Below is a **detailed task implementation guide** for the **AI/ML Developer** working on the **VerseProjection SaaS web app** MVP, focused on real-time Bible verse auto-detection and projection using public-domain KJV/WEB texts. This guide aligns with the updated PRD (artifact_id: a7c7a8e1-9f68-4929-bc62-8d5d19d66186, artifact_version_id: ca07a5bd-a168-4b01-9a91-8cc217441994), emphasizing continuous speech analysis, top 10 verse matching, user selection, and model learning via user feedback. The AI/ML Developer's role is to build and optimize the AI pipeline (OpenAI Whisper for speech-to-text, BERT for semantic verse matching) to achieve <1-second processing, 90%+ detection accuracy, and 95%+ inclusion of the correct verse in the top 10 matches, while supporting incremental learning and offline functionality via an Electron app. The guide provides comprehensive instructions for model development, fine-tuning, optimization, deployment, and testing, ensuring scalability for 1,000–10,000 users within a 6–9-month timeline and $150K–$400K budget.

### Task Implementation Guide for AI/ML Developer: VerseProjection MVP

#### 1. Overview
Role: AI/ML Developer
Objective: Develop, fine-tune, optimize, and deploy the AI/ML pipeline for real-time Bible verse detection, processing continuous preacher speech to transcribe (Whisper) and match (BERT) against KJV/WEB verses, outputting the top 10 matches for user selection. Implement a feedback loop to improve accuracy (5% increase in 3 months) and support offline inference in the Electron app. Ensure <1-second processing (<500ms transcription, <500ms matching), 90%+ accuracy, and 95%+ correct verse inclusion in top 10 matches.
Scope:
- **AI Pipeline**: OpenAI Whisper for speech-to-text, BERT for semantic verse matching, regex for explicit references, ONNX for optimization.
- **Feedback Loop**: Store user selections, retrain BERT monthly using feedback data.
- **Offline Mode**: Lightweight Whisper/BERT models for Electron app.

- **Deliverables**: Trained models (Whisper, BERT), inference pipeline, feedback retraining script, offline models, performance benchmarks, documentation.
Timeline: Months 3–8 (development and testing, per PRD), integrated with backend (Node.js/Express) and cloud (AWS SageMaker).
Budget Allocation: ~$50K–$100K (part of salaries, included in $150K–$400K).
Tools: Python, TensorFlow/PyTorch, Hugging Face Transformers, ONNX Runtime, Librosa, AWS SageMaker, PostgreSQL (pgvector), GitHub.

#### 2. AI/ML Requirements
- **Performance**:
  - Transcription (Whisper): <500ms per 5-second audio chunk.
  - Matching (BERT): <500ms for paraphrases, <100ms for explicit references.
  - Total: <1-second processing.
  - Accuracy: 90%+ for transcription, 95%+ correct verse in top 10 matches.
- **Scalability**: Support 1,000–10,000 concurrent users (AWS SageMaker auto-scaling).
- **Learning**: 5% accuracy improvement after 3 months via user feedback.
- **Offline**: CPU-based inference (8GB RAM, Core i5) with <1-second processing.
- **Compatibility**: Integrate with Node.js backend, PostgreSQL (pgvector), React frontend via WebSocket.
- **Cost**: ~$250–$450/month for SageMaker (inference: $200–$400, retraining: $50).

#### 3. AI Pipeline Architecture
- **Input**: Continuous audio stream (16-bit, 44.1 kHz, Opus codec, 5-second chunks every 2 seconds).
- **Processing**:
  - **Whisper**: Transcribes audio to text (~500ms, 90%+ accuracy).
  - **BERT**: Matches transcription to KJV/WEB verses (~500ms for paraphrases, <100ms for explicit references), ranks top 10 by cosine similarity.
  - **Regex**: Extracts explicit references (e.g., “John 3:16”) for faster matching.
  - **ONNX**: Optimizes inference for low latency.
- **Output**: JSON array of top 10 matches: [{`verse_id`, `reference`, `text`, `version`, `confidence`}, …].

- **Feedback**: Store user selections in PostgreSQL (`Feedback` table), retrain BERT monthly.
- **Offline**: Lightweight Whisper/BERT models in Electron app, using SQLite for verse storage.

#### 4. Task Breakdown
The AI/ML Developer's tasks are organized into five phases: Data Preparation, Model Development, Pipeline Integration, Offline Mode, and Testing/Optimization.

##### 4.1 Data Preparation
Objective: Collect and preprocess datasets for Whisper and BERT fine-tuning, ensuring sermon-specific accuracy and legal compliance.
Tasks:
1. **Sermon Audio Dataset (Whisper)**:
   - **Source**: ~10 hours of public-domain or synthetic sermon audio.
     - Public-domain: Sermons from Project Gutenberg, LibriVox, or open church archives (e.g., SermonAudio, license: CC0).
     - Synthetic: Generate clips using text-to-speech (e.g., ElevenLabs, open-source voices) with KJV/WEB verses and noise (clapping, music).
   - **Requirements**:
     - Format: 16-bit, 44.1 kHz WAV.
     - Content: Pastors citing verses (e.g., "John 3:16"), paraphrasing, preaching with noise (50–70 dB SNR).
     - Diversity: US/UK accents, male/female voices, varied noise levels.
   - **Preprocessing**:
     - Use Librosa to normalize audio (0 dBFS), apply noise filtering (high-pass: 100 Hz, low-pass: 8 kHz).
     - Segment into 5-second chunks, align with transcriptions (manual or auto-generated).
     - Output: ~10,000 chunks with transcriptions (JSON: [`audio_path`, `text`]).
   - **Storage**: AWS S3 (~1GB), accessible to SageMaker.
2. **Verse Matching Dataset (BERT)**:
   - **Source**: Public-domain KJV/WEB texts (~4MB) from CrossWire Bible Society.
     - ~31,000 verses (KJV: ~23,000, WEB: ~8,000).
   - **Requirements**:

- Pairs: Transcription → Verse (e.g., "God so loved the world" → John 3:16).
- Size: ~10,000 pairs (50% explicit references, 50% paraphrases).

- **Generation**:
  - Explicit: Extract verse citations from sermon transcripts (e.g., "John 3:16").
  - Paraphrases:
  - Manual: Curate ~2,000 pairs from sermon texts (e.g., "God loved the world and gave His Son" → John 3:16).
  - Synthetic: Use paraphrasing tools (e.g., T5 model, Hugging Face) to generate ~8,000 pairs from verse texts.
  - Validation: Manually verify 10% of synthetic pairs for accuracy.
- **Preprocessing**:
  - Tokenize texts (BERT tokenizer, max length: 128 tokens).
  - Generate embeddings (BERT-base-uncased) for verses, store in PostgreSQL (`Bible.embedding`, pgvector).
  - Output: JSONL file ({`transcription`, `verse_id`, `reference`, `text`}).
- **Storage**: AWS S3 (~10MB), PostgreSQL for embeddings (~8MB).

3. **Noise Augmentation**:
   - Add church-specific noise (clapping, music, congregation, 50–70 dB SNR) to audio using Librosa.
   - Sources: Free sound libraries (e.g., Freesound, CC0 license).
   - Output: Augmented audio dataset (~10 hours, ~1GB).
4. **Legal Compliance**:
   - Verify all data is public-domain or CC0 (e.g., CrossWire KJV/WEB, LibriVox sermons).
   - Document sources in GitHub (`README.md`, e.g., "KJV sourced from CrossWire, CC0").

Deliverables:
- Sermon audio dataset (~10 hours, S3).
- Verse matching dataset (~10,000 pairs, S3).
- Precomputed verse embeddings (PostgreSQL, ~8MB).
- Documentation (data sources, preprocessing scripts, GitHub).

Timeline: Month 3 (4 weeks).
Effort: ~80–120 hours.

##### 4.2 Model Development
Objective: Fine-tune and optimize Whisper and BERT for sermon-specific transcription and verse matching, achieving <1-second processing and 90%+ accuracy.
Tasks:
1. **Whisper Fine-Tuning (Speech-to-Text)**:
   - **Model**: Whisper-small (240M parameters, ~500MB, Hugging Face: `openai/whisper-small`).
   - **Input**: 5-second audio chunks (16-bit, 44.1 kHz).
   - **Output**: Transcription text (e.g., "God so loved the world").
   - **Fine-Tuning**:
        - Dataset: ~10 hours sermon audio (~10,000 chunks, JSON: {`audio_path`, `text`}).
        - Framework: PyTorch, Hugging Face Transformers.
        - Loss: CTC (Connectionist Temporal Classification).
        - Parameters:
        - Batch size: 16 (SageMaker ml.g4dn.xlarge, 16GB GPU).
        - Learning rate: 1e-5.
        - Epochs: 3–5 (monitor validation loss).
        - Augmentation: Add noise (clapping, music, 50–70 dB SNR) during training.
        - Validation: Split dataset (80% train, 20% validation), target WER (Word Error Rate) <10%.
   - **Optimization**:
        - Convert to ONNX (ONNX Runtime) for 30% faster CPU inference (<500ms).
        - Quantize to INT8 for offline mode (reduces model size to ~300MB).
   - **Metrics**:
        - WER: <10% on validation set.
        - Latency: <500ms per 5-second chunk (ml.t3.medium, CPU).
        - Accuracy: 90%+ for sermon speech (verse references, paraphrases).
2. **BERT Fine-Tuning (Verse Matching)**:
   - **Model**: BERT-base-uncased (110M parameters, ~400MB, Hugging Face: `bert-base-uncased`).
   - **Input**: Transcription text (max 128 tokens).

- **Output**: Top 10 verse matches ([{`verse_id`, `reference`, `text`, `version`, `confidence`}, …]).
- **Preprocessing**:
    - Precompute embeddings for all KJV/WEB verses (~31,000) using BERT.
    - Store in PostgreSQL (`Bible.embedding`, pgvector, 768-dimensional vectors, ~8MB).
    - Cache embeddings in Redis (~8MB) for sub-100ms queries.
- **Explicit Reference Detection**:
    - Implement regex: `[A-Za-z]+ \d+:\d+` (e.g., "John 3:16").
    - Query PostgreSQL (`SELECT * FROM Bible WHERE book='John' AND chapter=3 AND verse=16`).
    - Latency: <100ms, accuracy: 99%+.
- **Paraphrase Matching**:
    - Generate transcription embedding (BERT, 768-dimensional).
    - Compute cosine similarity against verse embeddings (pgvector, `SELECT id, reference, text, version, 1 - cosine_distance(embedding, $1) AS confidence ORDER BY confidence DESC LIMIT 10`).
    - Confidence threshold: 0.7 (adjustable, default).
    - Latency: ~500ms, accuracy: 95%+ correct verse in top 10.
- **Fine-Tuning**:
    - Dataset: ~10,000 transcription-verse pairs (50% explicit, 50% paraphrases).
    - Task: Sentence similarity (transcription → verse).
    - Loss: Triplet loss (anchor: transcription, positive: correct verse, negative: incorrect verse).
    - Parameters:
    - Batch size: 32 (ml.g4dn.xlarge).
    - Learning rate: 2e-5.
    - Epochs: 3–5 (monitor validation accuracy).
    - Validation: Split dataset (80% train, 20% validation), target top-10 accuracy: 95%+.
- **Optimization**:
    - Convert to ONNX for 30% faster inference (<500ms).
    - Quantize to INT8 for offline mode (~250MB).
    - Cache frequent queries (Redis, LRU policy, 8MB).
- **Metrics**:

        - Top-10 Accuracy: 95%+ (correct verse in top 10).
        - Latency: <500ms for paraphrases, <100ms for explicit references.
        - Confidence: Mean cosine similarity >0.7 for correct matches.
3. **Regex for Explicit References**:
   - **Logic**:
        - Pattern: `[A-Za-z]+ \d+:\d+` (e.g., “John 3:16”, “Romans 8:28”).
        - Normalize input (e.g., “First Corinthians” → “1 Corinthians”).
        - Query PostgreSQL directly for exact matches.
   - **Implementation**: Python (re module), integrated into BERT pipeline.
        - If regex matches, bypass BERT, return single match (confidence: 1.0).
        - Else, proceed to BERT for paraphrase matching.
   - **Metrics**:
        - Accuracy: 99%+ for valid references.
        - Latency: <100ms (PostgreSQL query).

Deliverables:
- Fine-tuned Whisper model (ONNX, ~300MB).
- Fine-tuned BERT model (ONNX, ~250MB).
- Precomputed verse embeddings (PostgreSQL, Redis).
- Regex script for explicit references (Python).
- Training scripts and logs (GitHub, `models/training/`).

Timeline: Months 3–4 (8 weeks).
Effort: ~160–240 hours.

##### 4.3 Pipeline Integration
Objective: Integrate Whisper and BERT into a real-time inference pipeline, connecting with backend (Node.js/Express), database (PostgreSQL), and frontend (React) via WebSocket.
Tasks:
1. **Inference Pipeline**:
   - **Framework**: Python (FastAPI for inference API, ONNX Runtime for models).
   - **Input**: Audio chunk (5-second, Opus codec, via WebSocket).
   - **Processing**:
        - Decode Opus to WAV (Librosa, <50ms).
        - Transcribe with Whisper (ONNX, <500ms).

    - Check for explicit references (regex, <100ms).
    - If explicit: Query PostgreSQL, return single match.
    - Else: Generate transcription embedding (BERT, ONNX), query PostgreSQL (pgvector) for top 10 matches (~500ms).
  - **Output**: JSON via WebSocket:
    ```json
    {
    "transcription": "God so loved the world",
    "matches": [
    {"verse_id": "uuid", "reference": "John 3:16", "text": "For God so loved...", "version": "KJV", "confidence": 0.92},
    ...
    ]
    }
    ```
  - **Latency**: <1 second (transcription: ~500ms, matching: ~100–500ms).
  - **Hosting**: AWS SageMaker (ml.t3.medium, 2 vCPUs, 4GB RAM, auto-scaling).
2. **Backend Integration**:
  - **API Endpoint**: `/infer` (POST, FastAPI).
    - Input: Base64-encoded Opus audio (5-second chunk).
    - Output: JSON (transcription, top 10 matches).
  - **WebSocket**: Use Socket.IO (Node.js) to stream audio and receive matches.
    - Client (React): Sends audio chunks every 2 seconds.
    - Server (Node.js): Forwards to FastAPI, relays JSON to frontend.
  - **Error Handling**:
    - Low audio quality (SNR <40 dB): Return error (`{"error": "Low audio quality"}`).
    - No matches (confidence <0.7): Return empty matches (`{"matches": []}`).
    - Log errors to AWS CloudWatch (e.g., `{"event": "transcription_failure", "details": {...}}`).
3. **Database Integration**:
  - **Verse Queries**:
    - Explicit: `SELECT * FROM Bible WHERE book=$1 AND chapter=$2 AND verse=$3`.

    - Paraphrase: `SELECT id, reference, text, version, 1 - cosine_distance(embedding, $1) AS confidence ORDER BY confidence DESC LIMIT 10`.
   - **Feedback Storage**:
    - Insert user selections: `INSERT INTO Feedback (user_id, timestamp, transcription, selected_verse_id, top_matches) VALUES ($1, $2, $3, $4, $5)`.
    - Size: ~10MB/month for 1,000 users (100 selections/user/month).
   - **Embedding Cache**:
    - Store verse embeddings in Redis (8MB, LRU, TTL: 1 week).
    - Fallback to PostgreSQL if cache miss.
4. **Frontend Integration**:
   - **Matches Display**: Send JSON to React frontend via WebSocket, render in Matches Table.
   - **Selection Feedback**: On user click, send selected `verse_id` to backend (`POST /feedback`), store in PostgreSQL.
   - **Real-Time**: Ensure transcription/matches update every 2 seconds (<200ms frontend render).

Deliverables:
- Inference pipeline (FastAPI, Python, `pipeline/infer.py`).
- API documentation (Swagger, `docs/api.md`).
- WebSocket integration (Node.js, Socket.IO).
- Database queries (PostgreSQL, `db/queries.sql`).
- Redis caching script (Python, `cache/redis.py`).

Timeline: Months 5–6 (8 weeks).
Effort: ~160–240 hours.

##### 4.4 Feedback Loop and Retraining
Objective: Implement a feedback loop to store user selections and retrain BERT monthly, improving accuracy by 5% in 3 months.
Tasks:
1. **Feedback Collection**:
   - **Schema**: `Feedback` table (PostgreSQL).
    - Columns: `id` (UUID), `user_id` (UUID), `timestamp` (datetime), `transcription` (text), `selected_verse_id` (UUID), `top_matches` (JSON).
   - **API Endpoint**: `/feedback` (POST, FastAPI).

- Input: `{transcription, selected_verse_id, top_matches}`.
- Action: Insert into `Feedback` table.

- **Validation**:
  - Ensure `selected_verse_id` exists in `Bible` table.
  - Filter low-confidence selections (confidence <0.5) to avoid noise.
- **Storage**: ~10MB/month for 1,000 users (100 selections/user/month).

2. **Retraining Pipeline**:
   - **Dataset**: Extract feedback data (`SELECT transcription, selected_verse_id FROM Feedback WHERE timestamp > $1`).
     - Initial: ~1,000–10,000 pairs/month for 1,000 users.
     - Combine with original dataset (~10,000 pairs) for stability.
   - **Preprocessing**:
     - Tokenize transcriptions (BERT tokenizer, max 128 tokens).
     - Generate triplets: (transcription, correct verse, incorrect verse).
     - Incorrect verse: Random verse or low-ranked match from `top_matches`.
   - **Training**:
     - Model: BERT-base-uncased (same as initial).
     - Task: Sentence similarity (triplet loss).
     - Parameters:
     - Batch size: 32 (ml.m5.large, 8GB RAM).
     - Learning rate: 1e-6 (fine-tuning).
     - Epochs: 1–2 (to avoid overfitting).
     - Validation: 20% of feedback data, target top-10 accuracy: +5% after 3 months.
   - **Optimization**: Convert to ONNX/INT8 post-retraining.
   - **Hosting**: AWS SageMaker (ml.m5.large, ~1 hour/month, ~$50).
   - **Schedule**: Monthly retraining (first run: Month 7, after pilot).
3. **Deployment**:
   - Update inference pipeline with new BERT model (SageMaker endpoint).
   - Test new model on validation set before deployment (ensure latency <500ms).
   - Rollback plan: Revert to previous model if accuracy degrades.
4. **Monitoring**:
   - Log retraining metrics to CloudWatch (e.g., `{"event": "retraining", "top_10_accuracy": 0.96, "latency": 450ms}`).
   - Track user selection rates (e.g., % of top-1 vs. lower-ranked selections).

   - Alert if accuracy drops post-retraining (CloudWatch alarm).

Deliverables:
- Feedback API endpoint (FastAPI, `api/feedback.py`).
- Retraining script (Python, `models/retrain.py`).
- Updated BERT model (ONNX, monthly releases).
- Monitoring dashboard (CloudWatch, `monitoring/retraining.json`).

Timeline: Months 6–7 (6 weeks).
Effort: ~120–180 hours.

##### 4.5 Offline Mode
Objective: Develop lightweight Whisper/BERT models for offline inference in the Electron app, running on 8GB RAM, Core i5 laptops.
Tasks:
1. **Model Optimization**:
   - **Whisper**:
      - Use Whisper-tiny (39M parameters, ~100MB after INT8 quantization).
      - Convert to ONNX/INT8 (ONNX Runtime, ~30% faster).
      - Latency: <500ms on Core i5 (2.5 GHz, 4 cores).
   - **BERT**:
      - Use DistilBERT-base-uncased (66M parameters, ~150MB after INT8).
      - Convert to ONNX/INT8.
      - Latency: <500ms for paraphrases, <100ms for explicit references.
   - **Storage**: ~250MB total (Whisper: 100MB, BERT: 150MB).
2. **Inference Pipeline**:
   - **Framework**: Python (ONNX Runtime, integrated with Electron via Node.js bindings).
   - **Input**: Audio chunk (5-second, captured via PyAudio).
   - **Processing**:
      - Transcribe with Whisper-tiny (<500ms).
      - Match with DistilBERT or regex (<500ms/<100ms).
      - Query SQLite (`Bible` table, ~4MB texts, ~8MB embeddings).
   - **Output**: JSON to Electron UI (top 10 matches).
   - **Latency**: <1 second total.
3. **Database**:
   - **Schema**: Mirror PostgreSQL `Bible` table (SQLite).

- Columns: `id`, `version`, `book`, `chapter`, `verse`, `text`, `embedding` (binary blob, 768 floats).
    - Size: ~12MB (4MB texts, 8MB embeddings).
  - **Queries**:
    - Explicit: `SELECT * FROM Bible WHERE book=$1 AND chapter=$2 AND verse=$3`.
    - Paraphrase: Custom cosine similarity (Python, NumPy, <500ms for ~31,000 verses).
  - **Feedback**: Store selections in SQLite (`Feedback` table), sync to PostgreSQL when online.
4. **Integration**:
  - **Electron**: Use `python-shell` to run Python inference in Electron.
  - **Sync**: On reconnect, upload SQLite `Feedback` to PostgreSQL (`INSERT INTO Feedback ...`).
  - **Storage Check**: Verify ~600MB free disk space on launch (Electron FS module).
5. **Testing**:
  - Test on low-end laptop (8GB RAM, Core i5, Windows 10).
  - Metrics:
    - Latency: <1 second.
    - Accuracy: 85%+ (slightly lower than online due to smaller models).
    - Top-10 Accuracy: 90%+ correct verse inclusion.

Deliverables:
- Offline models (Whisper-tiny, DistilBERT, ONNX/INT8, ~250MB).
- Offline inference pipeline (Python, `offline/infer.py`).
- SQLite database (Bible, Feedback, ~12MB).
- Electron integration script (Node.js, `offline/electron.py`).
- Test report (latency, accuracy, GitHub).

Timeline: Month 7–8 (6 weeks).
Effort: ~120–180 hours.

##### 4.6 Testing and Optimization
Objective: Validate AI pipeline performance, optimize for latency and accuracy, and ensure scalability.
Tasks:

1. **Unit Testing**:
  - **Whisper**: Test transcription on 1,000 audio chunks (WER <10%).
  - **BERT**: Test matching on 1,000 transcription-verse pairs (top-10 accuracy: 95%+).
  - **Regex**: Test explicit references (100 samples, 99%+ accuracy).
  - **Feedback**: Test insertion/retrieval (100 selections, 100% success).
  - **Framework**: Pytest, run in GitHub Actions.
2. **End-to-End Testing**:
  - Simulate church environment:
    - Audio: Sermon clips with noise (clapping, music, 50–70 dB SNR).
    - Input: Stream 5-second chunks every 2 seconds via WebSocket.
    - Output: Verify top 10 matches in React frontend (<1s from audio to display).
  - Metrics:
    - Latency: <1 second (transcription + matching).
    - Accuracy: 90%+ transcription, 95%+ top-10 inclusion.
    - Scalability: 1,000 concurrent streams (SageMaker auto-scaling).
  - Tool: JMeter for load testing, CloudWatch for monitoring.
3. **Optimization**:
  - **Latency**:
    - Profile pipeline (Python `cProfile`, identify bottlenecks).
    - Optimize ONNX models (e.g., fuse layers, reduce precision).
    - Increase batch size for overlapping chunks (e.g., process 2 chunks in parallel).
  - **Memory**:
    - Cap Whisper/BERT memory to 1GB each (ml.t3.medium).
    - Use Redis to cache embeddings (8MB, sub-100ms queries).
  - **Cost**:
    - Use SageMaker Reserved Instances to reduce costs (~$250–$450/month).
    - Limit retraining to 1 hour/month (~$50).
4. **Church Pilot Testing**:
  - Deploy to 5–10 churches (Month 8).
  - Collect feedback data (~1,000 selections/pilot).
  - Metrics:
    - User selection rate: % of top-1 vs. lower-ranked matches.
    - Accuracy: 90%+ (transcription), 95%+ (top-10).

- Latency: <1 second (CloudWatch logs).
   - Iterate: Adjust confidence threshold (e.g., 0.7 to 0.75) if low-ranked selections are frequent.
5. **Documentation**:
   - Model specs (Whisper-small, BERT-base, ONNX/INT8, latency, accuracy).
   - Pipeline setup (SageMaker, FastAPI, Redis).
   - Retraining process (data prep, training, deployment).
   - Offline mode (Electron integration, SQLite).
   - Store in GitHub (\`docs/ai.md\`).

Deliverables:
- Test suite (Pytest, \`tests/\`).
- Performance benchmarks (latency, accuracy, scalability, \`tests/benchmarks.md\`).
- Pilot test report (feedback, metrics, \`tests/pilot.md\`).
- Documentation (GitHub, \`docs/ai.md\`).

Timeline: Months 7–8 (6 weeks).
Effort: ~120–180 hours.

#### 5. Developer Collaboration
Work closely with backend, frontend, and DevOps teams to ensure seamless integration:
- **Backend Developer**:
  - Provide FastAPI endpoint (\`/infer\`, \`/feedback\`) specs (Swagger).
  - Share PostgreSQL queries (explicit, paraphrase, feedback).
  - Collaborate on WebSocket (Socket.IO, audio streaming, match delivery).
- **Frontend Developer**:
  - Define JSON format for matches (e.g., \`{transcription, matches: [...]}\`).
  - Support real-time rendering (<200ms) of transcription/matches.
  - Integrate feedback submission (\`POST /feedback\`).
- **DevOps Engineer**:
  - Deploy models to SageMaker (ml.t3.medium for inference, ml.m5.large for retraining).
  - Set up Redis (cache.t3.micro, 8MB embeddings).
  - Configure CloudWatch for monitoring (latency, accuracy, errors).
  - Optimize costs ($460–$900/month, Reserved Instances).

- **Meetings**:
  - Weekly sync (30min, align on APIs, performance).
  - Sprint reviews (Month 5–8, verify integration).
- **GitHub**:
  - Create issues for bugs (e.g., “Whisper latency >500ms on noisy audio”).
  - Store code/scripts (`models/`, `pipeline/`, `tests/`).

Deliverables: API specs, integration tests, GitHub issues, meeting notes.

#### 6. Risk Mitigation
- **Accuracy**:
  - Risk: Whisper/BERT fail on noisy audio or complex paraphrases.
  - Mitigation: Fine-tune on diverse dataset (noise, accents, paraphrases), provide override search bar.
- **Latency**:
  - Risk: Pipeline exceeds 1-second processing.
  - Mitigation: Optimize with ONNX/INT8, cache embeddings in Redis, profile bottlenecks.
- **Scalability**:
  - Risk: SageMaker fails under 1,000+ concurrent users.
  - Mitigation: Test with JMeter, configure auto-scaling (1–10 ml.t3.medium instances).
- **Learning**:
  - Risk: Feedback data is noisy, degrading accuracy.
  - Mitigation: Filter low-confidence selections (<0.5), validate retraining (rollback if accuracy drops).
- **Offline**:
  - Risk: Lightweight models perform poorly on low-end laptops.
  - Mitigation: Use Whisper-tiny/DistilBERT, test on Core i5, fallback to override search.
- **Legal**:
  - Risk: Non-public-domain data introduces licensing issues.
  - Mitigation: Use only CC0 sources (CrossWire, LibriVox), document compliance.

Deliverables: Risk assessment, mitigation plan.

#### 7. Success Criteria
- **Performance**:
  - Transcription: <500ms, 90%+ accuracy (WER <10%).
  - Matching: <500ms (paraphrases), <100ms (explicit), 95%+ top-10 accuracy.
  - Total: <1-second processing.
- **Scalability**: Supports 1,000–10,000 users (JMeter tests).
- **Learning**: 5% accuracy improvement after 3 months (CloudWatch metrics).
- **Offline**: <1-second latency, 85%+ accuracy on Core i5 (test report).
- **Pilot**: 90%+ of pilot churches (5–10) select correct verse from top 10 (user feedback).
- **Cost**: SageMaker costs within $250–$450/month (CloudWatch billing).

#### 8. Resources
- **PRD Reference**: artifact_id: a7c7a8e1-9f68-4929-bc62-8d5d19d66186, artifact_version_id: ca07a5bd-a168-4b01-9a91-8cc217441994.
- **Tools**:
  - Python (3.9+), TensorFlow (2.10+), PyTorch (1.12+).
  - Hugging Face Transformers (Whisper, BERT).
  - ONNX Runtime (1.12+), Librosa (0.9+).
  - AWS SageMaker, PostgreSQL (pgvector), Redis.
  - Pytest, JMeter, GitHub Actions.
- **Data Sources**:
  - CrossWire Bible Society (KJV/WEB, CC0).
  - LibriVox, SermonAudio (public-domain sermons, CC0).
  - Freesound (noise clips, CC0).
- **Documentation**:
  - Hugging Face: https://huggingface.co/docs/transformers/model_doc/whisper
  - ONNX Runtime: https://onnxruntime.ai/docs/
  - AWS SageMaker: https://docs.aws.amazon.com/sagemaker/
- **Team Contacts**:
  - Backend Developer: API integration, database queries.
  - Frontend Developer: WebSocket, JSON format.
  - DevOps Engineer: SageMaker, Redis, CloudWatch.
  - Product Manager: Data sourcing, pilot testing.

#### 9. Implementation Timeline

- **Month 3**:
  - Collect sermon audio (~10 hours), verse matching dataset (~10,000 pairs).
  - Preprocess data (Librosa, BERT embeddings).
  - Store in S3, PostgreSQL (embeddings).
- **Month 4**:
  - Fine-tune Whisper (WER <10%, <500ms).
  - Fine-tune BERT (95%+ top-10 accuracy, <500ms).
  - Implement regex for explicit references (<100ms).
  - Optimize models with ONNX/INT8.
- **Month 5**:
  - Build inference pipeline (FastAPI, WebSocket).
  - Integrate with backend (Node.js, PostgreSQL, Redis).
  - Test pipeline (unit tests, 1,000 samples).
- **Month 6**:
  - Implement feedback API and storage (`Feedback` table).
  - Develop retraining script (triplet loss, SageMaker).
  - Conduct end-to-end tests (JMeter, church simulation).
- **Month 7**:
  - Develop offline models (Whisper-tiny, DistilBERT, ONNX/INT8).
  - Integrate with Electron (SQLite, Python-shell).
  - Test offline mode (Core i5, 85%+ accuracy).
  - Run first retraining job (pilot feedback).
- **Month 8**:
  - Conduct pilot testing (5–10 churches, ~1,000 selections).
  - Optimize pipeline (latency, memory, cost).
  - Finalize documentation (GitHub).
  - Deliver test reports and benchmarks.

### Conclusion
This guide equips the AI/ML Developer to build a robust AI pipeline for the VerseProjection MVP, enabling real-time Bible verse detection with <1-second processing, 90%+ accuracy, and 95%+ top-10 match inclusion. The pipeline leverages Whisper for transcription, BERT for semantic matching, regex for

explicit references, and ONNX for optimization, with a feedback loop to improve accuracy by 5% in 3 months. Offline support via lightweight models ensures accessibility in low-internet environments. By integrating with backend, frontend, and cloud systems, and testing rigorously, the developer will deliver a scalable solution