Layer 1 (Foundation) implementation 3. Complete 4-minute video test with QA 4. Iterate and improve based on results

---

**Document Version:** 1.0

**Last Updated:** 2025-11-23

**Status:** Ready to Begin