

LiveCaps Multi-Language Transcription Architecture

Executive Summary

LiveCaps implements a sophisticated **parallel WebSocket architecture** for multi-language speech recognition. When users select multiple spoken languages, the system creates separate Deepgram connections for each language, processes the same audio through all of them simultaneously, and uses a **confidence-based winner selection algorithm** to choose the best transcription result.

Table of Contents

1. [Overview](#)
 2. [Architecture Diagram](#)
 3. [Core Components](#)
 4. [How It Works - Step by Step](#)
 5. [Single vs Multi Mode](#)
 6. [Confidence-Based Winner Selection](#)
 7. [Audio Distribution](#)
 8. [Translation Pipeline](#)
 9. [Key Files Reference](#)
-

1. Overview

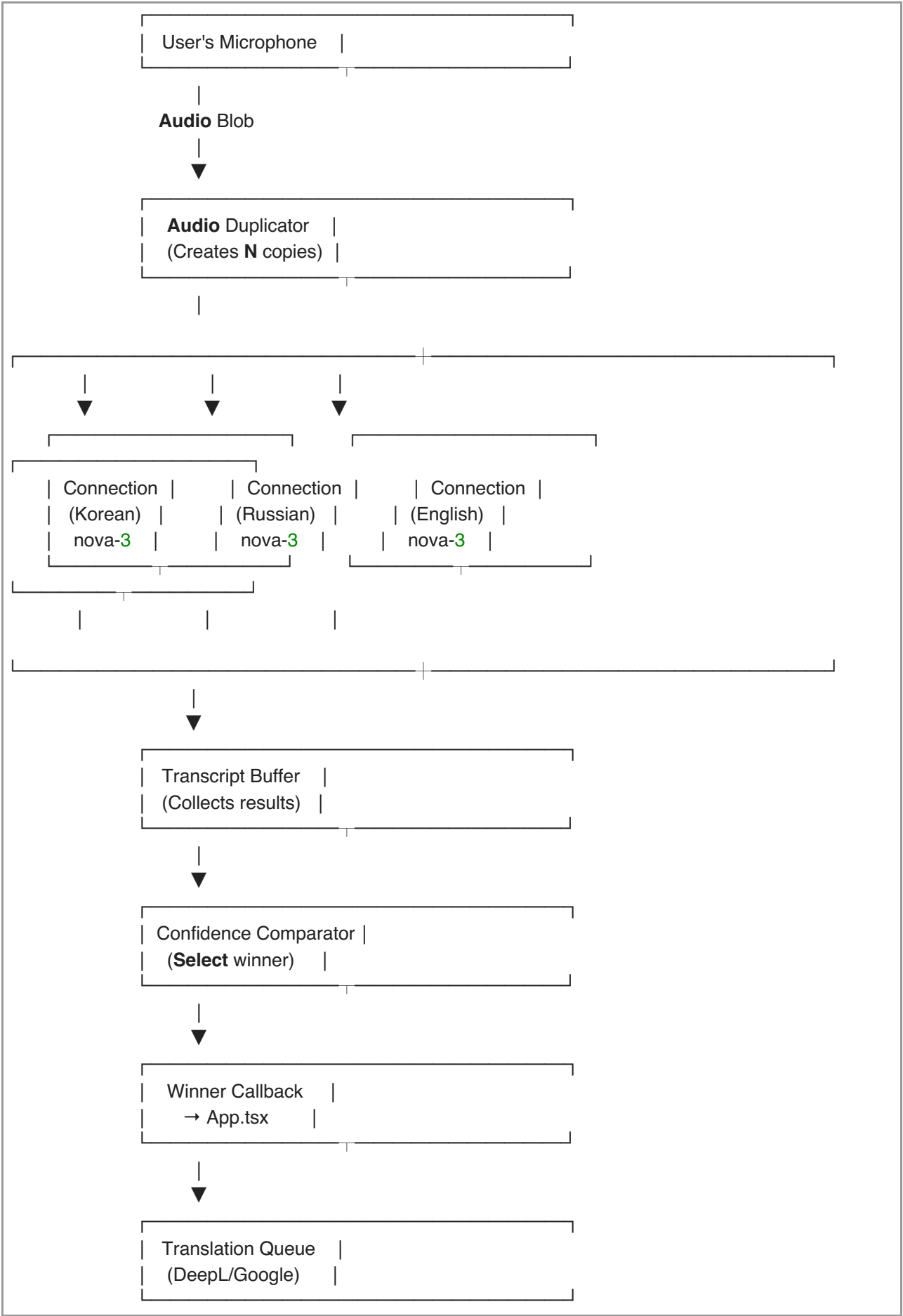
The Problem

Traditional speech-to-text systems require you to specify ONE source language. But what if:

- A speaker switches between languages (code-switching)?
- Multiple speakers use different languages?
- You're unsure which language will be spoken?

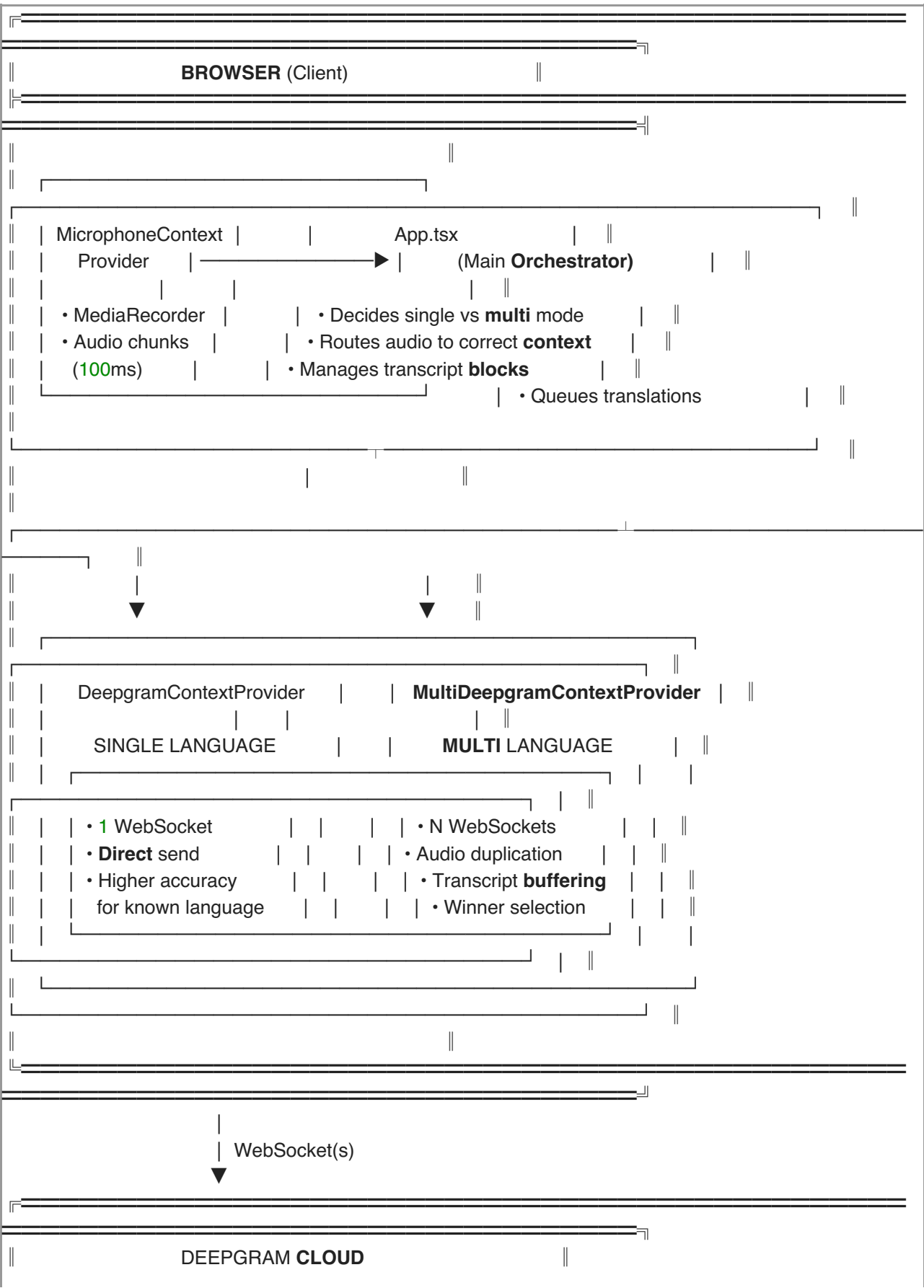
The Solution

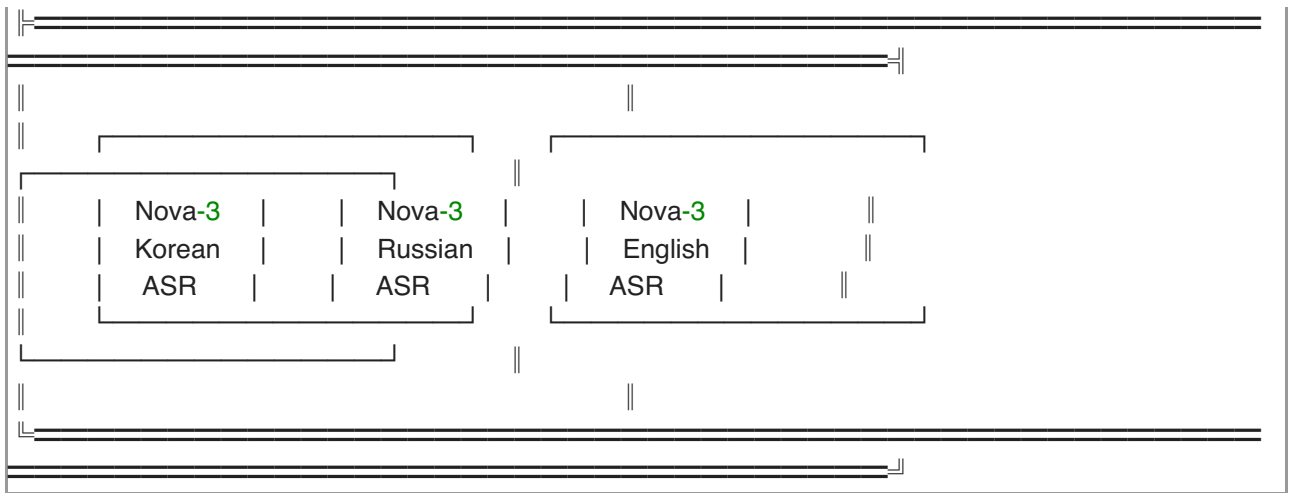
LiveCaps solves this by running **N parallel Deepgram connections** (one per selected language) and comparing their outputs in real-time:



2. Architecture Diagram

Data Flow





3. Core Components

3.1 MultiDeepgramContextProvider

Location: app/context/MultiDeepgramContextProvider.tsx

This is the heart of the multi-language system. It:

1. **Creates N parallel connections** to Deepgram
2. **Manages connection lifecycle** (open, close, error handling)
3. **Distributes audio** to all connections simultaneously
4. **Buffers transcript results** from all connections
5. **Selects the winner** based on confidence scores
6. **Emits winner events** to consuming components

```
// Key state managed by MultiDeepgramContextProvider
const [connections, setConnections] = useState<Map<string, DeepgramConnection>>>(new Map());
const [overallState, setOverallState] = useState<LiveConnectionState>(CLOSED);

// Key methods exposed
connectToDeepgram(languages: string[]) // Creates N connections
disconnectFromDeepgram() // Closes all connections
sendAudioToAll(audioData: Blob) // Distributes audio
onWinnerSelected(callback) // Subscribe to winner events
```

3.2 DeepgramConnection Interface

Each connection in the pool has this structure:

```
interface DeepgramConnection {
  id: string;           // "connection-ko", "connection-ru"
  language: string;     // "ko", "ru", "en"
  client: LiveClient;   // Deepgram SDK WebSocket client
  state: LiveConnectionState; // CLOSED, CONNECTING, OPEN
  lastTranscript: string; // Most recent text
  lastConfidence: number; // 0.0 - 1.0
  errorCount: number;    // For circuit breaking
  isHealthy: boolean;    // Circuit breaker status
  lastActivityTimestamp: number; // For health monitoring
}
```

3.3 TranscriptBuffer

Location: app/utils/confidenceComparison.ts

A time-windowed buffer that collects results from all connections before comparison:

```
class TranscriptBuffer {
  private buffer: Map<string, BufferedTranscript>;
  private windowMs: number; // Default: 50ms
  private expectedConnections: number;

  addResult(result: TranscriptResult): void;
  isWindowComplete(): boolean;
  getResults(): TranscriptResult[];
  clear(): void;
}
```

Why buffer? Network latency means responses arrive at different times. The buffer waits up to 50ms to collect all responses before comparing.

3.4 Audio Duplicator

Location: app/utils/audioDuplication.ts

Creates identical copies of audio blobs for each connection:

```
async function duplicateAudioBlob(blob: Blob, count: number): Promise<Blob[]> {
  const arrayBuffer = await blob.arrayBuffer();
  return Array(count).fill(null).map(() =>
    new Blob([arrayBuffer.slice(0)], { type: blob.type })
  );
}
```

4. How It Works - Step by Step

Step 1: Mode Selection

When the app loads, it determines the transcription mode:

```
// In App.tsx
const isMultiMode = transcriptionMode === "multi-detect"
    && sessionLanguages.spoken.length > 1;
```

- **Single Mode:** 1 spoken language → use DeepgramContextProvider
- **Multi Mode:** 2+ spoken languages → use MultiDeepgramContextProvider

Step 2: Connection Establishment

When entering multi-mode:

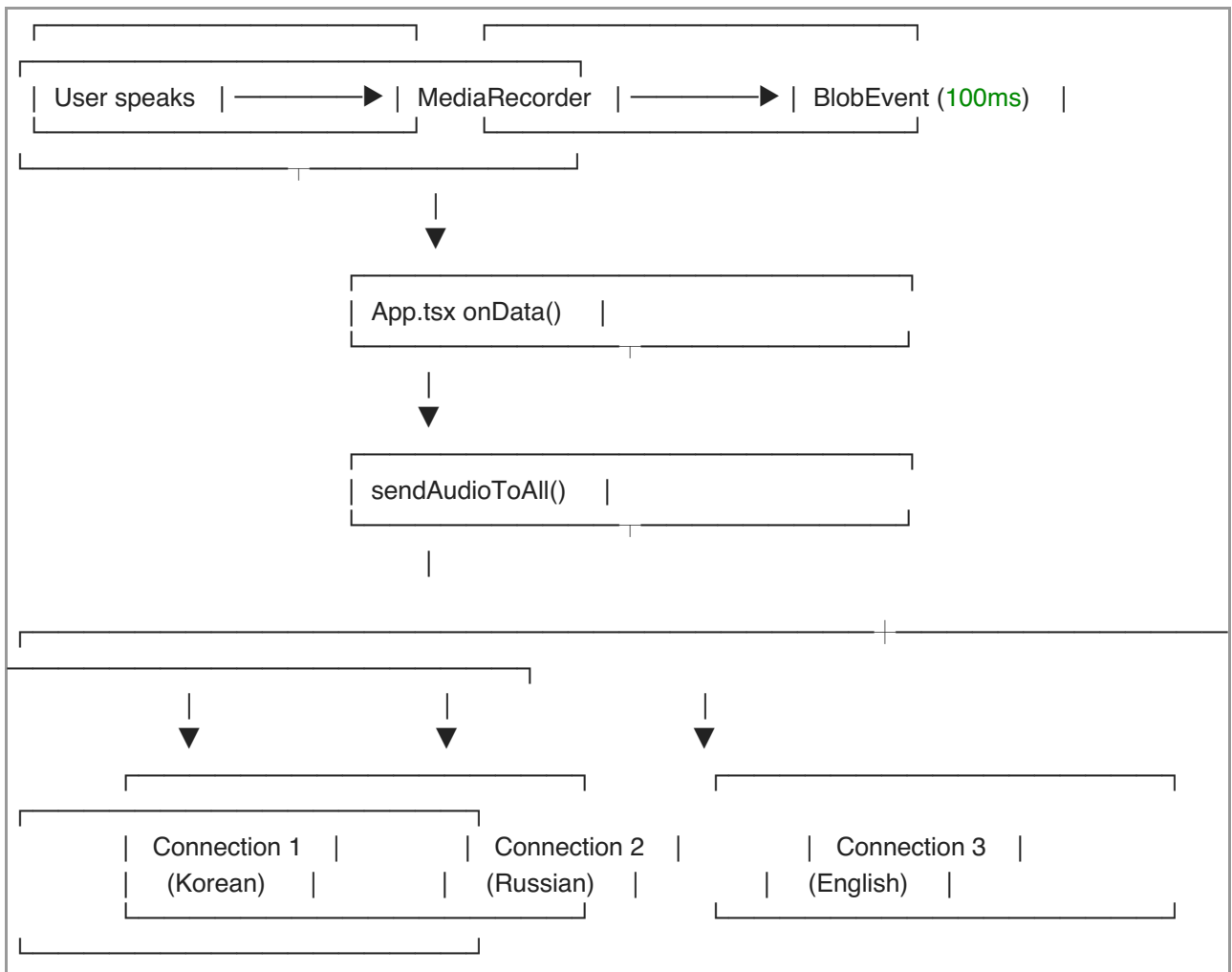
```
// App.tsx triggers connection
await multiContext.connectToDeepgram(["ko", "ru"]);

// MultiDeepgramContextProvider creates connections
for (const language of languages) {
    const connection = await createConnection(language, apiKey);
    connections.set(connection.id, connection);
}
```

Each connection uses **Nova-3** with language-specific configuration:

```
const options = {
  model: "nova-3",
  language: language,    // "ko", "ru", etc.
  interim_results: true,
  smart_format: true,
  punctuate: true,
  endpointing: 100,
  utterance_end_ms: 1500,
  vad_events: true,
};
```

Step 3: Audio Capture & Distribution



Step 4: Transcript Collection

Each connection receives its own transcription:

// Same audio produces different results:

Connection [ko]: "안녕하세요" (confidence: 0.95)

Connection [ru]: "а́нхасео" (confidence: 0.42)

Connection [en]: "an young ha say yo" (confidence: 0.38)

Step 5: Winner Selection

The TranscriptBuffer collects results within a 50ms window:

```
transcriptBuffer.addResult({
  connectionId: "connection-ko",
  language: "ko",
  transcript: "안녕하세요",
  confidence: 0.95,
  isFinal: true,
  timestamp: Date.now()
});

// When window completes or times out:
const winner = selectWinnerByConfidence(buffer.getResults());
```

Step 6: Winner Callback

The winner is emitted to App.tsx:

```
// In App.tsx
const cleanup = multiContext.onWinnerSelected(handleWinnerTranscript);

function handleWinnerTranscript(winner: WinnerTranscript) {
  // winner.transcript: "안녕하세요"
  // winner.language: "ko"
  // winner.confidence: 0.95

  // Create transcript block and queue translations
}
```

5. Single vs Multi Mode

Decision Logic

```
// In App.tsx
const isMultiMode = transcriptionMode === "multi-detect"
  && sessionLanguages.spoken.length > 1;

const connectionState = isMultiMode
  ? multiContext.overallState
  : singleContext.connectionState;
```

Comparison Table

Aspect	Single Mode	Multi Mode
WebSocket Connections	1	N (one per language)
API Calls	1x	Nx
Latency	Lower	Higher (buffering)
Accuracy	Higher for known lang	Best across all langs
Use Case	Know the speaker's language	Code-switching, unknown lang
Cost	1x	Nx

When to Use Each

Single Mode (Recommended when possible):

- You know what language will be spoken
- Single speaker with one language
- Cost-sensitive applications

Multi Mode (For complex scenarios):

- Code-switching (mixing languages)
- Multiple speakers with different languages
- Unsure which language will be spoken

6. Confidence-Based Winner Selection

The Algorithm

```
function selectWinnerByConfidence(results: TranscriptResult[]): WinnerTranscript {
  // 1. Filter out low-confidence results
  const validResults = results.filter(r =>
    r.confidence >= MIN_CONFIDENCE_THRESHOLD && // 0.3
    r.transcript.trim().length > 0
  );

  // 2. Find highest confidence
  let winner = validResults[0];
  for (const result of validResults) {
    if (result.confidence > winner.confidence) {
      winner = result;
    }
  }

  // 3. Return winner with metadata
  return {
    connectionId: winner.connectionId,
    language: winner.language,
    transcript: winner.transcript,
    confidence: winner.confidence,
    timestamp: Date.now(),
    allResults: results
  };
}
```

Configuration

```
const CONFIDENCE_CONFIG = {
  BUFFER_WINDOW_MS: 50,           // Wait time for all responses
  MIN_CONFIDENCE_THRESHOLD: 0.3,  // Minimum valid confidence
  SIGNIFICANT_DIFFERENCE_THRESHOLD: 0.05, // Margin for "clear" winner
  MAX_WAIT_TIME_MS: 100           // Force decision timeout
};
```

Example Comparison

Audio: "Привет, как дела?" (Russian: "Hello, how are you?")

Results received:

Language	Transcript	Confidence	
Russian	"Привет, как дела?"	0.94	← WINNER
Korean	"프리벳 까그 딜라"	0.31	
English	"pre vet kak dela"	0.45	

Winner: Russian (confidence 0.94)

7. Audio Distribution

The Challenge

WebSocket connections expect independent audio streams. Sending the same Blob object to multiple connections could cause issues.

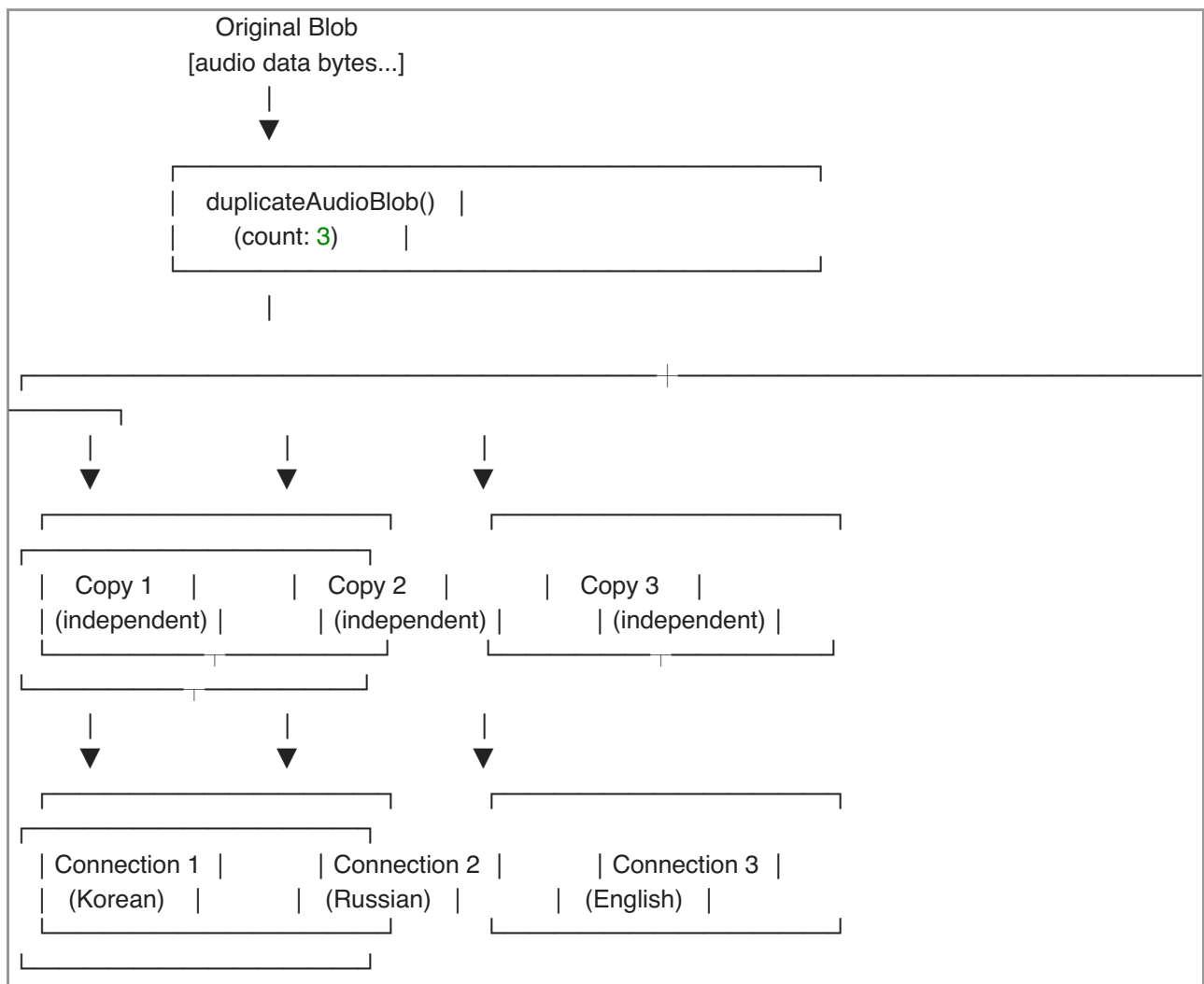
The Solution: Audio Duplication

```
// app/utils/audioDuplication.ts

export async function duplicateAudioBlob(
  blob: Blob,
  count: number
): Promise<Blob[]> {
  // Convert to ArrayBuffer (raw bytes)
  const arrayBuffer = await blob.arrayBuffer();

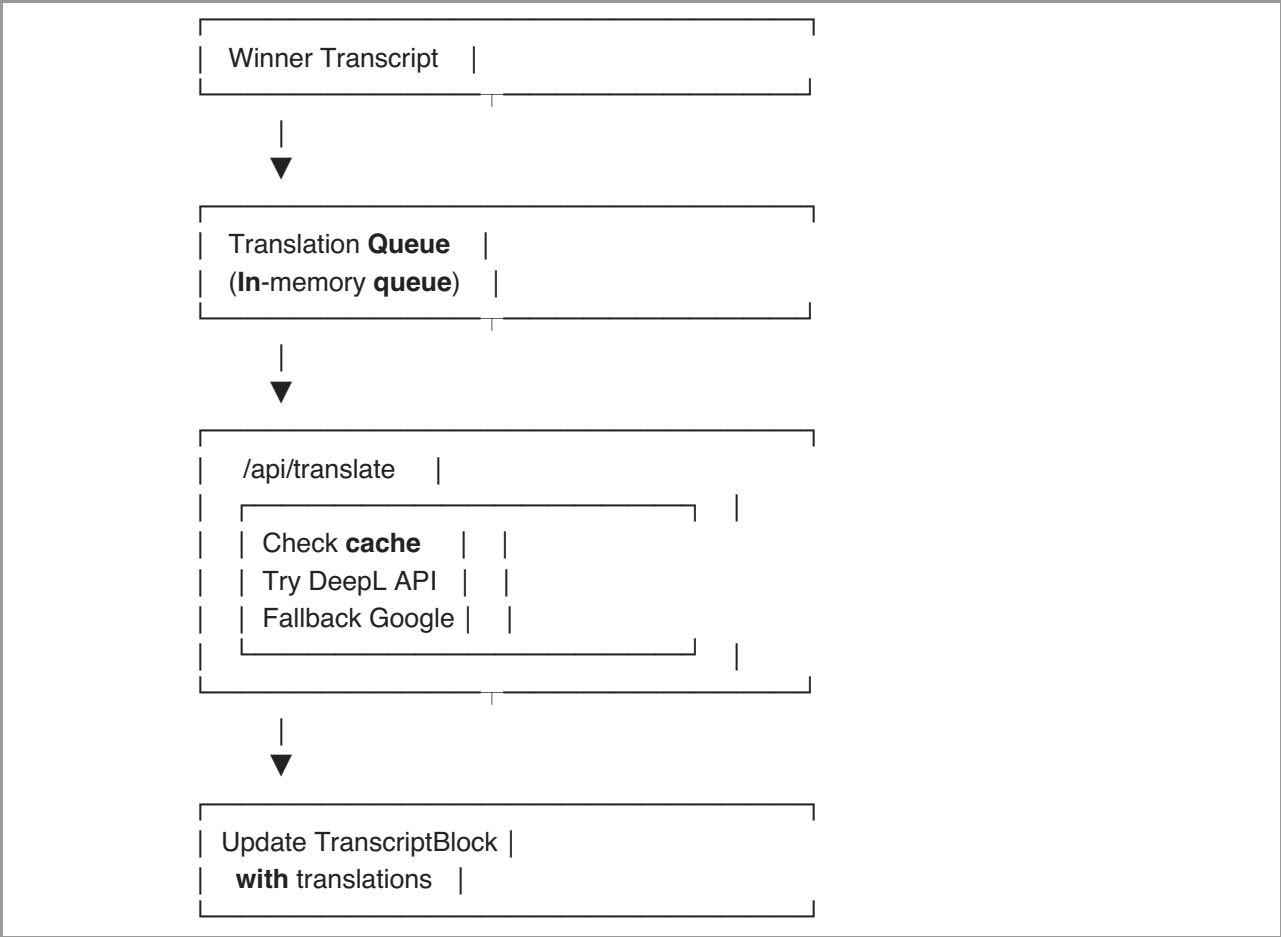
  // Create independent copies
  return Array(count)
    .fill(null)
    .map(() => new Blob(
      [arrayBuffer.slice(0)], // slice(0) creates a copy
      { type: blob.type }    // Preserve MIME type
    ));
}
```

Why This Matters



8. Translation Pipeline

After the winner is selected, the transcript goes through translation:



TranscriptBlock Structure

```
interface TranscriptBlock {
  id: string;
  original: {
    text: string;
    detectedLanguage: string;
    confidence: number;
  };
  translations: [
    {
      language: string; // "en", "ru"
      text: string; // Translated text
      cached: boolean;
    }
  ];
  timestamp: number;
}
```

9. Key Files Reference

File	Purpose
app/context/MultiDeepgramContextProvider.tsx	Multi-connection management, winner selection
app/context/DeepgramContextProvider.tsx	Single connection management
app/context/MicrophoneContextProvider.tsx	MediaRecorder, audio capture
app/components/App.tsx	Main orchestrator, mode switching, UI

app/utils/confidenceComparison.ts	Winner selection algorithm, buffer
app/utils/audioDuplication.ts	Audio blob copying
app/types/multiDeepgram.ts	TypeScript interfaces
app/services/translationService.ts	Translation API integration
app/api/translate/route.ts	Translation endpoint (DeepL + Google)
app/api/authenticate/route.ts	Deepgram API key provider

Summary

LiveCaps multi-language transcription works by:

1. **Creating parallel WebSocket connections** to Deepgram (one per language)
2. **Duplicating audio blobs** and sending identical copies to each connection
3. **Buffering transcript results** within a 50ms window
4. **Selecting the winner** based on highest confidence score
5. **Passing the winner** to the translation pipeline

This architecture enables real-time transcription of code-switched speech or unknown languages, at the cost of increased API usage proportional to the number of selected languages.

Document generated for LiveCaps project - January 2026